Universidad de Oviedo

Código fuente del Trabajo Fin de Máster realizado por

Sonia Madero García

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

DESARROLLO DE UNA APLICACIÓN SOFTWARE PARA EL SEGUIMIENTO Y ESTIMACIÓN DE LA TEMPERATURA DE CALENTAMIENTO DE PALANQUILLAS DE ACERO INOXIDABLE EN UN HORNO

FEBRERO 2016

ÍNDICE

1 PROYECTO ALTERNATIVA AL HOT

1.1 COM

1.1.1 Clase DatoLaminación

```
package com;
/**
 * Clase que permite acceder y actualizar los campos de la base de datos opc de
 * palanquillas
 *
 * @author SONIA
 */
public class DatoLaminacion
{

   /**
    * Enumeración de los campos del dato
    */
   public enum tipoDato
   {

       TIPO_ACERO, TIPO_PALANQUILLA, PESO, ALTO_PALANQUILLA,
       ANCHO_PALANQUILLA, LONGITUD_PALANQUILLA, MEDIDA_PRODUCTO,
       TIPO_PRODUCTO, LONGITUD_CORTE1, LONGITUD_CORTE2, CORTE_ROLLO,
       HORA_MOVIMIENTO, PESOFINAL, CALIDAD, ACABADO, PROCESO_LAMINACION,
       PROCESO_ESPECIAL_INTERNO,PROCESO_ESPECIAL_ACERA,
       PROCESO_ESPECIAL_UNICO, TOLERANCIA_MAS,
       TOLERANCIA_MENOS,CONTROL_DIMENSIONAL
};


   /**
    * Enumeración de los campos del programa
    */
   public enum tipoPrograma
   {

      TIPO_ACERO, MEDIDA_PRODUCTO, TIPO_PRODUCTO, TIPO_PALANQUILLA,
      LONGITUD_CORTE, COLADA
   };



   /**
    * Enumeración de los campos del dato de un paquete
    */
   public enum tipoPaquete
   {

      COLADA, MEDIDA_PRODUCTO, TIPO_PRODUCTO, LONGITUD_CORTE1,
      LONGITUD_CORTE2, TIPO_ACERO, HORA_MOVIMIENTO, CALIDAD, PESOFINAL,
      POSICIONPARQUE,CONTROL_DIMENSIONAL
   };
```

```
/**
 * Enumeración de los campos de la variable
 */
public enum tipoVar
{

    ABCISA, TEMPERATURA_SUPERIOR, TEMPERATURA_INFERIOR,
    TEMPERATURA_CORAZON
};

/**
 * Método que inserta un campo en un dato
 *
 * @param tipo campo del dato
 * @param dato cadena completa original del dato
 * @param valor valor del campo
 * @return Devuelve cadena completa final del dato
 */
static public String ModificaDatoPalanquilla(tipoDato tipo, String dato, String valor)
{

   String[] tokens = dato.split(";", -1);
   String resultado = "";

   if (tokens.length < tipoDato.values().length)
   {
      for (int i = tokens.length; i <= tipoDato.values().length; i++)
      {
         dato += "0;";
      }
      tokens = dato.split(";");
   }

   if (tipo == tipoDato.TIPO_ACERO)
   {
      String temp = "000";
      try
      {
         temp = String.format("%03d", Integer.valueOf(valor));
      }
      catch (NumberFormatException e)
      {
      }
      tokens[0] = temp;
   }
   else if (tipo == tipoDato.TIPO_PALANQUILLA)
   {
      tokens[1] = valor;
   }
   else if (tipo == tipoDato.PESO)
   {
      String temp = "0000";
      try
      {
```

```
            temp = String.format("%04d", Integer.valueOf(valor));
        }
        catch (NumberFormatException e)
        {
        }
        tokens[2] = temp;
    }
    else if (tipo == tipoDato.ALTO_PALANQUILLA)
    {
        String temp = "0000";
        try
        {
            temp = String.format("%04d", Integer.valueOf(valor));
        }
        catch (NumberFormatException e)
        {
        }
        tokens[3] = temp;
    }
    else if (tipo == tipoDato.ANCHO_PALANQUILLA)
    {
        String temp = "0000";
        try
        {
            temp = String.format("%04d", Integer.valueOf(valor));
        }
        catch (NumberFormatException e)
        {
        }
        tokens[4] = temp;
    }
    else if (tipo == tipoDato.LONGITUD_PALANQUILLA)
    {
        String temp = "0000";
        try
        {
            temp = String.format("%04d", Integer.valueOf(valor));
        }
        catch (NumberFormatException e)
        {
        }
        tokens[5] = temp;
    }
    else if (tipo == tipoDato.MEDIDA_PRODUCTO)
    {
        String temp = "";
        String[] dim = valor.split("x");
        for (int i = 0; i < dim.length; i++)
        {
            try
            {
                if (i > 0)
                {
                    temp += "x";
                }
```

```
            temp += String.format("%07.2f", Double.valueOf(dim[i].replace(",", "."))).replace(".", ",");
         }
         catch (NumberFormatException e)
         {
            temp += "0";
         }
      }
      tokens[6] = temp;
   }
   else if (tipo == tipoDato.TIPO_PRODUCTO)
   {
      tokens[7] = valor;
   }
   else if (tipo == tipoDato.LONGITUD_CORTE1)
   {
      String temp = "00000";
      try
      {
         temp = String.format("%05d", Integer.valueOf(valor));
      }
      catch (NumberFormatException e)
      {
      }
      tokens[8] = temp;
   }
   else if (tipo == tipoDato.LONGITUD_CORTE2)
   {
      String temp = "00000";
      try
      {
         temp = String.format("%05d", Integer.valueOf(valor));
      }
      catch (NumberFormatException e)
      {
      }
      tokens[9] = temp;
   }
   else if (tipo == tipoDato.CORTE_ROLLO)
   {
      tokens[10] = valor;
   }
   else if (tipo == tipoDato.HORA_MOVIMIENTO)
   {
      tokens[11] = valor;
   }
   else if (tipo == tipoDato.PESOFINAL)
   {
      String temp = "0000";
      try
      {
         temp = String.format("%04d", Integer.valueOf(valor));
      }
      catch (NumberFormatException e)
      {
      }
```

```
            tokens[12] = temp;
        }
        else if (tipo == tipoDato.CALIDAD)
        {
            tokens[13] = valor;
        }
        else if (tipo == tipoDato.ACABADO)
        {
            tokens[14] = valor;
        }
        else if (tipo == tipoDato.PROCESO_LAMINACION)
        {
            tokens[15] = valor;
        }
        else if (tipo == tipoDato.PROCESO_ESPECIAL_INTERNO)
        {
            tokens[16] = valor;
        }
        else if (tipo == tipoDato.PROCESO_ESPECIAL_ACERIA)
        {
            tokens[17] = valor;
        }
        else if (tipo == tipoDato.PROCESO_ESPECIAL_UNICO)
        {
            tokens[18] = valor;
        }
        else if (tipo == tipoDato.TOLERANCIA_MENOS)
        {
            tokens[19] = valor;
        }
        else if (tipo == tipoDato.TOLERANCIA_MAS)
        {
            tokens[20] = valor;
        }
        else if (tipo == tipoDato.CONTROL_DIMENSIONAL)
        {
            tokens[21] = valor;
        }
        else
        {
            return "";
        }
        for (int i = 0; i < tokens.length && i < tipoDato.values().length; i++)
        {
            resultado += tokens[i] + ";";
        }
        return resultado;
    }

    /**
     * Método que inserta un campo en un identificador de lote
     *
     * @param tipo campo del lote
     * @param programa cadena completa original del lote
     * @param valor valor del campo
```

```
 * @return Devuelve cadena completa final del lote
 */
static public String ModificaLote(tipoPrograma tipo, String programa, String valor)
{
   String resultado = "";
   if (programa.equals(""))
   {
      programa = "0;";
   }
   String[] tokens = programa.split(";");
   if (tokens.length < 6)
   {
      for (int i = tokens.length; i < 6; i++)
      {
         programa += "0;";
      }
      tokens = programa.split(";");
   }

   if (tipo == tipoPrograma.COLADA)
   {
      tokens[0] = valor;
   }
   else if (tipo == tipoPrograma.TIPO_ACERO)
   {
      String temp = "000";
      try
      {
         temp = String.format("%03d", Integer.valueOf(valor));
      }
      catch (NumberFormatException e)
      {
      }
      tokens[5] = temp;
   }
   else if (tipo == tipoPrograma.MEDIDA_PRODUCTO)
   {
      String temp = "";
      String[] dim = valor.split("x");
      for (int i = 0; i < dim.length; i++)
      {
         try
         {
            if (i > 0)
            {
               temp += "x";
            }
            //temp += String.format("%07.2f", Double.valueOf(dim[i].replace(",", "."))).replace(".", ",");
            temp += String.format("%.2f", Double.valueOf(dim[i].replace(",", "."))).replace(".", ",");
         }
         catch (NumberFormatException e)
         {
            temp += "0";
         }
      }
```

```
            tokens[1] = temp;
        }
        else if (tipo == tipoPrograma.TIPO_PRODUCTO)
        {
            tokens[2] = valor;
        }
        else if (tipo == tipoPrograma.LONGITUD_CORTE)
        {
            String temp = "";
            String[] mc = valor.split("-");
            for (int i = 0; i < mc.length; i++)
            {
                try
                {
                    if (i == 0)
                    {
                        tokens[3] = String.format("%05d", Integer.valueOf(mc[i]));
                    }
                    else if (i == 1)
                    {
                        tokens[4] = String.format("%05d", Integer.valueOf(mc[i]));
                    }
                }
                catch (NumberFormatException e)
                {
                    if (i == 0)
                    {
                        tokens[3] = "00000";
                    }
                    else if (i == 1)
                    {
                        tokens[4] = "00000";
                    }
                }
            }
        }
        for (String token : tokens)
        {
            resultado += token + ";";
        }
        return resultado;
    }

    /**
     * Método que devuelve un campo especificado de un dato
     *
     * @param tipo campo del dato
     * @param dato cadena completa original del dato
     * @return Devuelve el campo especificado de un dato
     */
    static public String ObtieneDatoPalanquilla(tipoDato tipo, String dato)
    {
        if (dato.equals(""))
        {
            return "";
```

```
}

String[] tokens = dato.split(";");
String resultado = "";
int indice = -1;

if (tipo == tipoDato.TIPO_ACERO)
{
   indice = 0;
}
else if (tipo == tipoDato.TIPO_PALANQUILLA)
{
   indice = 1;
}
else if (tipo == tipoDato.PESO)
{
   indice = 2;
}
else if (tipo == tipoDato.ALTO_PALANQUILLA)
{
   indice = 3;
}
else if (tipo == tipoDato.ANCHO_PALANQUILLA)
{
   indice = 4;
}
else if (tipo == tipoDato.LONGITUD_PALANQUILLA)
{
   indice = 5;
}
else if (tipo == tipoDato.MEDIDA_PRODUCTO)
{
   indice = 6;
}
else if (tipo == tipoDato.TIPO_PRODUCTO)
{
   indice = 7;
}
else if (tipo == tipoDato.LONGITUD_CORTE1)
{
   indice = 8;
}
else if (tipo == tipoDato.LONGITUD_CORTE2)
{
   indice = 9;
}
else if (tipo == tipoDato.CORTE_ROLLO)
{
   indice = 10;
}
else if (tipo == tipoDato.HORA_MOVIMIENTO)
{
   indice = 11;
}
else if (tipo == tipoDato.PESOFINAL)
```

```
    {
        indice = 12;
    }
    else if (tipo == tipoDato.CALIDAD)
    {
        indice = 13;
    }
    else if (tipo == tipoDato.ACABADO)
    {
        indice = 14;
    }
    else if (tipo == tipoDato.PROCESO_LAMINACION)
    {
        indice = 15;
    }
    else if (tipo == tipoDato.PROCESO_ESPECIAL_INTERNO)
    {
        indice = 16;
    }
    else if (tipo == tipoDato.PROCESO_ESPECIAL_ACERIA)
    {
        indice = 17;
    }
    else if (tipo == tipoDato.PROCESO_ESPECIAL_UNICO)
    {
        indice = 18;
    }
    else if (tipo == tipoDato.TOLERANCIA_MENOS)
    {
        indice = 19;
    }
    else if (tipo == tipoDato.TOLERANCIA_MAS)
    {
        indice = 20;
    }
    else if (tipo == tipoDato.CONTROL_DIMENSIONAL)
    {
        indice = 21;
    }
    else
    {
        return "";
    }
    if (indice < 0 || indice >= tokens.length)
    {
        return "";
    }
    String res = tokens[indice];
    return res;
}

/**
 * Método que devuelve un campo especificado de un dato de paquete
 *
 * @param tipo campo del dato de paquete
```

```
 * @param dato cadena completa original del dato de paquete
 * @return Devuelve el campo especificado de un dato de paquete
 */
static public String ObtieneDatoPaquete(tipoPaquete tipo, String dato)
{
   if (dato.equals(""))
   {
      return "";
   }

   String[] tokens = dato.split(";");
   String resultado = "";
   int indice;

   if (tipo == tipoPaquete.COLADA)
   {
      indice = 0;
   }
   else if (tipo == tipoPaquete.MEDIDA_PRODUCTO)
   {
      indice = 1;
   }
   else if (tipo == tipoPaquete.TIPO_PRODUCTO)
   {
      indice = 2;
   }
   else if (tipo == tipoPaquete.LONGITUD_CORTE1)
   {
      indice = 3;
   }
   else if (tipo == tipoPaquete.LONGITUD_CORTE2)
   {
      indice = 4;
   }
   else if (tipo == tipoPaquete.TIPO_ACERO)
   {
      indice = 5;
   }
   else if (tipo == tipoPaquete.HORA_MOVIMIENTO)
   {
      indice = 6;
   }
   else if (tipo == tipoPaquete.CALIDAD)
   {
      indice = 7;
   }
   else if (tipo == tipoPaquete.PESOFINAL)
   {
      indice = 8;
   }
   else if (tipo == tipoPaquete.POSICIONPARQUE)
   {
      indice = 9;
   }
   else if (tipo == tipoPaquete.CONTROL_DIMENSIONAL)
```

```
    {
       indice = 10;
    }
    else
    {
       return "";
    }
    if (indice < 0 || indice >= tokens.length)
    {
       return "";
    }
    return tokens[indice];
 }

 /**
  * Método que inserta un campo en una variable
  *
  * @param tipo campo de la variable
  * @param dato cadena completa original de la variable
  * @return Devuelve cadena completa final de la variable
  */
 static public String ObtieneVarPalanquilla(tipoVar tipo, String dato)
 {
    if (dato.equals(""))
    {
       return "";
    }

    String[] tokens = dato.split(";");
    int indice = -1;

    if (tipo == tipoVar.ABCISA)
    {
       indice = 0;
    }
    else if (tipo == tipoVar.TEMPERATURA_SUPERIOR)
    {
       indice = 1;
    }
    else if (tipo == tipoVar.TEMPERATURA_INFERIOR)
    {
       indice = 2;
    }
    else if (tipo == tipoVar.TEMPERATURA_CORAZON)
    {
       indice = 3;
    }
    else
    {
       return "";
    }
    if (indice < 0 || indice >= tokens.length)
    {
       return " ";
    }
```

```
        String res = tokens[indice];
        return res;
    }

    /**
     * Método que devuelve un campo especificado de una variable
     * @param tipo campo de la variable
     * @param dato cadena completa original de la variable
     * @param valor valor del campo
     * @return Devuelve el campo especificado de una variable
     */
    static public String ModificaVarPalanquilla(tipoVar tipo, String dato, String valor)
    {
        String[] tokens = dato.split(";", -1);
        String resultado = "";

        if (tokens.length < tipoVar.values().length)
        {
            for (int i = tokens.length; i <= tipoVar.values().length; i++)
            {
                dato += "0;";
            }
            tokens = dato.split(";");
        }

        if (tipo == tipoVar.ABCISA)
        {
            tokens[0] = valor;
        }
        else if (tipo == tipoVar.TEMPERATURA_SUPERIOR)
        {
            tokens[1] = valor;
        }
        else if (tipo == tipoVar.TEMPERATURA_INFERIOR)
        {
            tokens[2] = valor;
        }
        else if (tipo == tipoVar.TEMPERATURA_CORAZON)
        {
            tokens[3] = valor;
        }
        else
        {
            return "";
        }
        for (int i = 0; i < tokens.length && i < tipoDato.values().length; i++)
        {
            resultado += tokens[i] + ";";
        }
        return resultado;
    }
}
```

1.2 COM.ESTADO

1.2.1 Clase Registro

```
package com.estado;

import com.DatoLaminacion;
import com.Global;
import com.interfaz.laminacion.Estado;
import com.opc.OPCUA;
import com.prosysopc.ua.ServiceException;
import com.prosysopc.ua.StatusException;
import com.prosysopc.ua.client.AddressSpaceException;

/**
 * Clase que implementa un registro de la base de datos opc de palanquillas
 *
 * @author SONIA
 */
public class Registro
{

    /**
     * Identificador del registro
     */
    public String ID;
    /**
     * Índice de programa
     */
    public int set;
    /**
     * Índice de estado
     */
    public int sta;
    /**
     * Campos de dato
     */
    public String dat;
    /**
     * Índice en la tabla
     */
    private final int indice;
    /**
     * Instante del registro
     */
    public long ts;
    /**
     * Identificador del lote del registro
     */
    private String lote;
    /**
     * Identificador del programa del registro
     */
    private String programa;
    /**
```

```
 * Identificador del estado del registro
 */
private String estado;
/**
 * Identificador abreviado del estado del registro
 */
private String estadoAbrv;
/**
 * Nombre de la tabla de registros
 */
private final String nombre;
/**
 * Conexión OPC UA
 */
private final OPCUA opc;
/**
 * Dirección raíz de los tags opc que soportan el registro
 */
private final String raiz;
/**
 * Campos de variable
 */
public String var;
/**
 * Indica si el registro incluye campo de variables
 */
private final boolean isVar;

/**
 * Constructor de Registro
 *
 * @param i Índice del registro
 * @param nombre Nombre de la tabla de registros
 * @param opc Objeto de la conexión OPC
 * @param raiz Dirección raíz de los tags opc que soportan el registro
 * @param isVar Indica si el registro incluye campo de variables
 */
public Registro(int i, String nombre, OPCUA opc, String raiz, boolean isVar)
{
   ID = "";
   set = 0;
   sta = 0;
   dat = "";
   indice = i;
   lote = "";
   programa = "";
   estado = "";
   var = "";
   this.isVar = isVar;
   this.nombre = nombre;
   this.opc = opc;
   this.raiz = raiz;
}

/**
```

```
    * Método que devuelve la cadena del lote
    *
    * @return Devuelve la cadena del lote
    */
   public String getIDLote()
   {
      return lote;
   }

   /**
    * Método que actualiza la cadena del lote
    *
    */
   public void ActualizaLoteDato()
   {
      if (!dat.equals(""))
      {
         if (nombre.equals("paquetes"))
         {
          lote =DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.COLADA, lote,
          DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.COLADA, dat));
          lote=DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.MEDIDA_PRODUCTO,lote
          ,DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.MEDIDA_PRODUCTO,
          dat));
          lote = DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.TIPO_PRODUCTO, lote,
          DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.TIPO_PRODUCTO, dat));
          lote=DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.LONGITUD_CORTE, lote,
          DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.LONGITUD_CORTE1, dat)
          + "-"+
          DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.LONGITUD_CORTE2,
          dat));
          lote = DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.TIPO_ACERO, lote,
          DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.TIPO_ACERO, dat));
          }
          else
          {
             lote = DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.MEDIDA_PRODUCTO,
lote, DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.MEDIDA_PRODUCTO, dat));
             lote = DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.TIPO_PRODUCTO, lote,
DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.TIPO_PRODUCTO, dat));
             lote = DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.LONGITUD_CORTE,
lote,

DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.LONGITUD_CORTE1, dat) + "-" +
DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.LONGITUD_CORTE2, dat));
             lote = DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.TIPO_ACERO, lote,
DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.TIPO_ACERO, dat));
          }
      }
   }
```

```
/**
 * Método que actualiza la colada en la cadena del lote
 *
 */
public void ActualizaLoteColada()
{
   if (ID.length() >= 9)
   {
      lote = DatoLaminacion.ModificaLote(DatoLaminacion.tipoPrograma.COLADA, lote,
      ID.substring(2, 6));
   }
}

/**
 * Método que actualiza la cadena del estado en funciónd e su código
 * númerico
 *
 */
public void ActualizaNombreEstado()
{
   estado = Estado.getDescripcionEstadoCorto(sta);
}

/**
 * Método que devuelve la cadena del estado
 *
 * @return Devuelve la cadena del estado
 */
public String getNombreEstado()
{
   return estado;
}

/**
 * Método que actualiza la cadena del programa de una palanquilla
 *
 */
public void ActualizaNombreProgramaPalanquilla()
{
   String producto =
   DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.TIPO_PRODUCTO, dat);
   String str =
   DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.MEDIDA_PRODUCTO, dat);
   String medida = " ";
   if (!str.equals("") && str.contains(","))
   {
      String[] strDim = str.split("x");

      for (int i = 0; i < strDim.length; i++)
      {
         try
         {
            if (i > 0)
            {
               medida += "x";
```

```
                }
                medida += String.valueOf(Double.valueOf(strDim[i].replace(",", "."))).replace(".0", "");
            }
            catch (NumberFormatException ex)
            {
            }
        }
    }
    programa = producto + " " + medida;
}

/**
 * Método que devuelve la cadena del programa de un paquete
 *
 * @return Devuelve la cadena del programa de un paquete
 */
public String getNombreProgramaPaquete()
{
    String producto =
DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.TIPO_PRODUCTO, dat);
    if (producto.equals("A"))
    {
        return "Anulado";
    }
    else
    {
        String colada = DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.COLADA,
        dat);
        String str =
        DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.MEDIDA_PRODUCTO,
        dat);
        String medida = " ";
        if (!str.equals("") && str.contains(","))
        {
            String[] strDim = str.split("x");

            for (int i = 0; i < strDim.length; i++)
            {
                try
                {
                    if (i > 0)
                    {
                        medida += "x";
                    }
                    medida += String.valueOf(Double.valueOf(strDim[i].replace(",", "."))).replace(".0", "");
                }
                catch (NumberFormatException ex)
                {
                }
            }
        }
        String longitudCorte =
        DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.LONGITUD_CORTE1, dat)
        + "-" +
        DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.LONGITUD_CORTE2, dat);
```

```
        if (producto.equals("NO"))
        {
           return "ANULADO";
        }
        else
        {
           return producto + " " + medida + " " + colada + " " + longitudCorte;
        }
     }
  }

  /**
   * Método que actualiza la cadena del programa de un paquete
   *
   */
  public void ActualizaNombreProgramaPaquete()
  {
     programa = getNombreProgramaPaquete();
     String producto =
     DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.TIPO_PRODUCTO, dat);
     if (producto.equals("A"))
     {
        programa = "Anulado";
     }
     else
     {
         String colada = DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.COLADA,
         dat);
        String str =
        DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.MEDIDA_PRODUCTO,
        dat);
        String medida = " ";
        if (!str.equals("") && str.contains(","))
        {
           String[] strDim = str.split("x");

           for (int i = 0; i < strDim.length; i++)
           {
              try
              {
                 if (i > 0)
                 {
                    medida += "x";
                 }
                 medida += String.valueOf(Double.valueOf(strDim[i].replace(",", "."))).replace(".0", "");
              }
              catch (NumberFormatException ex)
              {
              }
           }
        }
         String longitudCorte =
         DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.LONGITUD_CORTE1, dat)
         + "-" +
         DatoLaminacion.ObtieneDatoPaquete(DatoLaminacion.tipoPaquete.LONGITUD_CORTE2, dat);
```

```
            if (producto.equals("NO"))
            {
               programa = "ANULADO";
            }
            else
            {
               programa = producto + " " + medida + " " + colada + " " + longitudCorte;
            }
         }
      }

      /**
       * Método que devuelve la cadena del programa
       *
       * @return Devuelve la cadena del programa
       */
      public String getPrograma()
      {
         return programa;
      }

      /**
       * Método que actualiza los campos de un registro
       *
       * @param identificacion Identificación del registro
       * @param programa Índice del programa
       * @param estado Índice del estado
       * @param dato Cadena de campos del dato
       * @param instante Instante del registro que permite ordenarlos en la tabla
       * @param var Cadena de campos de la variable
       */
synchronized void setRegistro(String identificacion, int programa, int estado, String dato, long instante,
String var)
      {
         setID(identificacion);
         setSet(programa);
         setSta(estado);
         setDat(dato);
         setTs(instante);
         if (isVar)
         {
            setVar(var);
         }
      }

      /**
       * Método que actualiza los campos de un registro
       *
       * @param programa Índice del programa
       * @param estado Índice del estado
       * @param dato Cadena de campos del dato
       * @param instante Instante del registro que permite ordenarlos en la tabla
       * @param var Cadena de campos de la variable
       */
      synchronized void setRegistro(int programa, int estado, String dato, long instante, String var)
```

```
{
   setSet(programa);
   setSta(estado);
   setDat(dato);
   setTs(instante);
   if (isVar)
   {
      setVar(var);
   }

}

/**
 * Método que devuelve la cadena de campos del dato
 *
 * @return Devuelve la cadena de campos del dato
 */
synchronized public String getDat()
{
   return dat;
}

/**
 * Método que actualiza la cadena de campos del dato
 *
 * @param dat Cadena de campos del dato
 */
public void setDat(String dat)
{
   try
   {
      opc.Escribe(raiz + ".dat" + String.format("%02d", indice + 1), dat);
      this.dat = dat;
      ActualizaLoteDato();
   }
   catch (ServiceException | AddressSpaceException | StatusException ex)
   {
      Global.info.Registra(ex);
   }
}

/**
 * Método que devuelve la identificación del registro
 *
 * @return Devuelve la identificación del registro
 */
synchronized public String getID()
{
   return ID;
}

/**
 * Método que actualiza la identificación del registro
 *
 * @param ID Identificación del registro
```

```
 */
public void setID(String ID)
{
   try
   {
      opc.Escribe(raiz + ".ID" + String.format("%02d", indice + 1), ID);
      this.ID = ID;
      ActualizaLoteColada();
   }
   catch (ServiceException | AddressSpaceException | StatusException ex)
   {
      Global.info.Registra(ex);
   }
}

/**
 * Método que devuelve el instante del registro
 *
 * @return Devuelve el instante del registro
 */
synchronized public long getTs()
{
   return ts;
}

/**
 * Método que actualiza el instante del registro
 *
 * @param ts Instante del registro
 */
public void setTs(long ts)
{

   try
   {
      opc.Escribe(raiz + ".ts" + String.format("%02d", indice + 1), String.valueOf(ts / 1000));
      this.ts = ts;
   }
   catch (ServiceException | AddressSpaceException | StatusException ex)
   {
      Global.info.Registra(ex);
   }
}

/**
 * Método que devuelve el índice del estado del registro
 *
 * @return Devuelve el índice del estado del registro
 */
synchronized public int getSta()
{
   return sta;
}

/**
```

```
 * Método que actualiza el índice del estado del registro
 *
 * @param sta Índice del estado del registro
 */
public void setSta(int sta)
{
   try
   {
      opc.Escribe(raiz + ".sta" + String.format("%02d", indice + 1), String.valueOf(sta));
      this.sta = sta;
   }
   catch (ServiceException | AddressSpaceException | StatusException ex)
   {
      Global.info.Registra(ex);
   }
}

/**
 * Método que devuelve el índice del programa del registro
 *
 * @return Devuelve el índice del programa del registro
 */
synchronized public int getSet()
{
   return set;
}

/**
 * Método que actualiza el índice del programa del registro
 *
 * @param set Índice del estado del registro
 */
public void setSet(int set)
{
   try
   {
      opc.Escribe(raiz + ".set" + String.format("%02d", indice + 1), String.valueOf(set));
      this.set = set;
   }
   catch (ServiceException | AddressSpaceException | StatusException ex)
   {
      Global.info.Registra(ex);
   }
}

/**
 * Método que devuelve el índice del registro
 *
 * @return Devuelve el índice del registro
 */
public int getIndice()
{
   return indice;
}
```

```
    /**
     * Método que actualiza la cadena de campos de la variable
     *
     * @param var Cadena de campos del dato
     */
    public void setVar(String var)
    {
        try
        {
            opc.Escribe(raiz + ".var" + String.format("%02d", indice + 1), var);
            this.var = var;
            ActualizaLoteDato();
        }
        catch (ServiceException | AddressSpaceException | StatusException ex)
        {
            Global.info.Registra(ex);
        }
    }

    /**
     * Método que devuelve la cadena de campos de la variable
     *
     * @return Devuelve la cadena de campos de la variable
     */
    synchronized public String getVar()
    {
        return var;
    }

}
```

1.2.2 Clase TrazaLaminación

```
package com.estado;

import com.Global;
import com.opc.ItemInfo;
import com.opc.OPCUA;
import java.io.File;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase que implementa una tabla de registros de la base de datos opc de
 * palanquillas
 *
 * @author SONIA
 */
public class TrazaLaminacion
{

    /**
     * número de registros
     */
    int num;
```

```
/**
 * Dirección raíz de los tags opc que soportan el registro
 */
public String raiz;
/**
 * Array de registros
 */
public final Registro[] registros;
/**
 * Nombre de la tabla
 */
public String nombre;
/**
 * Conexión OPC UA
 */
private final OPCUA opc;
/**
 * Array de items OPC
 */
private ItemInfo[] itemInfoArray;
/**
 * Separador directorios
 */
private final String separador;
/**
 * Indica si el registro incluye campo de variables
 */
private boolean isVar;

/**
 * Enumeración del orden de la tabla de registros
 */
public enum ordenTraza
{

   PRIMERO_MAS_ANTIGUO
};

/**
 * Método que modifica el instante de un registro para hacerlo más reciente
 *
 * @param i Índice del registro
 */
public void Rejuvenece(int i)
{
   Rejuvenece(i, registros[i].getTs());
}

/**
 * Método que modifica el instante de un registro tomando como base un
 * instante especificado para hacerlo más reciente. Sí encuentra algún
 * registro con el nuevo instante también lo rejuvenece para evitar colisión
 *
 * @param i Índice del registro
 * @param ts Instante de referencia
```

```
 */
public void Rejuvenece(int i, long ts)
{
   ts = ts / 1000;
   ts++;
   for (int j = 0; j < registros.length; j++)
   {
      if (registros[j].ts / 1000 == ts)
      {
         Rejuvenece(j);
         j = -1;
      }
   }
   registros[i].setTs(ts * 1000);
}

/**
 * Método que modifica el instante de un registro para hacerlo más antiguo
 *
 * @param i Índice del registro
 */
public void Envejece(int i)
{
   Envejece(i, registros[i].getTs());
}

/**
 * Método que modifica el instante de un registro tomando como base un
 * instante especificado para hacerlo más antiguo. Sí encuentra algún
 * registro con el nuevo instante también lo rejuvenece para evitar colisión
 *
 * @param i Índice del registro
 * @param ts Instante de referencia
 */
public void Envejece(int i, long ts)
{
   ts = ts / 1000;
   ts--;
   for (int j = 0; j < registros.length; j++)
   {
      if (registros[j].ts / 1000 == ts)
      {
         Envejece(j);
         j = -1;
      }
   }
   registros[i].setTs(ts * 1000);
}

/**
 * Constructor de la tabla de registros de la base de datos opc de
 * palanquillas
 *
 * @param n Número de registros
 * @param r Dirección raíz de los tags opc que soportan el registro
```

```
 * @param nmb Nombre de la tabla
 * @param opc Conexión OPC UA
 * @param isVar Indica si el registro incluye campo de variables
 */
public TrazaLaminacion(int n, String r, String nmb, OPCUA opc, boolean isVar)
{
   separador = System.getProperty("file.separator");
   this.opc = opc;
   registros = new Registro[n];
   this.isVar = isVar;
   for (int i = 0; i < n; i++)
   {
      registros[i] = new Registro(i, nmb, opc, r, isVar);
   }
   num = n;
   raiz = r;
   nombre = nmb;
}

/**
 * Método que devuelve el número de registros de la tabla
 *
 * @return Devuelve el número de registros de la tabla
 */
public int getSize()
{
   return num;
}

/**
 * Método que devuelve una lista ordenada de registros que corresponden a
 * alguno de los estados especificados
 *
 * @param orden Orden de la tabla de registros
 * @param filtroEstado Lista de cadenas de estado
 *
 * @return Devuelve la lista ordenada de registros
 */
public List<Registro> getRegistros(ordenTraza orden, List<String> filtroEstado)
{
   List<Registro> lista = new ArrayList<>();
   if (orden == ordenTraza.PRIMERO_MAS_ANTIGUO)
   {

      for (int i = MasAntiguo(), pal = 0;
            i >= 0 && i < num && pal < num;
            i = MasReciente(registros[i].getTs()), pal++)
      {
         if (filtroEstado == null)
         {
            lista.add(registros[i]);
         }
         else
         {
            for (String f : filtroEstado)
```

```
            {
               if (registros[i].getNombreEstado().contains(f))
               {
                  lista.add(registros[i]);
                  break;
               }
            }
         }
      }
   }
   return lista;
}

/**
 * Método que devuelve un registro según el índice especificado
 *
 * @param i Índice especificado
 *
 * @return Registro
 */
public Registro getRegistro(int i)
{
   if (i >= 0 && i < num)
   {
      return registros[i];
   }
   else
   {
      return null;
   }
}

/**
 * Método que devuelve el nombre de la tabla
 *
 * @return Nombre de la tabla
 */
public String getNombre()
{
   return nombre;
}

/**
 * Método que devuelve el array de items opc utilizados por la tabla
 *
 * @return Items opc de la tabla
 */
public ItemInfo[] getItemInfoArray()
{
   itemInfoArray = new ItemInfo[num * (isVar ? 6 : 5) + 1];
   int indice = 0;

   for (int i = 0; i < num; i++)
   {
itemInfoArray[indice++] = new ItemInfo(raiz + ".ID" + String.format("%02d", i + 1), true, -1);
```

```
itemInfoArray[indice++] = new ItemInfo(raiz + ".set" + String.format("%02d", i + 1), true, -1);
itemInfoArray[indice++] = new ItemInfo(raiz + ".sta" + String.format("%02d", i + 1), true, -1);
itemInfoArray[indice++] = new ItemInfo(raiz + ".dat" + String.format("%02d", i + 1), true, -1);
itemInfoArray[indice++] = new ItemInfo(raiz + ".ts" + String.format("%02d", i + 1), true, -1);
      if (isVar)
      {
         itemInfoArray[indice++] = new ItemInfo(raiz + ".var" + String.format("%02d", i + 1),
        true, -1);
      }
   }
   itemInfoArray[indice++] = new ItemInfo(raiz + ".bloqueado", true, -1);

   return itemInfoArray;
}

/**
 * Método que devuelve el nombre del registro más reciente de la tabla a
 * partir del índice de estado especificado
 *
 * @param sta Índice de estado especificado
 * @return Nombre del registro más reciente de la tabla
 */
public String PalanquillaMasReciente(int sta)
{
   int i;
   if (sta == 0)
   {
      i = MasReciente();
   }
   else
   {
      i = MasReciente(sta);
   }
   if (i > 0 && i < registros.length)
   {
      return registros[i].getID();
   }
   else
   {
      return "";
   }
}

/**
 * Método que devuelve el nombre del registro más antiguo de la tabla a
 * partir del índice de estado especificado
 *
 * @param sta Índice de estado especificado
 * @return Nombre del registro más reciente de la tabla
 */
public String PalanquillaMasAntigua(int sta)
{
   int i;
   if (sta == 0)
   {
```

```
            i = MasAntiguo();
        }
        else
        {
            i = MasAntiguo(sta);
        }
        if (i > 0 && i < registros.length)
        {
            return registros[i].getID();
        }
        else
        {
            return "";
        }
    }

    /**
     * Método que devuelve el índice del registro más antiguo de la tabla a
     * partir del instante especificado
     *
     * @param ref Instante de referencia
     * @return Índice del registro más antiguo de la tabla
     */
    public int MasAntiguo(long ref)
    {
        int indice = -1;
        long max = Long.MIN_VALUE;
        for (int i = 0; i < registros.length; i++)
        {
            if (registros[i].getID().equals("") || registros[i].getTs() >= ref || registros[i].getTs() == 0)
            {
                continue;
            }
            if (registros[i].getTs() > max)
            {
                max = registros[i].getTs();
                indice = i;
            }
        }
        return indice;
    }

    /**
     * Método que devuelve el índice del registro más antiguo de la tabla
     *
     * @return Índice del registro más antiguo de la tabla
     */
    public int MasAntiguo()
    {
        long min = Long.MAX_VALUE;
        int s = -1;
        for (int i = 0; i < registros.length; i++)
        {
            if (registros[i].getID().equals("") || registros[i].getTs() == 0)
            {
```

```
            continue;
        }
        if (registros[i].getTs() < min)
        {
            s = i;
            min = registros[i].getTs();
        }
    }
    return s;
}

/**
 * Método que devuelve el índice del registro más antiguo de la tabla con el
 * estado especificado
 *
 * @param estado Estado considerado
 * @return Índice del registro más antiguo de la tabla con el estado
 * especificado
 */
public int MasAntiguo(int estado)
{
    long min = Long.MAX_VALUE;
    int s = -1;
    for (int i = 0; i < registros.length; i++)
    {
        if (registros[i].getID().equals("") || registros[i].getTs() == 0 || registros[i].getSta() != estado)
        {
            continue;
        }
        if (registros[i].getTs() < min)
        {
            s = i;
            min = registros[i].getTs();
        }
    }
    return s;
}

/**
 * Método que devuelve el índice del registro más antiguo de la tabla con el
 * estado especificado, sin considerar barras de hierro
 *
 * @param estado Estado considerado
 * @return Índice del registro más antiguo de la tabla con el estado
 * especificado
 */
public int MasAntiguoSinHierro(int estado)
{
    long min = Long.MAX_VALUE;
    int s = -1;
    for (int i = 0; i < registros.length; i++)
    {
          if (registros[i].getID().equals("") || registros[i].getID().contains("000000") ||
          registros[i].getTs() == 0 || registros[i].getSta() != estado)
        {
```

```
            continue;
        }
        if (registros[i].getTs() < min)
        {
            s = i;
            min = registros[i].getTs();
        }
    }
    return s;
}

/**
 * Método que devuelve el índice del registro más reciente de la tabla a
 * partir del instante especificado
 *
 * @param ref Instante de referencia
 * @return Índice del registro más reciente de la tabla
 */
public int MasReciente(long ref)
{
    int indice = -1;
    long min = Long.MAX_VALUE;
    for (int i = 0; i < registros.length; i++)
    {
        if (registros[i].getID().equals("") || registros[i].getTs() <= ref || registros[i].getTs() == 0)
        {
            continue;
        }
        if (registros[i].getTs() < min)
        {
            min = registros[i].getTs();
            indice = i;
        }
    }
    return indice;
}

/**
 * Método que devuelve el índice del registro más reciente de la tabla
 *
 * @return Índice del registro más reciente de la tabla
 */
public int MasReciente()
{
    long dif = Long.MIN_VALUE;
    int s = -1;
    for (int i = 0; i < registros.length; i++)
    {
        if (registros[i].getID().equals("") || registros[i].getTs() == 0)
        {
            continue;
        }
        if (registros[i].getTs() > dif)
        {
            s = i;
```

```
            dif = registros[i].getTs();
        }
    }
    return s;
}

/**
 * Método que devuelve el índice del registro más reciente de la tabla con
 * el estado especificado
 *
 * @param estado Estado considerado
 * @return Índice del registro más reciente de la tabla con el estado
 * especificado
 */
public int MasReciente(int estado)
{
    long max = Long.MIN_VALUE;
    int s = -1;
    for (int i = 0; i < registros.length; i++)
    {
        if (registros[i].getID().equals("") || registros[i].getTs() == 0 || registros[i].getSta() != estado)
        {
            continue;
        }
        if (registros[i].getTs() > max)
        {
            s = i;
            max = registros[i].getTs();
        }
    }
    return s;
}

/**
 * Método que añade un registro a la tabla
 *
 * @param p Nombre del registro
 * @param set Índice de programa
 * @param sta Índice de estado
 * @param dat Campos de datos
 * @param ts Instante
 */
public void NuevaPalanquilla(String p, int set, int sta, String dat, long ts)
{
    for (int i = 0; i < num; i++)
    {
        if (registros[i].getID().equals(p))
        {
            registros[i].setRegistro(set, sta, dat, ts, "");
            return;
        }
    }

    long min = Long.MAX_VALUE;
    int indice = -1;
```

```
    for (int i = 0; i < num; i++)
    {
        //Busco un vacío o el más antiguo
        if (registros[i].getTs() < min)
        {
            min = registros[i].getTs();
            indice = i;
            if (min == 0)
            {
                break;
            }
        }
    }
    if (indice >= 0)
    {
        registros[indice].setRegistro(p, set, sta, dat, ts, "");
    }
}

/**
 * Método que añade un registro a la tabla con el instante actual
 *
 * @param p Nombre del registro
 * @param set Índice de programa
 * @param sta Índice de estado
 * @param dat Campos de datos
 */
public void NuevaPalanquilla(String p, int set, int sta, String dat)
{
    NuevaPalanquilla(p, set, sta, dat, System.currentTimeMillis());
}

/**
 * Método que elimina un registro de la tabla
 *
 * @param p Nombre del registro
 * @return True si se eliminó el registro
 */
public boolean EliminaPalanquilla(String p)
{

    for (int i = 0; i < num; i++)
    {
        if (registros[i].getID().trim().equals(p.trim()))
        {
            registros[i].setRegistro("", 0, 0, "", 0, "");
            return true;
        }
    }

    return false;
}

/**
```

```
 * Método que añade un registro a la tabla con un instante anterior al del
 * registro especificado
 *
 * @param p Nombre del registro
 * @param ref Nombre del registro de referencia
 * @param set Índice de programa
 * @param sta Índice de estado
 * @param dat Campos de datos
 * @return True si se ha añadido el registro
 */
public boolean InsertaPalanquillaAntesDe(String p, String ref, int set, int sta, String dat)
{
   if (ref.equals(""))
   {
      NuevaPalanquillaFinal(p, set, sta, dat);
   }
   else if (p.equals(ref))
   {
      for (int i = 0; i < num; i++)
      {
         if (registros[i].getID().equals(p))
         {
            registros[i].setRegistro(set, sta, dat, System.currentTimeMillis(), "");
            return true;
         }
      }
   }
   else
   {
      for (int i = 0; i < num; i++)
      {
         if (registros[i].getID().equals(ref)) //Referencia
         {
            long nuevaTs = registros[i].getTs() + 1000;
            for (int j = 0; j < num; j++)
            {
               if (registros[j].getTs() == nuevaTs)
               {
                  Rejuvenece(j, nuevaTs);
               }
            }
            NuevaPalanquilla(p, set, sta, dat, nuevaTs);
            return true;
         }
      }
   }
   return false;
}
```

```
        /**
         * Método que añade un registro a la tabla por el final (más antiguo)
         *
         * @param p Nombre del registro
         * @param set Índice de programa
         * @param sta Índice de estado
         * @param dat Campos de datos
         */
        public void NuevaPalanquillaFinal(String p, int set, int sta, String dat)
        {
           long min = Long.MAX_VALUE;
           int indiceMasViejo = -1;
           for (int i = 0; i < num; i++)
           {
              if (registros[i].getTs() < min && registros[i].getTs() > 0)
              {
                 min = registros[i].getTs();
                 indiceMasViejo = i;
              }
           }
           if (indiceMasViejo >= 0)
           {
              NuevaPalanquilla(p, set, sta, dat, registros[indiceMasViejo].getTs() - 1000);
           }
           else
           {
              NuevaPalanquilla(p, set, sta, dat, System.currentTimeMillis());
           }
        }

        /**
         * Método que actualiza los campos de un registro
         *
         * @param indice Índice del registro
         * @param identificacion Identificación del registro
         * @param programa Índice del programa
         * @param estado Índice del estado
         * @param dato Cadena de campos del dato
         * @param instante Instante del registro que permite ordenarlos en la tabla
         * @param var Cadena de campos de la variable
         */
        public void setRegistro(int indice, String identificacion, int programa, int estado, String dato,
long instante, String var)
        {
           registros[indice].setRegistro(identificacion, programa, estado, dato, instante, var);
        }

        /**
         * Método que actualiza el estado de un registro
         *
         * @param p Nombre del registro
         * @param estado Índice del estado
         *
         * @return True si se modificó el estado
         */
```

```
public boolean setEstado(String p, int estado)
{
   for (int i = 0; i < num; i++)
   {
      if (registros[i].getID().equals(p))
      {
         registros[i].setSta(estado);
         return true;
      }
   }
   return false;
}

/**
 * Método que actualiza el programa de un registro
 *
 * @param p Nombre del registro
 * @param programa Índice del programa
 *
 * @return True si se modificó el programa
 */
public boolean setPrograma(String p, int programa)
{
   for (int i = 0; i < num; i++)
   {
      if (registros[i].getID().equals(p))
      {
         registros[i].setSet(programa);
         return true;
      }
   }
   return false;
}

/**
 * Método que obtienene él índice de programa de un registro
 *
 * @param p Nombre del registro
 *
 * @return Índice del programa, 0 si no encontrado
 */
public int getPrograma(String p)
{
   for (int i = 0; i < num; i++)
   {
      if (registros[i].getID().equals(p))
      {
         return registros[i].getSet();
      }
   }
   return 0;
}

/**
 * Método que devuelve la cadena del lote
```

```
 *
 * @param i Índice del registro
 * @return Devuelve la cadena del lote
 */
public String getIDLote(int i)
{
   return registros[i].getIDLote();
}

/**
 * Método que devuelve la cadena del lote
 *
 * @param pal Nombre del registro
 * @return Devuelve la cadena del lote
 */
public String getIDLote(String pal)
{
   for (int i = 0; i < num; i++)
   {
      //Buscando la primera palanquilla en Evacuación
      if (registros[i].getID().equals(pal))
      {
         return getIDLote(i);
      }
   }
   return "";
}

/**
 * Método que actualiza la identificación del registro
 *
 * @param p Antigua identificación del registro
 * @param ID Nueva identificación del registro
 * @return True si se ha modificado la identificación del registro
 */
public boolean setID(String p, String ID)
{
   for (int i = 0; i < num; i++)
   {
      if (registros[i].getID().equals(p))
      {
         registros[i].setID(ID);
         return true;
      }
   }
   return false;
}

/**
 * Método que actualiza la cadena de campos del dato
 *
 * @param p Identificación del registro
 * @param dato Cadena de campos del dato
 * @return True si se modificó el dato
 */
```

```
public boolean setDato(String p, String dato)
{
   for (int i = 0; i < num; i++)
   {
      if (registros[i].getID().equals(p))
      {
         registros[i].setDat(dato);
         return true;
      }
   }
   return false;
}

/**
 * Método que devuelve la cadena de campos del dato
 *
 * @param p Identificación del registro
 * @return Devuelve la cadena de campos del dato
 */
public String getDato(String p)
{
   for (int i = 0; i < num; i++)
   {
      if (registros[i].getID().equals(p))
      {
         return registros[i].getDat();
      }
   }
   return "";
}

/**
 * Método que actualiza la cadena de campos de la variable
 *
 * @param p Identificación del registro
 * @param var Cadena de campos de la variable
 * @return True si se modificó la variable
 */
public boolean setVar(String p, String var)
{
   for (int i = 0; i < num; i++)
   {
      if (registros[i].getID().equals(p))
      {
         registros[i].setVar(var);
         return true;
      }
   }
   return false;
}

/**
 * Método que devuelve la cadena de campos de la variable
 *
 * @param p Identificación del registro
```

```
     * @return Devuelve la cadena de campos de la variable
     */
    public String getVar(String p)
    {
        for (int i = 0; i < num; i++)
        {
            if (registros[i].getID().equals(p))
            {
                return registros[i].getVar();
            }
        }
        return "";
    }

    /**
     * Método que devuelve el índice de estado
     *
     * @param p Identificación del registro
     * @return Devuelve el índice de estado
     */
    public int getEstado(String p)
    {
        for (int i = 0; i < num; i++)
        {
            if (registros[i].getID().equals(p))
            {
                return registros[i].getSta();
            }
        }
        return 0;
    }

     /**
     * Método que devuelve el índice de un registro
     *
     * @param nombre Identificación del registro
     * @return Devuelve el índice del registro
     */
    public int getIndice(String nombre)
    {
        for (int i = 0; i < num; i++)
        {
            if (registros[i].getID().equals(nombre))
            {
                return i;
            }
        }
        return -1;
    }
}
```

1.3 COM.INTERFAZ.LAMINACIÓN

1.3.1 Clase Estado

```
package com.interfaz.laminacion;

/**
 * Clase que implementa la codificación de estados de un registro
 *
 * @author SONIA
 */
public class Estado
{

   /**
    * Cadenas de descripción de estado de registros
    */
   static String[] valoresEstado = new String[]
   {
      "Nada", "Mesa carga", "Horno", "Salida horno", "Desbaste Duo",
      "Salida Duo", "Tren Alambrón", "Bobinadoras", "Bloque", "Transfer TP",
      "Desbaste TP", "Ripador TP", "Tren Perfiles", "Enfriadero",
      "Calidad Alambrón", "Evacuación",
      "Salida Alambrón", "Calidad Perfiles",
      "Prefinalizado Alambrón", "Prefinalizado Perfiles",
      "Finalizado Alambrón", "Finalizado Perfiles",
      "Eliminar", "Madeja directa", "Madeja repercutida", "Recuperada"
   };


   /**
    * Método que devuelve la descripción de estado a partir de su índice
    *
    * @param sta Índice de estado
    * @return Devuelve la cadena de estado
    */
   static public String getDescripcionEstado(int sta)
   {
      String codigo = "";
      if (sta == 1)
      {
         codigo = "Mesa carga";
      }
      else if (sta == 2)
      {
         codigo = "Pesado";
      }
      else if (sta / 100 == 1)
      {
         codigo = "Horno";
      }
      else if (sta / 100 == 2)
      {
         codigo = "Salida horno";
```

```
}
else if (sta / 100 == 3)
{
   codigo = "Desbaste Duo";
   int p = sta % 100;
   if (p != 0)
   {
      codigo = codigo + "=" + String.valueOf(p);
   }
}
else if (sta / 100 == 4)
{
   codigo = "Salida Duo";
}
else if (sta / 100 == 5)
{
   codigo = "Tren Alambrón";
   int p = sta % 500;
   if (p != 0)
   {
      if (p == 1)
      {
         codigo = codigo + "=C01";
      }
      else if (p == 2)
      {
         codigo = codigo + "=C02";
      }
      else if (p == 3)
      {
         codigo = codigo + "=C03";
      }
      else if (p == 4)
      {
         codigo = codigo + "=C04";
      }
      else if (p == 5)
      {
         codigo = codigo + "=C05";
      }
      else if (p == 6)
      {
         codigo = codigo + "=C1";
      }
      else if (p == 7)
      {
         codigo = codigo + "=C2";
      }
      else if (p == 8)
      {
         codigo = codigo + "=C3";
      }
      else if (p == 9)
      {
         codigo = codigo + "=C4";
```

```
        }
        else if (p == 10)
        {
           codigo = codigo + "=C5";
        }
        else if (p == 11)
        {
           codigo = codigo + "=C6";
        }
        else if (p == 12)
        {
           codigo = codigo + "=C7";
        }
        else if (p == 13)
        {
           codigo = codigo + "=C8";
        }
        else if (p == 14)
        {
           codigo = codigo + "=C9";

        }
        else if (p == 15)
        {
           codigo = codigo + "=C10";
        }
        else if (p == 16)
        {
           codigo = codigo + "=C11";
        }
     }
  }
  else if (sta / 100 == 6)
  {
     codigo = "Bobinadoras=" + String.valueOf(sta % 600);

  }
  else if (sta / 100 == 7)
  {
     codigo = "Bloque";
  }
  else if (sta / 100 == 8)
  {
     codigo = "Transfer TP";
  }
  else if (sta / 100 == 9)
  {
     codigo = "Desbaste TP";
  }
  else if (sta / 100 == 10)
  {
     codigo = "Ripador TP";
  }
  else if (sta / 100 == 11)
  {
```

```
            codigo = "Tren Perfiles";
        }
        else if (sta / 100 == 12)
        {
            codigo = "Enfriadero";
        }
        else if (sta / 100 == 13)
        {
            codigo = "Calidad Alambrón";
        }
        else if (sta / 100 == 14)
        {
            codigo = "Evacuación";
        }
        else if (sta / 100 == 15)
        {
            codigo = "Salida Alambrón";
        }
        else if (sta / 100 == 16)
        {
            codigo = "Calidad Perfiles";
        }
        else if (sta / 100 == 17)
        {
            codigo = "Prefinalizado Alambrón";
        }
        else if (sta / 100 == 18)
        {
            codigo = "Prefinalizado Perfiles";
        }
        else if (sta / 100 == 19)
        {
            codigo = "Finalizado Alambrón";
        }
        else if (sta / 100 == 20)
        {
            codigo = "Finalizado Perfiles";
        }
        else if (sta / 100 == 50)
        {
            codigo = "Madeja directa";
        }
        else if (sta / 100 == 51)
        {
            codigo = "Madeja repercutida";
        }
        else if (sta / 100 == 52)
        {
            codigo = "Recuperada";
        }
        return codigo;
    }
```

```
/**
 * Método que devuelve la descripción de estado corta a partir de su índice
 *
 * @param sta Índice de estado
 * @return Devuelve la cadena de estado
 */
static public String getDescripcionEstadoCorto(int sta)
{
   String estado = "";
   if (sta == 1)
   {
      estado = "Carga";
   }
   else if (sta == 2)
   {
      estado = "Pesado";
   }
   else if (sta / 100 == 1)
   {
      estado = "Horno";
   }
   else if (sta / 100 == 2)
   {
      estado = "Fuera";
   }
   else if (sta / 100 == 3)
   {
      estado = "Duo";
      int p = sta % 100;
      if (p != 0)
      {
         estado = estado + "=" + String.valueOf(p);
      }
   }
   else if (sta / 100 == 4)
   {
      estado = "Transfer";
   }
   else if (sta / 100 == 5)
   {
      estado = "Tren";
      int p = sta % 500;
      if (p != 0)
      {
         if (p == 1)
         {
            estado = estado + "=C01";
         }
         else if (p == 2)
         {
            estado = estado + "=C02";
         }
         else if (p == 3)
         {
            estado = estado + "=C03";
```

```
    }
    else if (p == 4)
    {
       estado = estado + "=C04";
    }
    else if (p == 5)
    {
       estado = estado + "=C05";
    }
    else if (p == 6)
    {
       estado = estado + "=C1";
    }
    else if (p == 7)
    {
       estado = estado + "=C2";
    }
    else if (p == 8)
    {
       estado = estado + "=C3";
    }
    else if (p == 9)
    {
       estado = estado + "=C4";
    }
    else if (p == 10)
    {
       estado = estado + "=C5";
    }
    else if (p == 11)
    {
       estado = estado + "=C6";
    }
    else if (p == 12)
    {
       estado = estado + "=C7";
    }
    else if (p == 13)
    {
       estado = estado + "=C8";
    }
    else if (p == 14)
    {
       estado = estado + "=C9";

    }
    else if (p == 15)
    {
       estado = estado + "=C10";
    }
    else if (p == 16)
    {
       estado = estado + "=C11";
    }
}
```

```
}
else if (sta / 100 == 6)
{
   estado = "Bob.=" + String.valueOf(sta % 600);

}
else if (sta / 100 == 7)
{
   estado = "Bloque";
}
else if (sta / 100 == 8)
{
   estado = "Transfer";
}
else if (sta / 100 == 9)
{
   estado = "Desbaste";
}
else if (sta / 100 == 10)
{
   estado = "Ripador";
}
else if (sta / 100 == 11)
{
   estado = "Tren";
}
else if (sta / 100 == 12)
{
   estado = "Enfr";
}
else if (sta / 100 == 13)
{
   estado = "Calidad";
}
else if (sta / 100 == 14)
{
   estado = "Evac";
}
else if (sta / 100 == 15)
{
   estado = "Salida";
}
else if (sta / 100 == 16)
{
   estado = "Calidad";
}
else if (sta / 100 == 17)
{
   estado = "Prefinalizado";
}
else if (sta / 100 == 18)
{
   estado = "Prefinalizado";
}
else if (sta / 100 == 19)
```

```
    {
        estado = "Final";
    }
    else if (sta / 100 == 20)
    {
        estado = "Final";
    }
    else if (sta / 100 == 50)
    {
        estado = "M. dir.";
    }
    else if (sta / 100 == 51)
    {
        estado = "M. rep.";
    }
    else if (sta / 100 == 52)
    {
        estado = "Rec.";
    }
    return estado;
}

/**
 * Método que devuelve el índice de estado a partir de su descripción
 *
 * @param strSta Descripción de estado
 * @return Devuelve el índice de estado
 */
static public int getCodigoEstado(String strSta)
{
    int sta = 0;
    if (strSta.equals("Mesa carga"))
    {
        sta = 1;
    }
    if (strSta.equals("Pesado"))
    {
        sta = 2;
    }
    else if (strSta.equals("Horno"))
    {
        sta = 100;
    }
    else if (strSta.equals("Salida horno"))
    {
        sta = 200;
    }
    else if (strSta.contains("Desbaste Duo"))
    {
        sta = 300;
    }
    else if (strSta.equals("Salida Duo"))
    {
        sta = 400;
    }
```

```
else if (strSta.contains("Tren Alambrón"))
{
   sta = 500;
   String caja = strSta.replace("Tren Alambrón=C", "");
   if (strSta.equals("Tren Alambrón"))
   {
   }
   else if (caja.contains("0"))
   {
      sta += Integer.valueOf(caja);
   }
   else
   {
      sta += Integer.valueOf(caja) + 5;
   }
}
else if (strSta.contains("Bobinadoras"))
{
   if (strSta.equals("Bobinadoras"))
   {
      sta = 600;
   }
   else
   {
      sta = 600 + Integer.valueOf(strSta.replace("Bobinadoras=", ""));
   }
}
else if (strSta.equals("Bloque"))
{
   sta = 700;
}
else if (strSta.equals("Transfer TP"))
{
   sta = 800;
}
else if (strSta.equals("Desbaste TP"))
{
   sta = 900;
}
else if (strSta.equals("Ripador TP"))
{
   sta = 1000;
}
else if (strSta.equals("Tren Perfiles"))
{
   sta = 1100;
}
else if (strSta.equals("Enfriadero"))
{
   sta = 1200;
}
else if (strSta.equals("Calidad Alambrón"))
{
   sta = 1300;
}
```

```
        else if (strSta.equals("Evacuación"))
        {
           sta = 1400;
        }
        else if (strSta.equals("Salida Alambrón"))
        {
           sta = 1500;
        }
        else if (strSta.equals("Calidad Perfiles"))
        {
           sta = 1600;
        }
        else if (strSta.equals("Prefinalizado Alambrón"))
        {
           sta = 1700;
        }
        else if (strSta.equals("Prefinalizado Perfiles"))
        {
           sta = 1800;
        }
        else if (strSta.equals("Finalizado Alambrón"))
        {
           sta = 1900;
        }
        else if (strSta.equals("Finalizado Perfiles"))
        {
           sta = 2000;
        }
        else if (strSta.equals("Madeja directa"))
        {
           sta = 5000;
        }
        else if (strSta.equals("Madeja repercutida"))
        {
           sta = 5100;
        }
        else if (strSta.equals("Recuperada"))
        {
           sta = 5200;
        }
        return sta;
     }

  }
```

1.3.2 Clase Laminación

```
package com.interfaz.laminacion;

import com.Global;
import com.estado.TrazaLaminacion;
import com.opc.ExcepcionOPC;
import com.opc.ItemInfo;
import com.opc.OPCUA;
import com.prosysopc.ua.SecureIdentityException;
import com.prosysopc.ua.ServiceException;
import com.prosysopc.ua.SessionActivationException;
import com.prosysopc.ua.StatusException;
import com.prosysopc.ua.client.AddressSpaceException;
import com.prosysopc.ua.client.MonitoredDataItem;
import com.prosysopc.ua.client.MonitoredDataItemListener;
import com.prosysopc.ua.client.ServerStatusListener;
import com.prosysopc.ua.client.UaClient;
import com.tipoInfo;
import java.io.IOException;
import java.net.URISyntaxException;
import java.text.DecimalFormat;
import org.opcfoundation.ua.builtintypes.DataValue;
import org.opcfoundation.ua.builtintypes.LocalizedText;
import org.opcfoundation.ua.common.ServiceResultException;
import org.opcfoundation.ua.core.ServerState;
import org.opcfoundation.ua.core.ServerStatusDataType;

/**
 * Clase que implementa funcionalidad específica de laminación
 *
 * @author SONIA
 */
public class Laminacion
{

   /**
    * Dirección raíz de tag lotes
    */
   private final String nombreLotes = "Trazabilidad.Lotes";

   /**
    * Tabla de tags y descripciones de variables de proceso del horno
    */
   String[][] nombreVariables = new String[][]
   {
      new String[]
      {
           "Trazabilidad.HOT.horno.calInferior.tmpraBoveda1", "Temperatura Boveda 1
  Calentamiento Inferior"
      },
      new String[]
      {
           "Trazabilidad.HOT.horno.calInferior.tmpraBoveda2", "Temperatura Boveda 2
  Calentamiento Inferior"
```

```
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.tmpraAire", "Temperatura Aire Calentamiento
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.caudalAire_medida", "Caudal Aire Calentamiento
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.caudalGas_medida", "Caudal Gas Calentamiento
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.K1", "Cte. limites cruzados K1 Calentamiento
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.K2", "Cte. limites cruzados K2 Calentamiento
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.K3", "Cte. limites cruzados K3 Calentamiento
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.K4", "Cte. limites cruzados K4 Calentamiento
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.relAireGas", "Relación Aire/Gas Calentamiento
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.temperatura_medida", "Temperatura Medida
Calentamiento Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.temperatura_consigna", "Temperatura Consigna
Calentamiento Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.temperatura_salida", "Salida Regulador
Calentamiento Inferior"
```

```
},
new String[]
{
   "Trazabilidad.HOT.horno.calInferior.Kp", "Cte. reg. Kp Calentamiento Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.calInferior.Ki", "Cte. reg. Ki Calentamiento Inferior"
},
new String[]
{
      "Trazabilidad.HOT.horno.calInferior.tmpraConsignaLocal", "Temperatura Consigna
      Remota Calentamiento Inferior"
},
new String[]
{
       "Trazabilidad.HOT.horno.calInferior.tmpraConsignaRemota", "Temperatura Consigna
      Remota Calentamiento Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.calInferior.spanGas", "Span Gas Calentamiento Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.calInferior.spanAire", "Span Aire Calentamiento Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.calInferior.relAireGasMedido", ""
},
new String[]
{
   "Trazabilidad.HOT.horno.calInferior.B", "Cte. caudal B Calentamiento Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.calInferior.A", "Cte. caudal A Calentamiento Inferior"
},
new String[]
{
       "Trazabilidad.HOT.horno.calInferior.tmpraPared1", "Temperatura Pared 1
      Calentamiento Inferior"
},
new String[]
{
       "Trazabilidad.HOT.horno.calInferior.tmpraPared2", "Temperatura Pared 2
      Calentamiento Inferior"
},
new String[]
{
      "Trazabilidad.HOT.horno.calInferior.ctrlTermopar", "Selección PARED/BOVEDA
      regulación Calentamiento Inferior"
},
new String[]
```

```
    {
        "Trazabilidad.HOT.horno.calInferior.ctrlRemoto", "Conf. Regulación (Local/Remoto)
        Calentamiento Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calInferior.ctrlCaudal", "Modo Regulación (Temp/Caudal)
Calentamiento Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.tmpraBoveda1", "Temperatura Boveda 1
Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.tmpraBoveda2", "Temperatura Boveda 2
Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.tmpraAire", "Temperatura Aire Calentamiento
Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.caudalAire_medida", "Caudal Aire Calentamiento
Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.caudalGas_medida", "Caudal Gas Calentamiento
Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.K1", "Cte. limites cruzados K1 Calentamiento
Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.K2", "Cte. limites cruzados K2 Calentamiento
Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.K3", "Cte. limites cruzados K3 Calentamiento
Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.K4", "Cte. limites cruzados K4 Calentamiento
Superior"
    },
    new String[]
```

```
    {
        "Trazabilidad.HOT.horno.calSuperior.relAireGas", "Relación Aire/Gas Calentamiento
Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.temperatura_medida", "Temperatura Medida
Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.temperatura_consigna", "Temperatura Consigna
Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.temperatura_salida", "Salida Regulador
Calentamiento Superior"
    },
    new String[]
    {
      "Trazabilidad.HOT.horno.calSuperior.Kp", "Cte. reg. Kp Calentamiento Superior"
    },
    new String[]
    {
      "Trazabilidad.HOT.horno.calSuperior.Ki", "Cte. reg. Ki Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.tmpraConsignaLocal", "Temperatura Consigna
Remota Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.tmpraConsignaRemota", "Temperatura Consigna
Remota Calentamiento Superior"
    },
    new String[]
    {
      "Trazabilidad.HOT.horno.calSuperior.spanGas", "Span Gas Calentamiento Superior"
    },
    new String[]
    {
      "Trazabilidad.HOT.horno.calSuperior.spanAire", "Span Aire Calentamiento Superior"
    },
    new String[]
    {
      "Trazabilidad.HOT.horno.calSuperior.relAireGasMedido", ""
    },
    new String[]
    {
      "Trazabilidad.HOT.horno.calSuperior.B", "Cte. caudal B Calentamiento Superior"
    },
    new String[]
    {
```

```
        "Trazabilidad.HOT.horno.calSuperior.A", "Cte. caudal A Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.tmpraPared1", "Temperatura Pared 1
Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.tmpraPared2", "Temperatura Pared 2
Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.ctrlTermopar", "Selección PARED/BOVEDA
regulación Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.ctrlRemoto", "Conf. Regulación (Local/Remoto)
Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.calSuperior.ctrlCaudal", "Modo Regulación (Temp/Caudal)
Calentamiento Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.tmpraBoveda1", "Temperatura Boveda 1 Igualación
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.tmpraBoveda2", "Temperatura Boveda 2 Igualación
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.tmpraAire", "Temperatura Aire Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.caudalAire_medida", "Caudal Aire Igualación
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.caudalGas_medida", "Caudal Gas Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.K1", "Cte. limites cruzados K1 Igualación Inferior"
    },
    new String[]
```

```
    {
       "Trazabilidad.HOT.horno.iguInferior.K2", "Cte. limites cruzados K2 Igualación Inferior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.K3", "Cte. limites cruzados K3 Igualación Inferior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.K4", "Cte. limites cruzados K4 Igualación Inferior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.relAireGas", "Relación Aire/Gas Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.temperatura_medida", "Temperatura Medida
Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.temperatura_consigna", "Temperatura Consigna
Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.temperatura_salida", "Salida Regulador Igualación
Inferior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.Kp", "Cte. reg. Kp Igualación Inferior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.Ki", "Cte. reg. Ki Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.tmpraConsignaLocal", "Temperatura Consigna
Remota Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.tmpraConsignaRemota", "Temperatura Consigna
Remota Igualación Inferior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.spanGas", "Span Gas Igualación Inferior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.spanAire", "Span Aire Igualación Inferior"
```

```
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.relAireGasMedido", ""
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.B", "Cte. caudal B Igualación Inferior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguInferior.A", "Cte. caudal A Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.tmpraPared1", "Temperatura Pared 1 Igualación
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.tmpraPared2", "Temperatura Pared 2 Igualación
Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.ctrlTermopar", "Selección PARED/BOVEDA
regulación Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.ctrlRemoto", "Conf. Regulación (Local/Remoto)
Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguInferior.ctrlCaudal", "Modo Regulación (Temp/Caudal)
Igualación Inferior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguSuperior.tmpraBoveda1", "Temperatura Boveda 1
Igualación Superior"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.iguSuperior.tmpraBoveda2", "Temperatura Boveda 2
Igualación Superior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.iguSuperior.tmpraAire", "Temperatura Aire Igualación Superior"
    },
    new String[]
    {
```

```
            "Trazabilidad.HOT.horno.iguSuperior.caudalAire_medida", "Caudal Aire Igualación
Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.caudalGas_medida", "Caudal Gas Igualación
Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.K1", "Cte. limites cruzados K1 Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.K2", "Cte. limites cruzados K2 Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.K3", "Cte. limites cruzados K3 Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.K4", "Cte. limites cruzados K4 Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.relAireGas", "Relación Aire/Gas Igualación
Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.temperatura_medida", "Temperatura Medida
Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.temperatura_consigna", "Temperatura Consigna
Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.temperatura_salida", "Salida Regulador Igualación
Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.Kp", "Cte. reg. Kp Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.Ki", "Cte. reg. Ki Igualación Superior"
        },
        new String[]
        {
```

```
            "Trazabilidad.HOT.horno.iguSuperior.tmpraConsignaLocal", "Temperatura Consigna
    Remota Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.tmpraConsignaRemota", "Temperatura Consigna
    Remota Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.spanGas", "Span Gas Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.spanAire", "Span Aire Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.relAireGasMedido", ""
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.B", "Cte. caudal B Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.A", "Cte. caudal A Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.tmpraPared1", "Temperatura Pared 1 Igualación
    Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.tmpraPared2", "Temperatura Pared 2 Igualación
    Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.ctrlTermopar", "Selección PARED/BOVEDA
    regulación Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.ctrlRemoto", "Conf. Regulación (Local/Remoto)
    Igualación Superior"
        },
        new String[]
        {
            "Trazabilidad.HOT.horno.iguSuperior.ctrlCaudal", "Modo Regulación (Temp/Caudal)
    Igualación Superior"
        },
        new String[]
        {
```

```
        "Trazabilidad.HOT.horno.general.presionHorno_medida", "Presión Horno Medida"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.general.presionHorno_consigna", "Presión Horno Consigna"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.general.presionHorno_salida", "Presión Horno Salida Regulador"
    },
    new String[]
    {
     "Trazabilidad.HOT.horno.general.presionAireCombustion_medida", "Presión Aire
Combustión Medida"
    },
    new String[]
    {
     "Trazabilidad.HOT.horno.general.presionAireCombustion_consigna", "Presión Aire
      Combustión Consigna"
    },
    new String[]
    {
          "Trazabilidad.HOT.horno.general.presionAireCombustion_salida", "Presión Aire
          Combustión Salida Regulador"
    },
    new String[]
    {
        "Trazabilidad.HOT.horno.general.caudalAguaRefrigerada", "Caudal Agua Refrigerada"
    },
    new String[]
    {
          "Trazabilidad.HOT.horno.general.tmpraAireCaliente_medida", "Temperatura Aire
          Caliente Medida"
    },
    new String[]
    {
          "Trazabilidad.HOT.horno.general.tmpraAireCaliente_consigna", "Temperatura Aire
          Caliente Consigna"
    },
    new String[]
    {
          "Trazabilidad.HOT.horno.general.tmpraAireCaliente_salida", "Temperatura Aire Caliente
          Salida Regulador"
    },
    new String[]
    {
          "Trazabilidad.HOT.horno.general.tmpraAntesRecuperador_medida", "Temperatura Antes
          Recuperador Medida"
    },
    new String[]
    {
          "Trazabilidad.HOT.horno.general.tmpraAntesRecuperador_consigna", "Temperatura
          Antes Recuperador Consigna"
    },
    new String[]
```

```
    {
       "Trazabilidad.HOT.horno.general.tmpraAntesRecuperador_salida", "Temperatura Antes
        Recuperador Salida Regulador"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.general.tmpraSalidaHorno", "Temperatura Salida Horno"
    },
    new String[]
    {
         "Trazabilidad.HOT.horno.general.tmpraHumosDespuesRecuperador", "Temperatura
         Humos Despues Recuperador"
    },
    new String[]
    {
             "Trazabilidad.HOT.horno.general.tmpraRecuperacionSuperior", "Temperatura
         Recuperación Superior"
    },
    new String[]
    {
           "Trazabilidad.HOT.horno.general.tmpraRecuperacionInferior", "Temperatura
         Recuperación Inferior"
    },
};

/**
   * Escuchador de evento de cambio en variables OPC suscritas. Actualiza los campos de la
clase Registro
 *
 */
  private final MonitoredDataItemListener dataChangeListener = new
MonitoredDataItemListener()
{

    @Override
              public void onDataChange(MonitoredDataItem sender, DataValue prevValue,
         DataValue value)
    {
        Global.info.Registra(sender.getNodeId().getValue().toString() + "=" +
        value.getValue().getValue(), tipoInfo.INFO);
       if (sender.getNodeId().getValue().toString().contains(horno.raiz))
       {
          if (horno != null && sender.getNodeId().getValue().toString().contains(horno.raiz +
          ".sta"))
          {
             String nombre = sender.getNodeId().getValue().toString();
             int indice = Integer.valueOf(nombre.replace(horno.raiz + ".sta", ""));
             if (horno.registros[indice - 1] != null)
             {
                horno.registros[indice - 1].sta = value.getValue().intValue();
                horno.registros[indice - 1].ActualizaNombreEstado();
             }
             return;
          }
          if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".set"))
```

```
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".set", ""));
   if (horno.registros[indice - 1] != null)
   {
      horno.registros[indice - 1].set = value.getValue().intValue();
   }
   return;
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".ID"))
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".ID", ""));
   if (horno.registros[indice - 1] != null)
   {
      horno.registros[indice - 1].ID = value.getValue().getValue().toString();
      horno.registros[indice - 1].ActualizaLoteColada();
   }
   return;
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".dat"))
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".dat", ""));
   if (horno.registros[indice - 1] != null)
   {
      horno.registros[indice - 1].dat = value.getValue().getValue().toString();
      horno.registros[indice - 1].ActualizaLoteDato();
      if (horno.nombre.equals("paquetes"))
      {
         horno.registros[indice - 1].ActualizaNombreProgramaPaquete();
      }
      else
      {
         horno.registros[indice - 1].ActualizaNombreProgramaPalanquilla();
      }
   }
   return;
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".var"))
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".var", ""));
   if (horno.registros[indice - 1] != null)
   {
      horno.registros[indice - 1].var = value.getValue().getValue().toString();
      horno.registros[indice - 1].ActualizaLoteDato();
   }
   return;
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".ts"))
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".ts", ""));
   if (horno.registros[indice - 1] != null)
```

```
                {
                    horno.registros[indice - 1].ts = value.getValue().longValue() * 1000;
                }
                return;
            }
        }
        if (sender.getNodeId().getValue().toString().contains(nombreLotes + ".pos"))
        {
            String nombre = sender.getNodeId().getValue().toString();
            int indice = Integer.valueOf(nombre.replace(nombreLotes + ".pos", ""));
            if (lotes[indice - 1] != null)
            {
                lotes[indice - 1] = value.getValue().getValue().toString();
            }
        }
        else if (sender.getNodeId().getValue().toString().equals(nombreLotes + ".indice"))
        {
            indiceLote = value.getValue().intValue();
        }
    }
};

/**
 * Escuchador de evento de cambio en el estado del servidor OPC
 *
 */
private final ServerStatusListener serverStatusListener = new ServerStatusListener()
{

    @Override
public void onShutdown(UaClient uaClient, long secondsTillShutdown, LocalizedText
shutdownReason)
    {

    }

    @Override
    public void onStateChange(UaClient uaClient, ServerState oldState, ServerState newState)
    {

    }

    @Override
    public void onStatusChange(UaClient uaClient, ServerStatusDataType status)
    {

    }
};

/**
 * Tabla OPC del horno
 */
public TrazaLaminacion horno;
/**
 * Conexión OPC UA
```

```
 */
public OPCUA opc;

/**
 * Descripción de lotes
 */
public String[] lotes;
/**
 * Descripción del lote actual
 */
String loteActual;
/**
 * Índice del lote actual
 */
int indiceLote;
/**
 * nümero de lotes
 */
final int nLotes = 100;

/**
 * Método que devuelve el valor de una variable de proceso incluido en
 * un array de valores leídos con el método LeerGrupoOPC de la clase OPCUA
 *
 * @param valores Array de valores opc leídos
 * @param nombreVariable Nombre de la variable de proceso, incluida en nombreVariables
 * @return Valor de la variable de proceso
 */
public String getValor(DataValue[] valores, String nombreVariable)
{
   DecimalFormat df = new DecimalFormat("0.0");
   for (int i = 0; i < nombreVariables.length; i++)
   {
      if (nombreVariables[i][1].equals(nombreVariable))
      {
                  if (nombreVariables[i][0].contains("ctrlTermopar") ||
         nombreVariables[i][0].contains("ctrlRemoto") ||
         nombreVariables[i][0].contains("ctrlCaudal"))
         {
            return valores[i].getValue().toString();
         }
         else
         {
            return df.format(valores[i].getValue().doubleValue()).replace(",",".");
         }
      }

   }
   return null;
}

/**
 * Método que devuelve el índice de un lote y si no existe lo crea
 *
 * @param lote Descripción del lote
```

```
 * @param crear Si es necesario añadir el lote a la tabla de lotes si la
 * descripción no está presente
 * @return Índice del lote
 */
public int getLote(String lote, boolean crear)
{
   if (crear)
   {
      Global.info.Info("Buscando lote: " + lote);
   }
   if (lote.equals(""))
   {
      if (crear)
      {
         Global.info.Aviso("Lote incorrecto");
      }
      return -1;
   }

   for (int i = 0; i < nLotes; i++)
   {
      if (lotes[i].equals(lote))
      {
         if (crear)
         {
            Global.info.Info("Lote= " + String.valueOf(i + 1));
         }
         return (i + 1);
      }
   }
   if (!crear)
   {
      return -1;
   }
   try
   {
      if (indiceLote == nLotes)
      {
         indiceLote = 0;
      }
      indiceLote++;
      Global.info.Info("Lote no encontrado. Escribe info lote en posición " +
       String.valueOf(indiceLote));
      opc.Escribe(nombreLotes + ".pos" + String.valueOf(indiceLote), lote);
      opc.Escribe(nombreLotes + ".indice", String.valueOf(indiceLote));
      return indiceLote;
   }
   catch (AddressSpaceException | ServiceException | StatusException ex)
   {
      Global.info.Registra(ex);
   }
   return -1;
}

/**
```

```
     * Constructor
     * Inicializa la tabla de lotes, crea la conexión OPC UA,
     * crea la tabla OPC del horno, añade los tags de las variables de proceso
     * y de la tabla OPC del horno y conecta
     * @throws com.prosysopc.ua.ServiceException
     * @throws com.prosysopc.ua.StatusException
     * @throws org.opcfoundation.ua.common.ServiceResultException
     * @throws com.opc.ExcepcionOPC
     * @throws com.prosysopc.ua.SessionActivationException
     * @throws java.net.URISyntaxException
     * @throws com.prosysopc.ua.SecureIdentityException
     * @throws java.io.IOException
     */
    public Laminacion() throws ServiceException, StatusException, ServiceResultException,
    ExcepcionOPC, SessionActivationException, URISyntaxException, SecureIdentityException,
    IOException
     {
        lotes = new String[nLotes];
        for (int i = 0;
                i < nLotes;
                i++)
        {
           lotes[i] = "";
        }

        opc = new OPCUA("MiOPCUA", "opc.tcp://172.19.1.32:49320", dataChangeListener,
        serverStatusListener);
        horno = new TrazaLaminacion(99, "Trazabilidad.Horno", "horno", opc, true);
        opc.AnnadeItem(horno.getItemInfoArray());

        for (int i = 0; i < nombreVariables.length; i++)
        {
           opc.AnnadeItem(new ItemInfo(nombreVariables[i][0], false, 0, nombreVariables[i][1]));
        }
        for (int i = 0; i < nLotes; i++)
        {
           opc.AnnadeItem(new ItemInfo(nombreLotes + ".pos" + String.valueOf(i + 1), true, -1));
        }
        opc.AnnadeItem(new ItemInfo(nombreLotes + ".indice", true, -1));

        opc.Conecta();
     }

}
```

1.4 COM.OPC

1.4.1 Clase ExcepcionOPC

```
package com.interfaz.laminacion;

import com.Global;
import com.estado.TrazaLaminacion;
import com.opc.ExcepcionOPC;
import com.opc.ItemInfo;
import com.opc.OPCUA;
import com.prosysopc.ua.SecureIdentityException;
import com.prosysopc.ua.ServiceException;
import com.prosysopc.ua.SessionActivationException;
import com.prosysopc.ua.StatusException;
import com.prosysopc.ua.client.AddressSpaceException;
import com.prosysopc.ua.client.MonitoredDataItem;
import com.prosysopc.ua.client.MonitoredDataItemListener;
import com.prosysopc.ua.client.ServerStatusListener;
import com.prosysopc.ua.client.UaClient;
import com.tipoInfo;
import java.io.IOException;
import java.net.URISyntaxException;
import java.text.DecimalFormat;
import org.opcfoundation.ua.builtintypes.DataValue;
import org.opcfoundation.ua.builtintypes.LocalizedText;
import org.opcfoundation.ua.common.ServiceResultException;
import org.opcfoundation.ua.core.ServerState;
import org.opcfoundation.ua.core.ServerStatusDataType;

/**
 * Clase que implementa funcionalidad específica de laminación
 *
 * @author SONIA
 */
public class Laminacion
{

    /**
     * Dirección raíz de tag lotes
     */
    private final String nombreLotes = "Trazabilidad.Lotes";

    /**
     * Tabla de tags y descripciones de variables de proceso del horno
     */
    String[][] nombreVariables = new String[][]
    {
        new String[]
        {
            "Trazabilidad.HOT.horno.calInferior.tmpraBoveda1", "Temperatura Boveda 1
  Calentamiento Inferior"
        },
        new String[]
```

```
{
        "Trazabilidad.HOT.horno.calInferior.tmpraBoveda2", "Temperatura Boveda 2
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.tmpraAire", "Temperatura Aire
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.caudalAire_medida", "Caudal Aire
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.caudalGas_medida", "Caudal Gas
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.K1", "Cte. limites cruzados K1
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.K2", "Cte. limites cruzados K2
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.K3", "Cte. limites cruzados K3
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.K4", "Cte. limites cruzados K4
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.relAireGas", "Relación Aire/Gas
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.temperatura_medida", "Temperatura Medida
    Calentamiento Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calInferior.temperatura_consigna", "Temperatura
    Consigna Calentamiento Inferior"
},
new String[]
```

```
{
            "Trazabilidad.HOT.horno.calInferior.temperatura_salida", "Salida Regulador
    Calentamiento Inferior"
},
new String[]
{
  "Trazabilidad.HOT.horno.calInferior.Kp", "Cte. reg. Kp Calentamiento Inferior"
},
new String[]
{
  "Trazabilidad.HOT.horno.calInferior.Ki", "Cte. reg. Ki Calentamiento Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.calInferior.tmpraConsignaLocal", "Temperatura
    Consigna Remota Calentamiento Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.calInferior.tmpraConsignaRemota", "Temperatura
    Consigna Remota Calentamiento Inferior"
},
new String[]
{
  "Trazabilidad.HOT.horno.calInferior.spanGas", "Span Gas Calentamiento Inferior"
},
new String[]
{
  "Trazabilidad.HOT.horno.calInferior.spanAire", "Span Aire Calentamiento Inferior"
},
new String[]
{
  "Trazabilidad.HOT.horno.calInferior.relAireGasMedido", ""
},
new String[]
{
  "Trazabilidad.HOT.horno.calInferior.B", "Cte. caudal B Calentamiento Inferior"
},
new String[]
{
  "Trazabilidad.HOT.horno.calInferior.A", "Cte. caudal A Calentamiento Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.calInferior.tmpraPared1", "Temperatura Pared 1
    Calentamiento Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.calInferior.tmpraPared2", "Temperatura Pared 2
    Calentamiento Inferior"
},
new String[]
{
```

```
                "Trazabilidad.HOT.horno.calInferior.ctrlTermopar", "Selección
        PARED/BOVEDA regulación Calentamiento Inferior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calInferior.ctrlRemoto", "Conf. Regulación
        (Local/Remoto) Calentamiento Inferior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calInferior.ctrlCaudal", "Modo Regulación
        (Temp/Caudal) Calentamiento Inferior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.tmpraBoveda1", "Temperatura Boveda 1
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.tmpraBoveda2", "Temperatura Boveda 2
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.tmpraAire", "Temperatura Aire
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.caudalAire_medida", "Caudal Aire
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.caudalGas_medida", "Caudal Gas
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.K1", "Cte. limites cruzados K1
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.K2", "Cte. limites cruzados K2
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.K3", "Cte. limites cruzados K3
        Calentamiento Superior"
    },
    new String[]
    {
```

```
                "Trazabilidad.HOT.horno.calSuperior.K4", "Cte. limites cruzados K4
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.relAireGas", "Relación Aire/Gas
        Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.temperatura_medida", "Temperatura
        Medida Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.temperatura_consigna", "Temperatura
        Consigna Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.temperatura_salida", "Salida Regulador
        Calentamiento Superior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.calSuperior.Kp", "Cte. reg. Kp Calentamiento Superior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.calSuperior.Ki", "Cte. reg. Ki Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.tmpraConsignaLocal", "Temperatura
        Consigna Remota Calentamiento Superior"
    },
    new String[]
    {
                "Trazabilidad.HOT.horno.calSuperior.tmpraConsignaRemota", "Temperatura
        Consigna Remota Calentamiento Superior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.calSuperior.spanGas", "Span Gas Calentamiento Superior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.calSuperior.spanAire", "Span Aire Calentamiento Superior"
    },
    new String[]
    {
       "Trazabilidad.HOT.horno.calSuperior.relAireGasMedido", ""
    },
    new String[]
    {
```

```
    "Trazabilidad.HOT.horno.calSuperior.B", "Cte. caudal B Calentamiento Superior"
},
new String[]
{
    "Trazabilidad.HOT.horno.calSuperior.A", "Cte. caudal A Calentamiento Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calSuperior.tmpraPared1", "Temperatura Pared 1
    Calentamiento Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calSuperior.tmpraPared2", "Temperatura Pared 2
    Calentamiento Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calSuperior.ctrlTermopar", "Selección
    PARED/BOVEDA regulación Calentamiento Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calSuperior.ctrlRemoto", "Conf. Regulación
    (Local/Remoto) Calentamiento Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.calSuperior.ctrlCaudal", "Modo Regulación
    (Temp/Caudal) Calentamiento Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguInferior.tmpraBoveda1", "Temperatura Boveda 1
    Igualación Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguInferior.tmpraBoveda2", "Temperatura Boveda 2
    Igualación Inferior"
},
new String[]
{
    "Trazabilidad.HOT.horno.iguInferior.tmpraAire", "Temperatura Aire Igualación Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguInferior.caudalAire_medida", "Caudal Aire
    Igualación Inferior"
},
new String[]
{
    "Trazabilidad.HOT.horno.iguInferior.caudalGas_medida", "Caudal Gas Igualación Inferior"
},
new String[]
```

```
{
   "Trazabilidad.HOT.horno.iguInferior.K1", "Cte. limites cruzados K1 Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.K2", "Cte. limites cruzados K2 Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.K3", "Cte. limites cruzados K3 Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.K4", "Cte. limites cruzados K4 Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.relAireGas", "Relación Aire/Gas Igualación Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.iguInferior.temperatura_medida", "Temperatura
      Medida Igualación Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.iguInferior.temperatura_consigna", "Temperatura
      Consigna Igualación Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.iguInferior.temperatura_salida", "Salida Regulador
      Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.Kp", "Cte. reg. Kp Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.Ki", "Cte. reg. Ki Igualación Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.iguInferior.tmpraConsignaLocal", "Temperatura
      Consigna Remota Igualación Inferior"
},
new String[]
{
            "Trazabilidad.HOT.horno.iguInferior.tmpraConsignaRemota", "Temperatura
      Consigna Remota Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.spanGas", "Span Gas Igualación Inferior"
```

```
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.spanAire", "Span Aire Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.relAireGasMedido", ""
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.B", "Cte. caudal B Igualación Inferior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguInferior.A", "Cte. caudal A Igualación Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguInferior.tmpraPared1", "Temperatura Pared 1
    Igualación Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguInferior.tmpraPared2", "Temperatura Pared 2
    Igualación Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguInferior.ctrlTermopar", "Selección
    PARED/BOVEDA regulación Igualación Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguInferior.ctrlRemoto", "Conf. Regulación
    (Local/Remoto) Igualación Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguInferior.ctrlCaudal", "Modo Regulación
    (Temp/Caudal) Igualación Inferior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguSuperior.tmpraBoveda1", "Temperatura Boveda 1
    Igualación Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguSuperior.tmpraBoveda2", "Temperatura Boveda 2
    Igualación Superior"
},
new String[]
{
   "Trazabilidad.HOT.horno.iguSuperior.tmpraAire", "Temperatura Aire Igualación Superior"
```

```
},
new String[]
{
        "Trazabilidad.HOT.horno.iguSuperior.caudalAire_medida", "Caudal Aire
    Igualación Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguSuperior.caudalGas_medida", "Caudal Gas
    Igualación Superior"
},
new String[]
{
  "Trazabilidad.HOT.horno.iguSuperior.K1", "Cte. limites cruzados K1 Igualación Superior"
},
new String[]
{
  "Trazabilidad.HOT.horno.iguSuperior.K2", "Cte. limites cruzados K2 Igualación Superior"
},
new String[]
{
  "Trazabilidad.HOT.horno.iguSuperior.K3", "Cte. limites cruzados K3 Igualación Superior"
},
new String[]
{
  "Trazabilidad.HOT.horno.iguSuperior.K4", "Cte. limites cruzados K4 Igualación Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguSuperior.relAireGas", "Relación Aire/Gas
    Igualación Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguSuperior.temperatura_medida", "Temperatura
    Medida Igualación Superior"
},
new String[]
{
        "Trazabilidad.HOT.horno.iguSuperior.temperatura_consigna", "Temperatura
    Consigna Igualación Superior"
  },
new String[]
{
        "Trazabilidad.HOT.horno.iguSuperior.temperatura_salida", "Salida Regulador
    Igualación Superior"
},
new String[]
{
  "Trazabilidad.HOT.horno.iguSuperior.Kp", "Cte. reg. Kp Igualación Superior"
},
new String[]
{
  "Trazabilidad.HOT.horno.iguSuperior.Ki", "Cte. reg. Ki Igualación Superior"
},
```

```
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.tmpraConsignaLocal", "Temperatura Consigna
Remota Igualación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.tmpraConsignaRemota", "Temperatura Consigna
Remota Igualación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.spanGas", "Span Gas Igualación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.spanAire", "Span Aire Igualación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.relAireGasMedido", ""
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.B", "Cte. caudal B Igualación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.A", "Cte. caudal A Igualación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.tmpraPared1", "Temperatura Pared 1 Igualación
Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.tmpraPared2", "Temperatura Pared 2 Igualación
Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.ctrlTermopar", "Selección PARED/BOVEDA
regulación Igualación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.ctrlRemoto", "Conf. Regulación (Local/Remoto)
Igualación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.iguSuperior.ctrlCaudal", "Modo Regulación (Temp/Caudal)
Igualación Superior"
            },
```

```
            new String[]
            {
               "Trazabilidad.HOT.horno.general.presionHorno_medida", "Presión Horno Medida"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.presionHorno_consigna", "Presión Horno Consigna"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.presionHorno_salida", "Presión Horno Salida Regulador"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.presionAireCombustion_medida", "Presión Aire
Combustión Medida"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.presionAireCombustion_consigna", "Presión Aire
Combustión Consigna"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.presionAireCombustion_salida", "Presión Aire
Combustión Salida Regulador"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.caudalAguaRefrigerada", "Caudal Agua Refrigerada"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraAireCaliente_medida", "Temperatura Aire Caliente
Medida"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraAireCaliente_consigna", "Temperatura Aire
Caliente Consigna"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraAireCaliente_salida", "Temperatura Aire Caliente
Salida Regulador"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraAntesRecuperador_medida", "Temperatura Antes
Recuperador Medida"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraAntesRecuperador_consigna", "Temperatura Antes
Recuperador Consigna"
```

```
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraAntesRecuperador_salida", "Temperatura Antes
Recuperador Salida Regulador"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraSalidaHorno", "Temperatura Salida Horno"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraHumosDespuesRecuperador", "Temperatura
Humos Despues Recuperador"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraRecuperacionSuperior", "Temperatura
Recuperación Superior"
            },
            new String[]
            {
               "Trazabilidad.HOT.horno.general.tmpraRecuperacionInferior", "Temperatura Recuperación
Inferior"
            },
         };

         /**
          * Escuchador de evento de cambio en variables OPC suscritas. Actualiza los campos de la clase
Registro
          *
          */
         private final MonitoredDataItemListener dataChangeListener = new
MonitoredDataItemListener()
         {

            @Override
            public void onDataChange(MonitoredDataItem sender, DataValue prevValue, DataValue
value)
            {
               Global.info.Registra(sender.getNodeId().getValue().toString() + "=" +
value.getValue().getValue(), tipoInfo.INFO);
               if (sender.getNodeId().getValue().toString().contains(horno.raiz))
               {
                  if (horno != null && sender.getNodeId().getValue().toString().contains(horno.raiz +
".sta"))
                  {
                     String nombre = sender.getNodeId().getValue().toString();
                     int indice = Integer.valueOf(nombre.replace(horno.raiz + ".sta", ""));
                     if (horno.registros[indice - 1] != null)
                     {
                        horno.registros[indice - 1].sta = value.getValue().intValue();
                        horno.registros[indice - 1].ActualizaNombreEstado();
                     }
                     return;
```

```
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".set"))
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".set", ""));
   if (horno.registros[indice - 1] != null)
   {
      horno.registros[indice - 1].set = value.getValue().intValue();
   }
   return;
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".ID"))
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".ID", ""));
   if (horno.registros[indice - 1] != null)
   {
      horno.registros[indice - 1].ID = value.getValue().getValue().toString();
      horno.registros[indice - 1].ActualizaLoteColada();
   }
   return;
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".dat"))
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".dat", ""));
   if (horno.registros[indice - 1] != null)
   {
      horno.registros[indice - 1].dat = value.getValue().getValue().toString();
      horno.registros[indice - 1].ActualizaLoteDato();
      if (horno.nombre.equals("paquetes"))
      {
         horno.registros[indice - 1].ActualizaNombreProgramaPaquete();
      }
      else
      {
         horno.registros[indice - 1].ActualizaNombreProgramaPalanquilla();
      }
   }
   return;
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".var"))
{
   String nombre = sender.getNodeId().getValue().toString();
   int indice = Integer.valueOf(nombre.replace(horno.raiz + ".var", ""));
   if (horno.registros[indice - 1] != null)
   {
      horno.registros[indice - 1].var = value.getValue().getValue().toString();
      horno.registros[indice - 1].ActualizaLoteDato();
   }
   return;
}
if (sender.getNodeId().getValue().toString().contains(horno.raiz + ".ts"))
{
   String nombre = sender.getNodeId().getValue().toString();
```

```
                    int indice = Integer.valueOf(nombre.replace(horno.raiz + ".ts", ""));
                    if (horno.registros[indice - 1] != null)
                    {
                       horno.registros[indice - 1].ts = value.getValue().longValue() * 1000;
                    }
                    return;
                 }
              }
              if (sender.getNodeId().getValue().toString().contains(nombreLotes + ".pos"))
              {
                 String nombre = sender.getNodeId().getValue().toString();
                 int indice = Integer.valueOf(nombre.replace(nombreLotes + ".pos", ""));
                 if (lotes[indice - 1] != null)
                 {
                    lotes[indice - 1] = value.getValue().getValue().toString();
                 }
              }
              else if (sender.getNodeId().getValue().toString().equals(nombreLotes + ".indice"))
              {
                 indiceLote = value.getValue().intValue();
              }
           }
        };

        /**
         * Escuchador de evento de cambio en el estado del servidor OPC
         *
         */
        private final ServerStatusListener serverStatusListener = new ServerStatusListener()
        {

           @Override
           public void onShutdown(UaClient uaClient, long secondsTillShutdown, LocalizedText
shutdownReason)
           {

           }

           @Override
           public void onStateChange(UaClient uaClient, ServerState oldState, ServerState newState)
           {

           }

           @Override
           public void onStatusChange(UaClient uaClient, ServerStatusDataType status)
           {

           }
        };

        /**
         * Tabla OPC del horno
         */
        public TrazaLaminacion horno;
```

```
        /**
         * Conexión OPC UA
         */
        public OPCUA opc;

        /**
         * Descripción de lotes
         */
        public String[] lotes;
        /**
         * Descripción del lote actual
         */
        String loteActual;
        /**
         * Índice del lote actual
         */
        int indiceLote;
        /**
         * nümero de lotes
         */
        final int nLotes = 100;

        /**
         * Método que devuelve el valor de una variable de proceso incluido en
         * un array de valores leídos con el método LeerGrupoOPC de la clase OPCUA
         *
         * @param valores Array de valores opc leídos
         * @param nombreVariable Nombre de la variable de proceso, incluida en nombreVariables
         * @return Valor de la variable de proceso
         */
        public String getValor(DataValue[] valores, String nombreVariable)
        {
           DecimalFormat df = new DecimalFormat("0.0");
           for (int i = 0; i < nombreVariables.length; i++)
           {
              if (nombreVariables[i][1].equals(nombreVariable))
              {
                 if (nombreVariables[i][0].contains("ctrlTermopar") ||
nombreVariables[i][0].contains("ctrlRemoto") || nombreVariables[i][0].contains("ctrlCaudal"))
                 {
                    return valores[i].getValue().toString();
                 }
                 else
                 {
                    return df.format(valores[i].getValue().doubleValue()).replace(",",".");
                 }
              }

           }
           return null;
        }

        /**
         * Método que devuelve el índice de un lote y si no existe lo crea
         *
```

```
     * @param lote Descripción del lote
     * @param crear Si es necesario añadir el lote a la tabla de lotes si la
     * descripción no está presente
     * @return Índice del lote
     */
    public int getLote(String lote, boolean crear)
    {
        if (crear)
        {
            Global.info.Info("Buscando lote: " + lote);
        }
        if (lote.equals(""))
        {
            if (crear)
            {
                Global.info.Aviso("Lote incorrecto");
            }
            return -1;
        }

        for (int i = 0; i < nLotes; i++)
        {
            if (lotes[i].equals(lote))
            {
                if (crear)
                {
                    Global.info.Info("Lote= " + String.valueOf(i + 1));
                }
                return (i + 1);
            }
        }
        if (!crear)
        {
            return -1;
        }
        try
        {
            if (indiceLote == nLotes)
            {
                indiceLote = 0;
            }
            indiceLote++;
            Global.info.Info("Lote no encontrado. Escribe info lote en posición " +
String.valueOf(indiceLote));
            opc.Escribe(nombreLotes + ".pos" + String.valueOf(indiceLote), lote);
            opc.Escribe(nombreLotes + ".indice", String.valueOf(indiceLote));
            return indiceLote;
        }
        catch (AddressSpaceException | ServiceException | StatusException ex)
        {
            Global.info.Registra(ex);
        }
        return -1;
    }
```

```
        /**
         * Constructor
         * Inicializa la tabla de lotes, crea la conexión OPC UA,
         * crea la tabla OPC del horno, añade los tags de las variables de proceso
         * y de la tabla OPC del horno y conecta
         * @throws com.prosysopc.ua.ServiceException
         * @throws com.prosysopc.ua.StatusException
         * @throws org.opcfoundation.ua.common.ServiceResultException
         * @throws com.opc.ExcepcionOPC
         * @throws com.prosysopc.ua.SessionActivationException
         * @throws java.net.URISyntaxException
         * @throws com.prosysopc.ua.SecureIdentityException
         * @throws java.io.IOException
         */
        public Laminacion() throws ServiceException, StatusException, ServiceResultException,
ExcepcionOPC, SessionActivationException, URISyntaxException, SecureIdentityException,
IOException
        {
           lotes = new String[nLotes];
           for (int i = 0;
                 i < nLotes;
                 i++)
           {
              lotes[i] = "";
           }

           opc = new OPCUA("MiOPCUA", "opc.tcp://172.19.1.32:49320", dataChangeListener,
serverStatusListener);
           horno = new TrazaLaminacion(99, "Trazabilidad.Horno", "horno", opc, true);
           opc.AnnadeItem(horno.getItemInfoArray());

           for (int i = 0; i < nombreVariables.length; i++)
           {
              opc.AnnadeItem(new ItemInfo(nombreVariables[i][0], false, 0, nombreVariables[i][1]));
           }
           for (int i = 0; i < nLotes; i++)
           {
              opc.AnnadeItem(new ItemInfo(nombreLotes + ".pos" + String.valueOf(i + 1), true, -1));
           }
           opc.AnnadeItem(new ItemInfo(nombreLotes + ".indice", true, -1));

           opc.Conecta();
        }



     }
```

1.4.2 Clase GlobalOPC

```
package com.opc;

/**
 *
 * Clase que implementa métodos globales de OPC UA
 *
 * @author SONIA
 */
public class GlobalOPC
{
    /**
     * Método para proteger de escrituras en OPC indeseadas
     *
     * @param tag Nombre del tag OPC
     * @return True si se permite escribir el tag
     */
    public static boolean permiteEscribir(String tag)
    {
        return false;
    }
}
```

1.4.3 Clase ItemInfo

```
package com.opc;

import com.Global;
import com.prosysopc.ua.ServiceException;
import com.prosysopc.ua.StatusException;
import com.prosysopc.ua.client.AddressSpaceException;
import com.prosysopc.ua.nodes.UaDataType;
import com.prosysopc.ua.nodes.UaNode;
import com.prosysopc.ua.nodes.UaVariable;
import com.tipoInfo;
import org.opcfoundation.ua.builtintypes.DataValue;
import org.opcfoundation.ua.builtintypes.NodeId;
import org.opcfoundation.ua.builtintypes.Variant;
import org.opcfoundation.ua.core.Attributes;
import org.opcfoundation.ua.builtintypes.StatusCode;

/**
 * Clase que implementa un item OPC UA para su uso con el OPC UA Client SDK de Prosys
 *
 * @author SONIA
 */

public class ItemInfo
{
    /**
     * Nombre del tag
     */
```

```
private String nombre;
/**
 * Descripción del tag
 */
private String descripcion;
/**
 * Si True suscripción a cambios
 */
public boolean notificacionCambio;
/**
 * Índice de grupo para lectura agrupada
 */
public int lecturaAgrupada;
/**
 * ID de Nodo objeto Prosys
 */
private NodeId IDNodo;
/**
 * Conexión OPC UA
 */
private OPCUA opc;
/**
 * Valor en objeto Prosys
 */
private DataValue valor;
/**
 * Índice del tag
 */
private int indice;
/**
 * Banda muerta. Si se excede genera evento de cambio
 */
private double deadband;

/**
 * Constructor de tag OPC UA
 *
 * @param n Nombre del tag
 * @param db Banda muerta. Si 0 no se suscribe a cambios
 * @param la índice de grupo para lectura agrupada
 */
public ItemInfo(String n, int db, int la)
{
   this();
   nombre = n;
   if (db < 0)
   {
      notificacionCambio = false;
      deadband = 0;
   }
   else
   {
      notificacionCambio = true;
      deadband = db;
   }
```

```
    lecturaAgrupada = la;
    descripcion = "";
}

/**
 * Constructor de tag OPC UA
 *
 * @param n Nombre del tag
 * @param odc Si True se suscribe a cambios
 * @param la índice de grupo para lectura agrupada
 */
public ItemInfo(String n, boolean odc, int la)
{
    this();
    nombre = n;
    notificacionCambio = odc;
    lecturaAgrupada = la;
    deadband = 0;
    descripcion = "";
}

/**
 * Constructor de tag OPC UA
 *
 * @param n Nombre del tag
 * @param odc Si True se suscribe a cambios
 * @param la índice de grupo para lectura agrupada
 * @param descr Descripción del tag
 */
public ItemInfo(String n, boolean odc, int la, String descr)
{
    this(n, odc, la);
    descripcion = descr;

}

/**
 * Constructor de tag OPC UA
 *
 */
public ItemInfo()
{
    nombre = "";
    notificacionCambio = false;
    lecturaAgrupada = -1;
    deadband = 0;
    descripcion = "";
}

/**
 * @return nombre del tag
 */
public String getNombre()
{
```

```
    return nombre;
}

/**
 * @param nombre Fija nombre al tag
 */
public void setNombre(String nombre)
{
    this.nombre = nombre;
}

/**
 * @return si está suscrito a cambios
 */
public boolean isNotificacionCambio()
{
    return notificacionCambio;
}

/**
 * @param notificacionCambio Fija suscripción a cambios
 */
public void setNotificacionCambio(boolean notificacionCambio)
{
    this.notificacionCambio = notificacionCambio;
}

/**
 * @return el objeto Prosys del nodo del tag
 */
public NodeId getIDNodo()
{
    return IDNodo;
}

/**
 * @param IDNodo Fija el objeto Prosys del nodo del tag
 */
public void setIDNodo(NodeId IDNodo)
{
    this.IDNodo = IDNodo;
}

/**
 * @return La conexión OPC UA
 */
public OPCUA getOpc()
{
    return opc;
}

/**
 * @param opc Fija la conexión OPC UA
 */
public void setOpc(OPCUA opc)
```

```
{
   this.opc = opc;
}

/**
 * @return el valor del tag
 */
public DataValue getValor()
{
   return valor;
}

/**
 * @param valor Fija el valor del tag
 */
public void setValor(DataValue valor)
{
   this.valor = valor;
}

/**
 * @return la descripción del tag
 */
public String getDescripcion()
{
   return descripcion;
}

/**
 * @param descripcion Fija la descripción del tag
 */
public void setDescripcion(String descripcion)
{
   this.descripcion = descripcion;
}

/**
 * @return el índice del tag
 */
public int getIndice()
{
   return indice;
}

/**
 * @param indice Fija el indice del tag
 */
public void setIndice(int indice)
{
   this.indice = indice;
}

 /**
 * Método que escribe un valor en el tag OPC
 *
```

```
         * @param valor Valor a escribir
         * @throws com.prosysopc.ua.ServiceException
         * @throws com.prosysopc.ua.client.AddressSpaceException
         * @throws com.prosysopc.ua.StatusException
         */
        public void Escribe(String valor) throws ServiceException, AddressSpaceException,
StatusException
        {

            UaNode node = opc.client.getAddressSpace().getNode(IDNodo);
            Global.info.Info("Escribiendo OPC  en " + IDNodo.getValue().toString() + "=" + valor);

            if (GlobalOPC.permiteEscribir(IDNodo.getValue().toString()))
            {
            // Find the DataType if setting Value - for other properties you must
                // find the correct data type yourself
                UaDataType dataType = null;
                if (node instanceof UaVariable)
                {
                    UaVariable v = (UaVariable) node;
                    // Initialize DataType node, if it is not initialized yet
                    if (v.getDataType() == null)
                    {
                        v.setDataType(opc.client.getAddressSpace().getType(v.getDataTypeId()));
                    }
                    dataType = (UaDataType) v.getDataType();
                }
                try
                {
                    Object convertedValue = dataType != null ?
opc.client.getAddressSpace().getDataTypeConverter().parseVariant(valor, dataType)
                            : valor;

                    boolean status = opc.client.writeAttribute(IDNodo, Attributes.Value, convertedValue);
                    if (status)
                    {
                        Global.info.Registra("OK", tipoInfo.INFO);
                    }
                    else
                    {
                        Global.info.Registra("OK (completes asynchronously)", tipoInfo.INFO);
                    }
                    setValor(new DataValue((Variant) convertedValue));
                }
                catch (ServiceException | StatusException e)
                {
                    Global.info.Registra(e);
                    throw new RuntimeException(e);
                }
            }
        }
```

```
    /**
     * Método que lee un valor del tag OPC
     *
     * @return el valor del tag
     * @throws com.prosysopc.ua.ServiceException
     * @throws com.prosysopc.ua.StatusException
     */
    public String Leer() throws ServiceException, StatusException
    {
        DataValue value = opc.client.readAttribute(IDNodo, Attributes.Value);
        if (value.getStatusCode() == StatusCode.GOOD)
        {
            setValor(value);
            return value.getValue().getValue().toString();
        }
        else
        {
            return "";
        }
    }

    /**
     * @return la banda muerta del tag
     */
    public double getDeadband()
    {
        return deadband;
    }

    /**
     * @param deadband Fija la banda muerta del tag
     */
    public void setDeadband(double deadband)
    {
        this.deadband = deadband;
    }
}
```

1.4.4 Clase OPCUA

```
package com.opc;

import com.Global;
import com.prosysopc.ua.ApplicationIdentity;
import com.prosysopc.ua.CertificateValidationListener;
import com.prosysopc.ua.PkiFileBasedCertificateValidator.CertificateCheck;
import com.prosysopc.ua.PkiFileBasedCertificateValidator.ValidationResult;
import com.prosysopc.ua.SecureIdentityException;
import com.prosysopc.ua.ServiceException;
import com.prosysopc.ua.client.Subscription;
import com.prosysopc.ua.client.UaClient;
import java.io.IOException;
import java.net.URISyntaxException;
import java.util.Iterator;
import java.util.List;
import org.opcfoundation.ua.builtintypes.ExtensionObject;
import org.opcfoundation.ua.core.ReadValueId;
import com.prosysopc.ua.SessionActivationException;
import com.prosysopc.ua.StatusException;
import com.prosysopc.ua.UserIdentity;
import com.prosysopc.ua.client.AddressSpaceException;
import com.prosysopc.ua.client.InvalidServerEndpointException;
import com.prosysopc.ua.client.MonitoredDataItem;
import com.prosysopc.ua.client.MonitoredDataItemListener;
import com.prosysopc.ua.client.MonitoredEventItem;
import com.prosysopc.ua.client.ServerConnectionException;
import com.prosysopc.ua.client.ServerStatusListener;
import com.prosysopc.ua.client.SubscriptionAliveListener;
import com.prosysopc.ua.client.SubscriptionNotificationListener;
import com.prosysopc.ua.nodes.UaDataType;
import com.prosysopc.ua.nodes.UaNode;
import com.prosysopc.ua.nodes.UaVariable;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.EnumSet;
import java.util.Locale;
import java.util.StringTokenizer;
import org.opcfoundation.ua.builtintypes.DataValue;
import org.opcfoundation.ua.builtintypes.DiagnosticInfo;
import org.opcfoundation.ua.builtintypes.LocalizedText;
import org.opcfoundation.ua.builtintypes.NodeId;
import org.opcfoundation.ua.core.Attributes;
import org.opcfoundation.ua.builtintypes.StatusCode;
import org.opcfoundation.ua.builtintypes.UnsignedInteger;
import org.opcfoundation.ua.builtintypes.Variant;
import org.opcfoundation.ua.common.ServiceResultException;
import org.opcfoundation.ua.core.ApplicationDescription;
import org.opcfoundation.ua.core.ApplicationType;
import org.opcfoundation.ua.core.DataChangeFilter;
import org.opcfoundation.ua.core.DataChangeTrigger;
```

```
import org.opcfoundation.ua.core.DeadbandType;
import org.opcfoundation.ua.core.Identifiers;
import org.opcfoundation.ua.core.MonitoringMode;
import org.opcfoundation.ua.core.NotificationData;
import org.opcfoundation.ua.core.ReferenceDescription;

import org.opcfoundation.ua.core.TimestampsToReturn;
import org.opcfoundation.ua.transport.security.Cert;
import org.opcfoundation.ua.transport.security.SecurityMode;

/**
 * Clase que implementa la conexión OPC UA y hace de interfaz con el OPC UA
 * Client SDK de Prosys
 *
 * @author SONIA
 */
public class OPCUA
{

    /**
     * Lista de grupos de tags (identificadores de valores a leer) para la
     * lectura agrupada
     */
    private List<List<ReadValueId>> rvis;
    /**
     * Lista de grupos de tags (nodos) para la lectura agrupada
     */
    public List<List<NodeId>> listaNodos;
    /**
     * Cliente OPC UA
     */
    public UaClient client;
    /**
     * Suscripción
     */
    private Subscription subscription;
    /**
     * Lista de grupos de tags (dentificadores de valores a leer) para la
     * lectura agrupada
     */
    private final boolean showReadValueDataType = true;
    /**
     * Lista de items OPC
     */
    private final List<ItemInfo> variablesOPC;
    /**
     * Lista de items OPC aún no conectados
     */
    public final List<ItemInfo> variablesOPCDesligadas;
    /**
     * Número de grupos OPC
     */
    int numeroGruposOPC;
    /**
     * Escuchador de evento de cambio en variables OPC suscritas.
```

```
 */
private final MonitoredDataItemListener dataChangeListener;

/**
 * Nombre de la aplicación
 */
String APP_NAME;
/**
 * URL del servidor
 */
String URL;

/**
 * Escuchador de evento de cambio en el estado del servidor OPC
 */
private final ServerStatusListener serverStatusListener;

private boolean conectado = false;

/**
 * Método que comprueba si el array de valores leídos está OK
 *
 * @param datos Array de valores opc leídos
 * @return True si todos OK
 */
public boolean isOK(DataValue[] datos)
{
   for (DataValue dato : datos)
   {
      if (!dato.getStatusCode().isGood())
      {
         return false;
      }
   }
   return true;
}

/**
 * Método que comprueba si el cliente está conectado
 *
 * @return True si conectado
 */
public boolean isConectado()
{
   return conectado;
}

/**
 * Método que obtiene el item según su descripción
 *
 * @param descr Descripción del tag
 * @return Item OPC
 */
public ItemInfo getItemInfo(String descr)
{
```

```
    for (int i = 0; i < variablesOPC.size(); i++)
    {
        if (variablesOPC.get(i).getDescripcion().equals(descr))
        {
            return variablesOPC.get(i);
        }
    }
    return null;
}

/**
 * Método que obtiene el item según su índice
 *
 * @param i Índice del item OPC
 * @return Item OPC
 */
public ItemInfo getItemInfo(int i)
{
    if (i >= 0 && i < variablesOPC.size())
    {
        return variablesOPC.get(i);
    }
    else
    {
        return null;
    }
}

/**
 * Método que obtiene número de tags en un grupo
 *
 * @param grupo Índice del grupo
 * @return Número de items en el grupo
 */
public int getNumPorGrupo(int grupo)
{
    int n = 0;
    for (int i = 0; i < variablesOPC.size(); i++)
    {
        if (variablesOPC.get(i).lecturaAgrupada == grupo)
        {
            n++;
        }
    }
    return n;
}

/**
 * Constructor de tag OPC UA
 *
 * @param APP_NAME Nombre de la aplicación
 * @param URL URL del servidor
 * @param dataChangeListener Escuchador de evento de cambio en variables OPC
 * suscritas.
 * @param serverStatusListener Escuchador de evento de cambios de estado del
```

```
         * servidor
         */
        public OPCUA(String APP_NAME, String URL, MonitoredDataItemListener
dataChangeListener, ServerStatusListener serverStatusListener)
        {
           this.dataChangeListener = dataChangeListener;
           this.serverStatusListener = serverStatusListener;
           this.variablesOPC = new ArrayList<>();
           this.variablesOPCDesligadas = new ArrayList<>();
           this.numeroGruposOPC = 0;
           this.APP_NAME = APP_NAME;
           this.URL = URL;
        }

        /**
         * Método que obtiene el nodo a parir del nombre
         *
         * @param nombre Nombre del nodo
         * @return Nodo Prosys del tag
         */
        private NodeId getNodeId(String nombre)
        {
           for (ItemInfo item : variablesOPC)
           {
              if (item.getNombre().equals(nombre))
              {
                 return item.getIDNodo();
              }
           }
           return null;
        }

        /**
         * Método que lee del servidor el valor de un tag a partir de su nodo
         *
         * @param nodeId Nodo
         * @return Valor del tag
         */
        String Leer(NodeId nodeId) throws ServiceException, StatusException
        {
           DataValue value = client.readAttribute(nodeId, Attributes.Value);

           if (value.getStatusCode() == StatusCode.GOOD)
           {
              return value.getValue().getValue().toString();
           }
           else
           {
              return "";
           }
        }

        /**
         * Método que lee del servidor el valor de un tag a partir de su nombre
         *
```

```
 * @param tag Nombre del tag
 * @return Valor del tag
 * @throws com.prosysopc.ua.ServiceException
 * @throws com.prosysopc.ua.StatusException
 */
public String Leer(String tag) throws ServiceException, StatusException
{
   return Leer(getNodeId(tag));
}

/**
 * Método que escribe en el servidor el valor de un tag a partir de su nodo
 *
 * @param nodeId Nodo
 * @param valor Valor a escribir
 */
void EscribeItem(NodeId nodeId, String valor) throws ServiceException,
      AddressSpaceException, StatusException
{
   UaNode node = client.getAddressSpace().getNode(nodeId);
   Global.info.Info("Escribiendo OPC  en " + nodeId.getValue().toString() + "=" + valor);
   //if (nodeId.getValue().toString().contains("receta.BD5.receta"))
   if (GlobalOPC.permiteEscribir(nodeId.getValue().toString()))
   {
      // Find the DataType if setting Value - for other properties you must
      // find the correct data type yourself
      UaDataType dataType = null;
      if (node instanceof UaVariable)
      {
         UaVariable v = (UaVariable) node;
         // Initialize DataType node, if it is not initialized yet
         if (v.getDataType() == null)
         {
            v.setDataType(client.getAddressSpace().getType(v.getDataTypeId()));
         }
         dataType = (UaDataType) v.getDataType();
      }
      try
      {
         Object convertedValue = dataType != null ?
        client.getAddressSpace().getDataTypeConverter().parseVariant(valor, dataType)
              : valor;

         boolean status = client.writeAttribute(nodeId, Attributes.Value, convertedValue);
         if (status)
         {
            Global.info.Info("OK");
         }
         else
         {
            Global.info.Info("OK (completes asynchronously)");
         }
      }
      catch (ServiceException | StatusException e)
      {
```

```
            Global.info.Registra(e);
            throw new RuntimeException(e);
        }
    }
}

/**
 * Método que escribe en el servidor el valor de un tag a partir del nombre
 * del tag
 *
 * @param tag Nombre del tag
 * @param valor Valor a escribir
 * @throws com.prosysopc.ua.ServiceException
 * @throws com.prosysopc.ua.client.AddressSpaceException
 * @throws com.prosysopc.ua.StatusException
 */
public void Escribe(String tag, String valor) throws ServiceException,
        AddressSpaceException, StatusException
{
    EscribeItem(getNodeId(tag), valor);
}

/**
 * Método que obtiene el nombre de un item a partir de su nodo
 *
 * @param nodeId Nodo
 * @return Nombre del item
 */
private String getNameFromNodeId(NodeId nodo)
{
    for (ItemInfo item : variablesOPC)
    {
        if (item.getIDNodo() == nodo)
        {
            return item.getNombre();
        }
    }
    return "";
}
/**
 * Escuchador del Validador de Certificado
 *
 */
  private CertificateValidationListener validationListener = (Cert certificate,
ApplicationDescription applicationDescription, EnumSet<CertificateCheck> passedChecks) ->
{
    Global.info.Info("Certificado del servidor: Sujeto = "
            + certificate.getCertificate().getSubjectX500Principal().toString()
            + ", Expedido = " + certificate.getCertificate().getIssuerX500Principal().toString()
            + ", Válido desde =" + certificate.getCertificate().getNotBefore().toString()
            + " hasta = " + certificate.getCertificate().getNotAfter().toString());

    if (!passedChecks.contains(CertificateCheck.Signature))
    {
        Global.info.Aviso("El certificado NO ESTÁ FIRMADO de forma fiable");
```

```
        }

        if (!passedChecks.contains(CertificateCheck.Validity))
        {
            Date today = new Date();
            final boolean isYoung = certificate.getCertificate().getNotBefore().compareTo(today) > 0;
            final boolean isOld = certificate.getCertificate().getNotAfter().compareTo(today) < 0;

            final String oldOrYoung = isOld ? "(anymore)" : (isYoung ? "(yet)" : "");
            Global.info.Aviso("El intervalo del certificado no es válido =" + oldOrYoung.toString());
        }

        if (!passedChecks.contains(CertificateCheck.Uri))
        {
                    Global.info.Aviso("La URI del certificado no se corresponde con la URI del
            servidor =" + applicationDescription.getApplicationUri().toString());
        }

        if (!passedChecks.contains(CertificateCheck.SelfSigned))
        {
            Global.info.Aviso("El certificado está firmado por si mismo");
        }
        // Aceptar el certificado si es confiable, válido y verificado

        if (passedChecks.containsAll(CertificateCheck.COMPULSORY))
        {
            return ValidationResult.AcceptPermanently;
        }
        else
        {
            return ValidationResult.Reject;
        }
    };

    /**
     * Escuchador de la vigencia de la suscripción
     *
     */
      private final SubscriptionAliveListener subscriptionAliveListener = new
SubscriptionAliveListener()
    {
        @Override
        public void onAlive(Subscription s)
        {
            // El cliente reconoce que la conexión está viva, aunque no hay cambios para enviar
            /*Logger.getLogger(Principal.class.getName()).info("Suscripción viva: ID="
             + s.getSubscriptionId() + " lastAlive=" + timestampToString(s.getLastAlive()));*/
        }

        @Override
        public void onTimeout(Subscription s)
        {
            // El cliente no reconoció que la conexión está viva y maxKeepAliveCount ha sido excedido
            Global.info.Info("Suscripción timeout: ID" + s.getSubscriptionId()
                    + " lastAlive=" + timestampToString(s.getLastAlive()));
```

```
    }

    private String timestampToString(Calendar lastAlive)
    {
                return lastAlive == null ? "<never>" :
        SimpleDateFormat.getDateTimeInstance().format(lastAlive.getTime());

    }
};

/**
 * Escuchador de la notificación de la suscripción
 *
 */
        private final SubscriptionNotificationListener subscriptionListener = new
        SubscriptionNotificationListener()
{
    @Override
    public void onDataChange(Subscription s, MonitoredDataItem mdi, DataValue value)
    {
    }

    @Override
    public void onError(Subscription s, Object o, Exception excptn)
    {
      throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void onEvent(Subscription s, MonitoredEventItem mei, Variant[] vrnts)
    {

    }

    @Override
    public void onNotificationData(Subscription s, NotificationData nd)
    {

    }

    @Override
    public void onStatusChange(Subscription s, StatusCode sc, StatusCode sc1, DiagnosticInfo
di)
    {

    }

    @Override
            public long onMissingData(UnsignedInteger lastSequenceNumber, long
        sequenceNumber, long newSequenceNumber, StatusCode serviceResult)
    {
      throw new UnsupportedOperationException("Not supported yet.");
    }
```

```
    @Override
            public void onBufferOverflow(Subscription subscription, UnsignedInteger
        sequenceNumber, ExtensionObject[] notificationData)
    {
        throw new UnsupportedOperationException("Not supported yet.");
    }
};

/**
 * Método que añade todos los items de la lista provisional
 * a una conexión cliente OPC UA
 *
 * @throws com.prosysopc.ua.ServiceException
 * @throws com.prosysopc.ua.StatusException
 * @throws org.opcfoundation.ua.common.ServiceResultException
 * @throws com.opc.ExcepcionOPC
 */
        public void ligaVariablesOPC() throws ServiceException, StatusException,
        ServiceResultException, ExcepcionOPC
{
    for (ItemInfo item : variablesOPCDesligadas)
    {
        ligaVariableOPC(item);
    }
    variablesOPCDesligadas.clear();
}

/**
 * Método que añade un item OPC a una conexión cliente OPC UA.
 * Busca su dirección en el espacio de
 * direcciones del servidor OPC y obtiene su nodo que liga al objeto
 * ItemInfo. Si tiene espeficado un grupo de lectura agrupada lo añade a la
 * lista de tags de lectura agrupada, como nodo y como identificador de
 * valor leído. Si se le ha especificado suscripción a cambios de valor,
 * realiza la suscripción del item
 *
 * @param item Item OPC
 * @return True si fue encontrado en el espacio de direcciones del servidor
 * @throws com.prosysopc.ua.ServiceException
 * @throws com.prosysopc.ua.StatusException
 * @throws org.opcfoundation.ua.common.ServiceResultException
 * @throws com.opc.ExcepcionOPC
 */
public boolean ligaVariableOPC(ItemInfo item) throws ServiceException,
      StatusException, ServiceResultException, ExcepcionOPC
{
    client.getAddressSpace().setMaxReferencesPerNode(1000);
    //client.getAddressSpace().setReferenceTypeId(Identifiers.);
            List<ReferenceDescription> references =
        client.getAddressSpace().browse(Identifiers.RootFolder);

    boolean encontrado = false;
    NodeId nodeId = null;

    for (int i = 0; i < references.size(); i++)
```

```
{
   ReferenceDescription r = references.get(i);
   String temporal = r.getBrowseName().getName().toString();

   if (temporal.equals("Objects"))
   {
      nodeId = client.getAddressSpace().getNamespaceTable().toNodeId(r.getNodeId());
      encontrado = true;
      break;
   }
}
if (!encontrado)
{
   return false;
}
else
{
   StringTokenizer st = new StringTokenizer(item.getNombre(), ".");
   while (st.hasMoreTokens())
   {
      String nombre = st.nextToken();
      client.getAddressSpace().setMaxReferencesPerNode(1000);
      //client.getAddressSpace().setReferenceTypeId(Identifiers.);
      references = client.getAddressSpace().browse(nodeId);
      encontrado = false;

      for (int i = 0; i < references.size(); i++)
      {
         ReferenceDescription r = references.get(i);
         String temporal = r.getBrowseName().getName().toString();

         if (temporal.equals(nombre))
         {
            encontrado = true;
            nodeId = client.getAddressSpace().getNamespaceTable().toNodeId(r.getNodeId());
            item.setIDNodo(nodeId);
            break;
         }
      }
      if (!encontrado)
      {
         Global.info.Error("Conexión OPC: " + item.getNombre() + " no encontrado");
         throw new ExcepcionOPC("Variable OPC no encontrada: " + item.getNombre());
         //return false;
      }
   }
   if (item.lecturaAgrupada >= 0 && item.lecturaAgrupada < numeroGruposOPC)
   {
      encontrado = false;

      for (ReadValueId rv : rvis.get(item.lecturaAgrupada))
      {
         if (rv.getNodeId() == nodeId)
         {
            encontrado = true;
```

```
                    break;
                }
            }
            if (!encontrado)
            {
                rvis.get(item.lecturaAgrupada).add(new ReadValueId(nodeId, Attributes.Value, null,
                null));
                listaNodos.get(item.lecturaAgrupada).add(nodeId);
            }
        }
        //System.out.println("Por aquí 4 " + item.getNombre());
        if (!subscription.hasItem(nodeId, Attributes.Value) && item.notificacionCambio)
        {
            MonitoredDataItem dataItem = new MonitoredDataItem(nodeId,  Attributes.Value,
            MonitoringMode.Reporting);
            dataItem.addChangeListener(dataChangeListener);

            if (item.getDeadband() > 0)
            {
                DataChangeFilter filter = new DataChangeFilter();
                filter.setDeadbandValue(item.getDeadband());
                filter.setTrigger(DataChangeTrigger.StatusValue);

            //filter.setDeadbandType(UnsignedInteger.valueOf(DeadbandType.Percent.getValue()));

            filter.setDeadbandType(UnsignedInteger.valueOf(DeadbandType.Absolute.getValue()));
                dataItem.setDataChangeFilter(filter);
            }

            subscription.addItem(dataItem);

        }
        if (!client.hasSubscription(subscription.getSubscriptionId()))
        {
            client.addSubscription(subscription);
        }
        subscription.setPublishingEnabled(true);
        return true;
    }
}

/**
 * Método que devuelve un array de números en coma flotante flotantes a
 * partir de un array de valores OPC leídos
 *
 * @param valores Array de valores OPC leídos
 * @return array de números en coma flotante
 */
public double[] toDoubleArray(DataValue[] valores)
{
    try
    {
        double[] v = new double[valores.length];
        for (int i = 0; i < valores.length; i++)
        {
```

```
            v[i] = valores[i].getValue().doubleValue();
        }
        return v;
    }
    catch (NullPointerException | NumberFormatException ex)
    {
        return null;
    }
}

/**
 * Método que lee un grupo de items
 *
 * @param items Lista de items a leer
 * @return array de valores OPC leídos
 * @throws com.prosysopc.ua.ServiceException
 */
public DataValue[] leerGrupoOPC(ItemInfo[] items) throws ServiceException
{
    NodeId[] nid = new NodeId[items.length];
    for (int i = 0; i < items.length; i++)
    {
        nid[i] = items[i].getIDNodo();
    }
    return client.readValues(nid, TimestampsToReturn.Both);
}

/**
 * Método que lee un grupo de items
 *
 * @param g Índice del grupo a leer
 * @return array de valores OPC leídos
 * @throws com.prosysopc.ua.ServiceException
 */
public DataValue[] leerGrupoOPC(int g) throws ServiceException
{
    if (g >= numeroGruposOPC)
    {
        return null;
    }
            return client.readValues(listaNodos.get(g).toArray(new
        NodeId[listaNodos.get(g).size()]), TimestampsToReturn.Both);
}

/**
 * Método que escribe un grupo de items
 *
 * @param g Índice del grupo a escribir
 * @param datos Valores a escribir
 * @return Códigos de estado
 * @throws com.prosysopc.ua.ServiceException
 * @throws com.prosysopc.ua.StatusException
 */
            public StatusCode[] escribeGrupoOPC(int g, DataValue[] datos) throws
        ServiceException, StatusException
```

```
{
   if (g >= numeroGruposOPC)
   {
      return null;
   }
   if (GlobalOPC.permiteEscribir(""))
   {
                  return client.writeValues(listaNodos.get(g).toArray(new
        NodeId[listaNodos.get(g).size()]), datos);
   }
   else
   {
      return null;
   }
}

/**
 * Método que añade un item a la lista provisional
 *
 * @param item Item OPC a añadir
 * @return Índice del item
 * @throws com.prosysopc.ua.ServiceException
 * @throws com.prosysopc.ua.StatusException
 * @throws org.opcfoundation.ua.common.ServiceResultException
 * @throws com.opc.ExcepcionOPC
 */
  public int AnnadeItem(ItemInfo item) throws ServiceException, StatusException,
ServiceResultException, ExcepcionOPC
{
   for (int i = 0; i < variablesOPC.size(); i++)
   {
      if (variablesOPC.get(i).getNombre().equals(item.getNombre())
            && variablesOPC.get(i).lecturaAgrupada == item.lecturaAgrupada)
      {
         item.setIndice(variablesOPC.get(i).getIndice());
         return variablesOPC.get(i).getIndice();
      }
   }
   variablesOPC.add(item);
   variablesOPCDesligadas.add(item);
   if (item.lecturaAgrupada + 1 > numeroGruposOPC)
   {
      numeroGruposOPC = item.lecturaAgrupada + 1;
   }
   if (item.lecturaAgrupada >= 0)
   {
      item.setIndice(getNumPorGrupo(item.lecturaAgrupada) - 1);
   }
   else
   {
      item.setIndice(-1);
   }
   return item.getIndice();
}
```

```
/**
 * Método que añade un array de items a la lista provisional
 *
 * @param items Array de items OPC a añadir
 * @throws com.prosysopc.ua.ServiceException
 * @throws com.prosysopc.ua.StatusException
 * @throws org.opcfoundation.ua.common.ServiceResultException
 * @throws com.opc.ExcepcionOPC
 */
  public void AnnadeItem(ItemInfo[] items) throws ServiceException, StatusException,
ServiceResultException, ExcepcionOPC
{
   for (ItemInfo item : items)
   {
      item.setOpc(this);
      //if(items[i].getNombre().contains("posicion2"))
      AnnadeItem(item);
   }
}

/**
 * Método que devuelve el número de items añadidos
 *
 * @return Número de items
 */
public int getSizeItems()
{
   return variablesOPC.size();
}

/**
 * Método que desconecta el cliente OPC
 *
 */
public void Desconecta()
{
   if (client != null)
   {
      client.disconnect();
   }
}

/**
 * Método que dispone las listas de lectura agrupada para un nuevo grupo y
 * devuelve el índice de grupo
 *
 * @return Índice de grupo
 */
public int addGrupo(int grupo)
{
   if (grupo < 0)
   {
      numeroGruposOPC++;
      rvis.add(new ArrayList<>());
      listaNodos.add(new ArrayList<>());
```

```
            return numeroGruposOPC - 1;
        }
        else
        {
            return grupo;
        }
    }

    /**
     * Método que conecta el cliente OPC al que añade los items de la lista
     * provisional
     *
     * @return True si conexión exitosa
     * @throws com.prosysopc.ua.SessionActivationException
     * @throws java.net.URISyntaxException
     * @throws com.prosysopc.ua.SecureIdentityException
     * @throws java.io.IOException
     * @throws com.opc.ExcepcionOPC
     * @throws com.prosysopc.ua.StatusException
     * @throws org.opcfoundation.ua.common.ServiceResultException
     */
            public boolean Conecta() throws SessionActivationException, URISyntaxException,
            SecureIdentityException, IOException, ServiceException, StatusException,
            ServiceResultException, ExcepcionOPC
    {
        if (conectado)
        {
            return true;
        }

        rvis = new ArrayList<>();
        listaNodos = new ArrayList<>();

        for (int ng = 0; ng < numeroGruposOPC; ng++)
        {
            rvis.add(new ArrayList<>());
            listaNodos.add(new ArrayList<>());
        }

        //client = new UaClient("opc.tcp://172.19.7.79:49320");
        client = new UaClient(URL);
        // Ficheros PKI para hacer seguimiento del servidores fiables o rechazados
        //final PkiFileBasedCertificateValidator validator = new PkiFileBasedCertificateValidator();

        //client.setCertificateValidator(validator);
        // ...y reaccionar a los resultados de validación con un manejador a medida
        //validator.setValidationListener(validationListener);
        // ApplicationDescription es enviado al servidor...
        ApplicationDescription appDescription = new ApplicationDescription();

        appDescription.setApplicationName(new LocalizedText(APP_NAME, Locale.ENGLISH));
                    // 'localhost' (en minúsculas) en la URI es convertido al nombre host del ordenador
            en el que se ejecuta la aplicación
        appDescription.setApplicationUri("urn:localhost:UA:MiOPCUA");
        appDescription.setProductUri("urn:prosysopc.com:UA:MiOPCUA");
```

```
appDescription.setApplicationType(ApplicationType.Client);

// Definir la identidad del cliente , incluyendo el certificado de seguridad
//final ApplicationIdentity identity =
ApplicationIdentity.loadOrCreateCertificate(appDescription, "Sample Organisation",
//       /* Clave de Seguridad Privada */ null,
//       /* Ruta del Fichero Clave*/ new File(validator.getBaseDir(), "private"),
//       /* Habilitar la renovación del certificado*/ true);
// Definir la identidad del cliente , incluyendo el certificado de seguridad
final ApplicationIdentity identity = new ApplicationIdentity();
identity.setApplicationDescription(appDescription);

client.setApplicationIdentity(identity);

// Definir a nuestro usuario como local. La de defecto es Locale.getDefault()
client.setLocale(Locale.ENGLISH);

// Definir la temporización de la comunicación por defecto en milisegundos
client.setTimeout(10000);
client.setStatusCheckTimeout(50000);

// EScuchar los cambios de estado del servidor
client.addServerStatusListener(serverStatusListener);

// Definir el modo de seguridad
// - Por defecto (in UaClient) es BASIC128RSA15_SIGN_ENCRYPT
// client.setSecurityMode(SecurityMode.BASIC128RSA15_SIGN_ENCRYPT);
// client.setSecurityMode(SecurityMode.BASIC128RSA15_SIGN);
// client.setSecurityMode(SecurityMode.NONE);
client.setSecurityMode(SecurityMode.NONE);

// Si el servidor soporta la verificación de usuario, puedes fijar la identidad del servidor
// - Por defecto Anonymous
client.setUserIdentity(new UserIdentity());
// - Utiliza la verificación username/password (observa que requiere seguridad
// client.setUserIdentity(new UserIdentity("my_name", "my_password"));
// - Lee el certificado de usuario y la clave privada de los ficheros
// client.setUserIdentity(new UserIdentity(new
// java.net.URL("my_certificate.der"),
// new java.net.URL("my_privatekey.pfx"), "my_privatekey_password"));

try
{
   Global.info.Info("Utilizando SecurityMode " + client.getSecurityMode());
   client.connect();
   try
   {
      Global.info.Info("ServerStatus: " + client.getServerStatus());
   }
   catch (StatusException ex)
   {
      Global.info.Registra(ex);
      throw new RuntimeException(ex);
   }
}
```

```
catch (InvalidServerEndpointException e)
{
   Global.info.Registra(e);
   /*try
    {
    // In case we have selected a wrong endpoint, print out the
    // supported ones
    println("Endpoints supported by the server");
    for (EndpointDescription e : endpoints)
    {
      println(String.format("%s [%s,%s]", e.getEndpointUrl(), e.getSecurityPolicyUri(),
      e.getSecurityMode()));
    }
    }
    catch (Exception ex)
    {
    // never mind, if the endpoints are not available
    }*/
   throw new RuntimeException(e);
}
catch (ServerConnectionException e)
{
   Global.info.Registra(e);
   /*try
    {
    // En caso de haber seleccionado un modo de seguridda no disponible, imprimir los modos
    soportados
    println("SecurityModes supported by the server:");
    for (SecurityMode m : supportedSecurityModes)
    {
    println(m.toString());
    }
    }
    catch (ServerConnectionException e1)
    {
    // never mind, if the security modes are not available
    }
    catch (ServiceException e2)
    {
    // never mind, if the security modes are not available
    }*/
   throw new RuntimeException(e);
}
catch (SessionActivationException e)
{
   Global.info.Registra(e);
   /*try
    {
    println("The server supports the following user tokens:");
    for (UserTokenPolicy p : client.getSupportedUserIdentityTokens())
    {
    println(p.getTokenType().toString());
    }
    }
    catch (ServiceException e1)
```

```
            {
            // never mind, if not available
            }*/
            throw new RuntimeException(e);
        }
        catch (ServiceException e)
        {
            Global.info.Registra(e);
            throw new RuntimeException(e);
        }
        if (subscription
                == null)
        {
            subscription = new Subscription();
            // PublishingInterval por defecto es 1000 ms
            // subscription.setPublishingInterval(100);

            // EScuchar a los eventos alive y timeout events de la suscripción
            subscription.addAliveListener(subscriptionAliveListener);
            // Listen to notifications - the data changes and events are
            // handled using the item listeners (see below), but in many
            // occasions, it may be best to use the subscription
            // listener also to handle those notifications
            subscription.addNotificationListener(subscriptionListener);
        }

        Iterator iter = variablesOPC.iterator();

        while (iter.hasNext())
        {
            ligaVariableOPC((ItemInfo) iter.next());
        }
        conectado = true;
        return true;
    }
}
```

1.5 HOT II

1.5.1 Clase BISRA

```
package hot;

public class BISRA
{

   /**
    * Array de conductividades térmicas de acero en función del código BISRA y la temperatura
    */
   static double[][] K = new double[][]
   {
      new double[]
      {
         65.302, 62.791, 60.279, 57.767, 55.674, 53.163, 51.07, 48.558, 46.465, 43.535, 41.023,
         39.349, 37.674, 36.0, 33.907, 31.814, 30.14, 27.628, 27.209, 27.209, 27.628, 28.047, 28.465,
         29.302, 29.721, 30.14, 30.558, 30.977
      },
      new double[]
      {
         59.442//, 58.605, 57.767, 55.256, 53.581, 51.488, 49.395, 47.721, 44.791, 42.279, 40.186,
         38.093, 36.0, 33.907, 31.814, 29.721, 28.465, 27.209, 26.791, 27.209, 27.628, 28.047,
         28.465, 29.302, 29.721, 30.14, 30.558, 30.977
      },
      new double[]
      {
         51.907, 51.488, 51.07, 49.814, 48.558, 46.465, 44.372, 43.535, 42.698, 41.023, 39.349,
         37.674, 35.581, 33.907, 31.814, 28.465, 25.953, 25.953, 26.372, 26.791, 27.209, 28.047,
         28.465, 29.302, 29.721, 30.14, 30.558, 30.977
      },
      new double[]
      {
         51.907, 51.488, 50.651, 49.814, 48.14, 46.884, 45.628, 44.372, 41.86, 40.186, 38.093, 36.0,
         33.907, 32.233, 30.14, 27.209, 24.698, 24.698, 25.535, 25.953, 26.791, 27.209, 28.047,
         28.884, 29.721, 30.558, 31.395, 32.233
      },
      new double[]
      {
         48.14, 48.14, 48.14, 47.721, 46.465, 45.209, 43.953, 43.116, 41.023, 39.767, 38.512, 36.837,
         36.0, 33.488, 31.395, 28.465, 26.791, 25.535, 25.953, 25.953, 26.791, 27.209, 28.047,
         28.884, 29.721, 30.558, 31.395, 32.233
      },
      new double[]
      {
         49.814, 49.395, 48.14, 46.884, 45.209, 43.116, 41.442, 40.186, 38.093, 36.419, 35.163,
         33.907, 32.651, 31.395, 30.14, 26.791, 24.279, 24.279, 25.535, 26.372, 26.791, 27.628,
         28.465, 29.302, 30.14, 30.977, 31.814, 32.651
      },
      new double[]
      {
```

```
    51.07, 50.233, 48.977, 47.721, 46.047, 43.953, 41.86, 40.186, 38.512, 36.837, 35.581,
    34.326, 33.07, 31.814, 30.558, 27.209, 24.279, 24.279, 25.116, 25.953, 26.791, 27.628,
    28.465, 28.884, 29.721, 30.558, 31.395, 32.233
},
new double[]
{
    45.209, 45.209, 44.791, 44.791, 42.698, 41.023, 40.186, 38.512, 37.256, 36.0, 34.744,
    33.488, 31.814, 30.14, 28.465, 27.209, 23.86, 23.86, 24.698, 25.535, 25.953, 26.791, 27.209,
    28.047, 28.465, 28.884, 29.302, 29.721
},
new double[]
{
    46.047, 46.047, 46.465, 45.628, 44.791, 43.953, 43.116, 41.442, 40.186, 38.512, 37.256,
    36.0, 34.326, 32.651, 31.395, 30.558, 29.721, 26.372, 26.791, 26.791, 27.209, 28.047,
    28.465, 29.302, 29.721, 30.14, 30.558, 30.977
},
new double[]
{
    36.419, 37.256, 37.674, 38.093, 38.512, 39.349, 38.093, 37.256, 36.419, 35.581, 34.326,
    33.07, 31.814, 30.14, 28.465, 25.116, 25.116, 25.535, 26.372, 26.791, 27.628, 28.047,
    28.465, 29.302, 30.14, 30.977, 31.814, 32.651
},
new double[]
{
    34.326, 35.163, 36.0, 36.419, 36.837, 36.837, 36.837, 36.837, 36.419, 35.581, 35.163,
    34.326, 33.488, 31.814, 29.302, 26.791, 25.953, 25.953, 26.372, 27.209, 27.628, 28.047,
    28.884, 29.302, 29.709, 30.14, 30.558, 30.977
},
new double[]
{
    33.488, 34.744, 35.581, 36.0, 36.419, 36.419, 36.419, 36.419, 36.419, 35.581, 34.326, 33.07,
    31.814, 30.14, 28.465, 26.791, 25.953, 25.953, 26.372, 27.209, 27.628, 28.047, 28.884,
    29.302, 29.721, 30.14, 30.558, 30.977
},
new double[]
{
    33.07, 33.488, 33.907, 34.744, 35.163, 35.581, 35.581, 35.581, 35.581, 34.744, 33.488,
    32.233, 30.558, 29.302, 28.047, 27.209, 26.791, 27.209, 27.628, 28.047, 28.465, 28.884,
    29.302, 29.721, 30.14, 30.558, 30.977, 31.395
},
new double[]
{
    48.558, 47.721, 46.465, 45.209, 44.372, 43.535, 42.279, 40.605, 38.512, 37.256, 35.581,
    33.907, 31.814, 30.14, 28.884, 27.209, 25.953, 25.953, 26.791, 27.628, 28.047, 28.465,
    28.884, 29.721, 30.14, 30.558, 30.977, 31.395
},
new double[]
{
    43.116, 42.698, 42.698, 42.279, 41.86, 41.442, 40.605, 39.767, 38.93, 37.674, 36.419,
    35.163, 33.907, 32.651, 30.977, 28.465, 26.372, 26.372, 27.209, 27.628, 28.047, 28.884,
    29.302, 29.721, 30.14, 30.558, 30.977, 31.395
},
new double[]
{
```

```
        25.116, 26.791, 28.465, 29.302, 30.14, 30.977, 30.977, 30.977, 30.977, 30.977, 30.977,
        30.558, 30.14, 29.302, 28.047, 26.372, 25.116, 25.116, 25.535, 25.953, 26.372, 27.209,
        27.628, 28.465, 29.302, 30.14, 30.977, 31.814
    },
    new double[]
    {
        12.977, 13.814, 14.651, 15.488, 16.326, 17.163, 18.0, 18.837, 19.256, 19.674, 20.512,
        21.349, 21.767, 22.186, 22.605, 23.023, 23.442, 23.86, 24.279, 24.698, 25.535, 25.953,
        26.791, 27.209, 28.047, 28.884, 29.721, 30.558
    },
    new double[]
    {
        12.558, 13.814, 14.651, 15.488, 16.326, 16.744, 17.581, 18.419, 18.837, 19.674, 20.512,
        21.349, 22.186, 22.605, 23.442, 24.279, 25.116, 25.953, 26.372, 27.209, 27.628, 28.047,
        28.465, 28.884, 29.721, 30.558, 31.395, 32.233
    },
    new double[]
    {
        15.907, 15.907, 16.326, 16.744, 17.163, 17.581, 18.419, 19.256, 20.093, 20.93, 21.767,
        23.023, 23.86, 24.698, 25.535, 26.372, 26.791, 26.372, 26.791, 27.628, 28.047, 28.465,
        28.884, 29.302, 29.721, 30.14, 30.558, 30.977
    },
    new double[]
    {
        26.791, 27.209, 27.628, 27.628, 27.628, 28.047, 28.047, 28.047, 27.628, 27.628, 27.209,
        26.791, 26.372, 25.953, 25.535, 25.116, 25.116, 25.953, 26.791, 26.791, 27.628, 28.465,
        28.884, 29.721, 30.558, 31.395, 32.233, 33.07
    },
    new double[]
    {
        25.116, 25.953, 26.372, 26.791, 27.209, 27.209, 27.628, 27.628, 27.628, 27.628, 27.209,
        27.209, 26.791, 25.953, 25.535, 25.116, 25.116, 25.953, 26.791, 27.209, 27.628, 28.465,
        28.884, 29.721, 30.14, 30.558, 30.977, 31.395
    },
    new double[]
    {
        24.279, 25.116, 25.953, 26.791, 27.209, 27.628, 28.047, 28.465, 28.465, 28.465, 28.047,
        27.628, 27.209, 27.209, 26.791, 26.372, 25.953, 26.372, 26.791, 27.209, 27.628, 28.047,
        28.465, 28.884, 29.302, 29.721, 30.14, 30.558
    }
};

/**
 * Array de calores específicos de acero en función del código BISRA y la temperatura
 */
static double[][] Cp = new double[][]
{
    new double[]
    {
        485.6, 485.6, 494.0, 510.7, 527.4, 544.2, 560.9, 581.9, 611.2, 644.7, 682.3, 728.4, 778.6,
        833.0, 983.7, 987.9, 837.2, 820.5, 749.3, 665.6, 669.8, 669.8, 669.8, 669.8, 669.8, 669.8,
        669.8, 669.8
    },
    new double[]
    {
```

```
    485.6, 485.6, 494.0, 510.7, 531.6, 548.4, 560.9, 581.9, 611.2, 644.7, 678.1, 720.0, 766.0,
    820.5, 996.3, 1050.7, 912.6, 837.2, 732.6, 653.0, 657.2, 661.4, 661.4, 661.4, 661.4, 665.6,
    669.8, 669.8
},
new double[]
{
    485.6, 485.6, 494.0, 510.7, 527.4, 544.2, 565.1, 586.0, 611.2, 644.7, 682.3, 724.2, 766.0,
    816.3, 1138.6, 1193.0, 845.6, 690.7, 648.8, 648.8, 648.8, 648.8, 653.0, 661.4, 669.8, 682.3,
    686.5, 686.5
},
new double[]
{
    485.6, 485.6, 494.0, 506.5, 519.1, 540.0, 560.9, 577.7, 598.6, 632.1, 669.8, 699.1, 720.0,
    749.3, 1176.3, 1100.9, 565.1, 527.4, 586.0, 623.7, 627.9, 632.1, 636.3, 648.8, 661.4, 678.1,
    686.5, 686.5
},
new double[]
{
    477.2, 477.2, 494.0, 514.9, 527.4, 544.2, 565.1, 586.0, 602.8, 627.9, 657.2, 694.9, 740.9,
    807.9, 1343.7, 1347.9, 736.7, 598.6, 607.0, 627.9, 627.9, 627.9, 636.3, 648.8, 661.4, 674.0,
    678.1, 678.1
},
new double[]
{
    485.6, 485.6, 502.3, 527.4, 540.0, 556.7, 577.7, 594.4, 615.3, 648.8, 682.3, 703.3, 720.0,
    749.3, 1423.3, 1343.7, 636.3, 640.5, 619.5, 627.9, 636.3, 644.7, 657.2, 665.6, 674.0, 678.1,
    678.1, 678.1
},
new double[]
{
    502.3, 502.3, 510.7, 531.6, 544.2, 556.7, 573.5, 594.4, 619.5, 648.8, 686.5, 728.4, 766.0,
    807.9, 1511.2, 1406.5, 623.7, 619.5, 619.5, 615.3, 619.5, 632.1, 644.7, 661.4, 674.0, 686.5,
    694.9, 694.9
},
new double[]
{
    485.6, 485.6, 502.3, 531.6, 544.2, 552.6, 569.3, 586.0, 607.0, 627.9, 648.8, 678.1, 720.0,
    782.8, 1452.6, 1368.8, 653.0, 636.3, 623.7, 623.7, 632.1, 640.5, 644.7, 653.0, 661.4, 669.8,
    669.8, 669.8
},
new double[]
{
    477.2, 477.2, 485.6, 502.3, 519.1, 535.8, 552.6, 577.7, 602.8, 632.1, 674.0, 720.0, 757.7,
    807.1, 1147.0, 1135.3, 686.5, 544.2, 560.9, 594.4, 602.8, 611.2, 619.5, 627.9, 636.3, 640.5,
    644.7, 644.7
},
new double[]
{
    485.6, 485.6, 494.0, 510.7, 527.4, 544.2, 560.9, 577.7, 602.8, 640.5, 682.3, 724.2, 770.2,
    1214.0, 1293.5, 778.6, 615.3, 636.3, 644.7, 644.7, 644.7, 644.7, 640.5, 644.7, 653.0, 657.2,
    661.4, 661.4
},
new double[]
{
```

```
    494.0, 494.0, 502.3, 514.9, 527.4, 552.6, 573.5, 586.0, 615.3, 653.0, 694.9, 749.3, 795.3,
    1059.1, 1239.1, 866.5, 565.1, 573.5, 607.0, 636.3, 636.3, 636.3, 644.7, 653.0, 653.0, 653.0,
    653.0, 653.0
},
new double[]
{
    485.6, 485.6, 502.3, 519.1, 527.4, 544.2, 560.9, 581.9, 611.2, 644.7, 686.5, 736.7, 782.8,
    975.3, 1264.2, 975.3, 569.3, 573.5, 598.6, 627.9, 644.7, 648.8, 653.0, 657.2, 661.4, 665.6,
    661.4, 661.4
},
new double[]
{
    485.6, 485.6, 494.0, 514.9, 535.8, 548.4, 569.3, 594.4, 619.5, 653.0, 694.9, 745.1, 795.3,
    937.7, 1356.3, 1147.0, 636.3, 636.3, 640.5, 636.3, 632.1, 636.3, 636.3, 640.5, 644.7, 648.8,
    653.0, 653.0
},
new double[]
{
    494.0, 494.0, 502.3, 514.9, 527.4, 544.2, 565.1, 586.0, 607.0, 640.5, 678.1, 715.8, 757.7,
    807.9, 1167.9, 1218.1, 749.3, 569.3, 594.4, 619.5, 627.9, 627.9, 619.5, 619.5, 627.9, 636.3,
    644.7, 644.7
},
new double[]
{
    477.2, 477.2, 485.6, 506.5, 523.3, 535.8, 556.7, 581.9, 602.8, 632.1, 674.0, 715.8, 753.5,
    799.5, 1218.1, 1247.4, 728.4, 577.7, 590.2, 598.6, 607.0, 611.2, 619.5, 632.1, 636.3, 640.5,
    644.7, 644.7
},
new double[]
{
    502.3, 502.3, 502.3, 514.9, 535.8, 548.4, 565.1, 590.2, 615.3, 648.8, 686.5, 728.4, 766.0,
    803.7, 866.5, 1134.4, 987.9, 619.5, 627.9, 632.1, 640.5, 648.8, 657.2, 665.6, 674.0, 682.3,
    686.5, 686.5
},
new double[]
{
    519.1, 519.1, 527.4, 552.6, 573.5, 586.0, 602.8, 607.0, 611.2, 615.3, 653.0, 699.1, 674.0,
    644.7, 644.7, 648.8, 653.0, 661.4, 665.6, 669.8, 678.1, 682.3, 690.7, 694.9, 699.1, 703.3,
    703.3, 703.3
},
new double[]
{
    502.3, 502.3, 506.5, 514.9, 527.4, 540.0, 548.4, 544.2, 540.0, 556.7, 573.5, 581.9, 590.2,
    590.2, 586.0, 586.0, 586.0, 590.2, 594.4, 594.4, 598.6, 602.8, 607.0, 615.3, 632.1, 653.0,
    661.4, 661.4
},
new double[]
{
    510.7, 510.7, 519.1, 531.6, 535.8, 544.2, 552.6, 560.9, 577.7, 590.2, 611.2, 640.5, 640.5,
    627.9, 623.7, 632.1, 644.7, 644.7, 644.7, 648.8, 653.0, 657.2, 665.6, 669.8, 674.0, 678.1,
    678.1, 678.1
},
new double[]
{
```

```
        468.8, 468.8, 485.6, 510.7, 523.3, 540.0, 565.1, 590.2, 619.5, 661.4, 703.3, 753.5, 799.5,
        845.6, 891.6, 795.3, 745.1, 736.7, 661.4, 648.8, 648.8, 653.0, 653.0, 648.8, 648.8, 653.0,
        653.0, 653.0
    },
    new double[]
    {
        468.8, 468.8, 481.4, 502.3, 523.3, 540.0, 556.7, 581.9, 615.3, 657.2, 703.3, 749.3, 807.9,
        887.4, 958.6, 887.4, 870.7, 807.9, 653.0, 648.8, 644.7, 644.7, 648.8, 653.0, 661.4, 669.8,
        678.1, 678.1
    },
    new double[]
    {
        410.2, 410.2, 418.6, 431.2, 443.7, 460.5, 477.2, 494.0, 510.7, 535.8, 569.3, 590.2, 602.8,
        623.7, 678.1, 720.0, 699.1, 707.4, 657.2, 586.0, 602.8, 615.3, 611.2, 611.2, 611.2, 607.0,
        611.2, 611.2
    }
};

/**
 * Array de densidades del acero en función del código BISRA y la temperatura
 */
static double[][] Ro = new double[][]
{
    new double[]
    {
        7876.0, 7861.0, 7846.0, 7830.0, 7814.0, 7798.0, 7781.0, 7763.0, 7745.0, 7727.0, 7708.0,
        7688.0, 7668.0, 7648.0, 7628.0, 7610.0, 7598.0, 7601.0, 7602.0, 7580.0, 7550.0, 7523.0,
        7495.0, 7467.0, 7439.0, 7411.0, 7383.0, 7355.0
    },
    new double[]
    {
        7861.0, 7847.0, 7832.0, 7816.0, 7800.0, 7783.0, 7765.0, 7748.0, 7730.0, 7711.0, 7692.0,
        7673.0, 7653.0, 7632.0, 7613.0, 7594.0, 7582.0, 7589.0, 7594.0, 7572.0, 7543.0, 7515.0,
        7488.0, 7461.0, 7434.0, 7407.0, 7380.0, 7353.0
    },
    new double[]
    {
        7863.0, 7849.0, 7834.0, 7819.0, 7803.0, 7787.0, 7770.0, 7753.0, 7736.0, 7718.0, 7699.0,
        7679.0, 7659.0, 7635.0, 7617.0, 7620.0, 7624.0, 7616.0, 7600.0, 7574.0, 7548.0, 7522.0,
        7496.0, 7470.0, 7444.0, 7418.0, 7392.0, 7366.0
    },
    new double[]
    {
        7858.0, 7845.0, 7832.0, 7817.0, 7801.0, 7784.0, 7766.0, 7748.0, 7730.0, 7711.0, 7692.0,
        7672.0, 7652.0, 7628.0, 7613.0, 7624.0, 7635.0, 7617.0, 7590.0, 7564.0, 7538.0, 7512.0,
        7486.0, 7460.0, 7434.0, 7408.0, 7382.0, 7356.0
    },
    new double[]
    {
        7848.0, 7835.0, 7821.0, 7806.0, 7790.0, 7773.0, 7755.0, 7737.0, 7719.0, 7700.0, 7681.0,
        7661.0, 7641.0, 7617.0, 7599.0, 7607.0, 7613.0, 7589.0, 7561.0, 7534.0, 7508.0, 7481.0,
        7455.0, 7429.0, 7403.0, 7377.0, 7351.0, 7325.0
    },
    new double[]
    {
```

```
    7855.0, 7842.0, 7829.0, 7815.0, 7800.0, 7784.0, 7767.0, 7749.0, 7731.0, 7713.0, 7694.0,
    7675.0, 7655.0, 7632.0, 7612.0, 7604.0, 7594.0, 7565.0, 7533.0, 7509.0, 7485.0, 7460.0,
    7436.0, 7412.0, 7388.0, 7364.0, 7340.0, 7316.0
},
new double[]
{
    7855.0, 7842.0, 7829.0, 7815.0, 7800.0, 7784.0, 7767.0, 7749.0, 7731.0, 7713.0, 7694.0,
    7675.0, 7655.0, 7632.0, 7612.0, 7604.0, 7594.0, 7565.0, 7533.0, 7509.0, 7485.0, 7460.0,
    7436.0, 7412.0, 7388.0, 7364.0, 7340.0, 7316.0
},
new double[]
{
    7834.0, 7822.0, 7809.0, 7796.0, 7781.0, 7765.0, 7749.0, 7731.0, 7713.0, 7694.0, 7675.0,
    7655.0, 7634.0, 7613.0, 7592.0, 7581.0, 7565.0, 7528.0, 7489.0, 7463.0, 7438.0, 7413.0,
    7388.0, 7363.0, 7338.0, 7313.0, 7288.0, 7263.0
},
new double[]
{
    7854.0, 7840.0, 7826.0, 7811.0, 7794.0, 7777.0, 7760.0, 7742.0, 7723.0, 7704.0, 7685.0,
    7666.0, 7646.0, 7622.0, 7605.0, 7615.0, 7626.0, 7614.0, 7590.0, 7561.0, 7532.0, 7503.0,
    7474.0, 7445.0, 7416.0, 7387.0, 7358.0, 7329.0
},
new double[]
{
    7859.0, 7846.0, 7833.0, 7818.0, 7803.0, 7788.0, 7772.0, 7755.0, 7737.0, 7719.0, 7700.0,
    7680.0, 7663.0, 7650.0, 7643.0, 7648.0, 7650.0, 7626.0, 7598.0, 7572.0, 7546.0, 7520.0,
    7494.0, 7468.0, 7442.0, 7416.0, 7390.0, 7364.0
},
new double[]
{
    7851.0, 7838.0, 7824.0, 7809.0, 7793.0, 7777.0, 7760.0, 7744.0, 7727.0, 7710.0, 7693.0,
    7675.0, 7657.0, 7637.0, 7625.0, 7638.0, 7650.0, 7627.0, 7596.0, 7569.0, 8542.0, 7516.0,
    7490.0, 7464.0, 7438.0, 7412.0, 7386.0, 7360.0
},
new double[]
{
    7852.0, 7839.0, 7825.0, 7810.0, 7795.0, 7779.0, 7762.0, 7745.0, 7728.0, 7710.0, 7692.0,
    7673.0, 7655.0, 7637.0, 7626.0, 7638.0, 7647.0, 7624.0, 7594.0, 7568.0, 7542.0, 7515.0,
    7489.0, 7463.0, 7437.0, 7411.0, 7385.0, 7359.0
},
new double[]
{
    7863.0, 7850.0, 7836.0, 7821.0, 7805.0, 7790.0, 7774.0, 7757.0, 7739.0, 7722.0, 7704.0,
    7685.0, 7668.0, 7651.0, 7641.0, 7653.0, 7662.0, 7638.0, 7608.0, 7583.0, 7557.0, 7533.0,
    7510.0, 7487.0, 7464.0, 7441.0, 7418.0, 7395.0
},
new double[]
{
    7847.0, 7833.0, 7818.0, 7803.0, 7787.0, 7770.0, 7753.0, 7736.0, 7718.0, 7700.0, 7681.0,
    7662.0, 7643.0, 7619.0, 7603.0, 7611.0, 7619.0, 7604.0, 7579.0, 7552.0, 7525.0, 7499.0,
    7473.0, 7447.0, 7421.0, 7395.0, 7369.0, 7343.0
},
new double[]
{
```

```
            7850.0, 7835.0, 7820.0, 7804.0, 7788.0, 7772.0, 7755.0, 7738.0, 7720.0, 7702.0, 7683.0,
            7664.0, 7644.0, 7621.0, 7606.0, 7616.0, 7625.0, 7602.0, 7574.0, 7549.0, 7524.0, 7499.0,
            7475.0, 7451.0, 7427.0, 7403.0, 7379.0, 7355.0
        },
        new double[]
        {
            7729.0, 7717.0, 7703.0, 7688.0, 7672.0, 7656.0, 7639.0, 7622.0, 7605.0, 7588.0, 7570.0,
            7552.0, 7533.0, 7513.0, 7495.0, 7486.0, 7477.0, 7464.0, 7442.0, 7418.0, 7392.0, 7365.0,
            7339.0, 7313.0, 7287.0, 7261.0, 7235.0, 7209.0
        },
        new double[]
        {
            7877.0, 7857.0, 7834.0, 7811.0, 7785.0, 7758.0, 7730.0, 7700.0, 7672.0, 7650.0, 7632.0,
            7615.0, 7595.0, 7569.0, 7538.0, 7503.0, 7464.0, 7419.0, 7378.0, 7353.0, 7330.0, 7304.0,
            7263.0, 7222.0, 7181.0, 7140.0, 7099.0, 7058.0
        },
        new double[]
        {
            8166.0, 8150.0, 8132.0, 8112.0, 8091.0, 8069.0, 8046.0, 8023.0, 7999.0, 7976.0, 7952.0,
            7928.0, 7904.0, 7880.0, 7856.0, 7832.0, 7808.0, 7783.0, 7757.0, 7731.0, 7705.0, 7678.0,
            7651.0, 7624.0, 7597.0, 7570.0, 7543.0, 7516.0
        },
        new double[]
        {
            7921.0, 7905.0, 7886.0, 7865.0, 7843.0, 7821.0, 7799.0, 7777.0, 7754.0, 7730.0, 7707.0,
            7683.0, 7658.0, 7634.0, 7609.0, 7584.0, 7559.0, 7535.0, 7510.0, 7486.0, 7461.0, 7436.0,
            7412.0, 7388.0, 7364.0, 7340.0, 7316.0, 7292.0
        },
        new double[]
        {
            7749.0, 7738.0, 7725.0, 7713.0, 7700.0, 7686.0, 7671.0, 7657.0, 7642.0, 7627.0, 7611.0,
            7595.0, 7580.0, 7564.0, 7548.0, 7529.0, 7515.0, 7520.0, 7523.0, 7504.0, 7477.0, 7451.0,
            7425.0, 7399.0, 7373.0, 7347.0, 7321.0, 7295.0
        },
        new double[]
        {
            7745.0, 7734.0, 7722.0, 7709.0, 7696.0, 7682.0, 7667.0, 7653.0, 7638.0, 7623.0, 7608.0,
            7591.0, 7575.0, 7558.0, 7542.0, 7528.0, 7519.0, 7524.0, 7524.0, 7497.0, 7463.0, 7430.0,
            7397.0, 7364.0, 7331.0, 7298.0, 7265.0, 7232.0
        },
        new double[]
        {
            8696.0, 8682.0, 8667.0, 8651.0, 8635.0, 8619.0, 8602.0, 8586.0, 8569.0, 8552.0, 8534.0,
            8516.0, 8498.0, 8481.0, 8462.0, 8441.0, 8425.0, 8429.0, 8430.0, 8404.0, 8372.0, 8342.0,
            8313.0, 8284.0, 8255.0, 8226.0, 8197.0, 8168.0
        }
    };

    /**
     * Array de temperaturas
     */
    static double[] T = new double[]
    {
```

```
    273.0, 323.0, 373.0, 423.0, 473.0, 523.0, 573.0, 623.0, 673.0, 723.0, 773.0, 823.0, 873.0, 923.0,
        973.0, 1023.0, 1073.0, 1123.0, 1173.0, 1223.0, 1273.0, 1323.0, 1373.0, 1423.0, 1473.0,
        1523.0, 1573.0, 1623.0
};

/**
 * Método que da la conductividad térmica del acero para una temperatura y código BISRA
 *
 * @param Temperatura Temperatura
 * @param codigoBISRA Código BISRA
 * @return Conductividad térmica
 */
public static Double getConductividad(double Temperatura, int codigoBISRA)
{
    if (Temperatura <= T[0])
    {
        return K[codigoBISRA][0];
    }
    else if (Temperatura >= T[T.length - 1])
    {
        return K[codigoBISRA][T.length - 1];
    }
    else
    {
        for (int i = 0; i < T.length - 1; i++)
        {
            if (Temperatura > T[i] && Temperatura <= T[i + 1])
            {
                return (Temperatura - T[i]) * (K[codigoBISRA][i + 1] - K[codigoBISRA][i]) / (T[i + 1]
        - T[i]) + K[codigoBISRA][i];
            }
        }
    }
    return null;
}

/**
 * Método que da el calor específico del acero para una temperatura y código BISRA
 *
 * @param Temperatura Temperatura
 * @param codigoBISRA Código BISRA
 * @return Calor específico
 */
public static Double getCalorEspecifico(double Temperatura, int codigoBISRA)
{
    if (Temperatura <= T[0])
    {
        return Cp[codigoBISRA][0];
    }
    else if (Temperatura >= T[T.length - 1])
    {
        return Cp[codigoBISRA][T.length - 1];
    }
    else
    {
```

```
        for (int i = 0; i < T.length - 1; i++)
        {
            if (Temperatura > T[i] && Temperatura <= T[i + 1])
            {
                return (Temperatura - T[i]) * (Cp[codigoBISRA][i + 1] - Cp[codigoBISRA][i]) / (T[i +
         1] - T[i]) + Cp[codigoBISRA][i];
            }
        }
    }
    return null;
}

/**
 * Método que da la densidad del acero para una temperatura y código BISRA
 *
 * @param Temperatura Temperatura
 * @param codigoBISRA Código BISRA
 * @return Densidad
 */
public static Double getDensidad(double Temperatura, int codigoBISRA)
{
    if (Temperatura <= T[0])
    {
        return Ro[codigoBISRA][0];
    }
    else if (Temperatura >= T[T.length - 1])
    {
        return Ro[codigoBISRA][T.length - 1];
    }
    else
    {
        for (int i = 0; i < T.length - 1; i++)
        {
            if (Temperatura > T[i] && Temperatura <= T[i + 1])
            {
                return (Temperatura - T[i]) * (Ro[codigoBISRA][i + 1] - Ro[codigoBISRA][i]) / (T[i +
         1] - T[i]) + Ro[codigoBISRA][i];
            }
        }
    }
    return null;
}
```

1.5.2 Clase EtiquetaColor

```
package hot;

import javafx.scene.control.Label;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;

class EtiquetaColor extends Label
{
   public EtiquetaColor(Color c)
   {
      getStyleClass().clear();
      setTextFill(c);
      setFont(Font.font(getFont().getName(),14));
   }

   public void setSizeFont(double size)
   {
      setFont(Font.font(getFont().getName(),size));
   }

   public EtiquetaColor(String text, Color c)
   {
      super(text);
      getStyleClass().clear();
      setTextFill(c);
      setFont(Font.font(getFont().getName(),14));
   }
}
```

1.5.3 Clase HOT

```
package hot;

import Jama.Matrix;
import com.DatoLaminacion;
import static com.DatoLaminacion.ObtieneVarPalanquilla;
import com.Global;
import com.estado.Registro;
import com.estado.TrazaLaminacion;
import com.interfaz.Esqueleto;
import com.interfaz.MessageDialog;
import com.interfaz.ModalDialog;
import com.interfaz.laminacion.Laminacion;
import com.opc.ExcepcionOPC;
import com.prosysopc.ua.SecureIdentityException;
import com.prosysopc.ua.ServiceException;
import com.prosysopc.ua.StatusException;
import static hot.PalanquillaHOT.getCodigoBisra;
import static hot.Principal.scene;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.URISyntaxException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Platform;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.effect.ColorAdjust;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundFill;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.Priority;
```

```
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Paint;
import javafx.scene.paint.Stop;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.util.Duration;
import org.opcfoundation.ua.builtintypes.DataValue;
import org.opcfoundation.ua.common.ServiceResultException;

/**
 *
 * @author SONIA
 */
public class HOT
{

    private ImageView img;
    Stop[] stops = new Stop[]
    {
       //new Stop(0, Color.web("#F8F8FF", 0.5)/*.brighter().brighter()*/), new Stop(0.5,
           Color.web("#DCDCDC", 0.5)/*.brighter().brighter()*/)
       new Stop(0, Color.web("#FFFFFF", 0.8)/*.brighter().brighter()*/), new Stop(0.4,
           Color.web("#FFE4B5", 1)/*.brighter().brighter()*/)
    //new Stop(0, Color.web("#F0FFFF", 0.7)/*.brighter().brighter()*/), new Stop(0.7,
           Color.web("#00008B", 0.1)/*.brighter().brighter()*/)
    };
    LinearGradient linearGrad = new LinearGradient(0, 0, 0, 1, true, CycleMethod.NO_CYCLE,
           stops);

    public Esqueleto esqueleto;
    private Laminacion laminacion;

    java.awt.Color[] coloresFondo;
    private int x;
    private Text labelContador;
    private int cuenta = -1;
    static Group root;

    List<PalanquillaHOT> listaTC = new ArrayList<>();

    static double caudalAireCalentamientoSuperior;
    static double temperaturaBoveda1CalentamientoSuperior;

    VistasHorno prueba;
    Timeline temporizador;

    ZonalRadiacion zonaSuperior = new ZonalRadiacion("up",
          new double[]
          {
             0, 4.96, 4.96 + 4.8125
          },
```

```
        new double[]
        {
            1, 1.4, 1.4
        },
        new double[]
        {
            3.345, 9.675, 13.318
        },
        14.56, 7.9, 0.1, 0.26);

ZonalRadiacion zonaInferior = new ZonalRadiacion("down",
        new double[]
        {
            0, 4.96, 4.96 + 4.8125
        },
        new double[]
        {
            1.820, 1.820, 1.820
        },
        new double[]
        {
            3.4, 8.68, 12.56
        },
        14.56, 7.9, 0.1, 0.26);
static public Pane pane;

javafx.scene.paint.Color colorAWT2Scene(java.awt.Color awtColor)
{
    int r = awtColor.getRed();
    int g = awtColor.getGreen();
    int b = awtColor.getBlue();
    int a = awtColor.getAlpha();
    double opacity = a / 255.0;
    return javafx.scene.paint.Color.rgb(r, g, b, opacity);
}

void setSizeFuente(double size, ObservableList<Node> lista)
{
    for (Node nodo : lista)
    {
        if (nodo instanceof EtiquetaColor)
        {
            EtiquetaColor etiqueta = (EtiquetaColor) nodo;
            etiqueta.setSizeFont(size);
        }

        if (nodo instanceof GridPane)
        {
            GridPane grid = (GridPane) nodo;
            grid.setVgap(size);
        }

        if (nodo instanceof VBoxCuadro)
        {
            VBoxCuadro v = (VBoxCuadro) nodo;
```

```
            v.setSpacing(size);
        }

        if (nodo instanceof Parent)
        {
            Parent padre = (Parent) nodo;
            if (padre.getChildrenUnmodifiable().size() > 0)
            {
                setSizeFuente(size, padre.getChildrenUnmodifiable());
            }
        }

    }
}

double formatea(double valor, int numeroDecimales)
{
    //return Double.valueOf(String.format("%."+String.valueOf(numeroDecimales), valor));
    int factor = 1;
    for (int i = 0; i < numeroDecimales; i++)
    {
        factor *= 10;
    }
    return ((int) (valor * factor)) / (double) factor;
}

public void ImprimeRegistro(String palanquilla, double[] datos)
{
    LocalDateTime actual = LocalDateTime.now();
    PrintWriter pw;

    String nombreDir = actual.format(DateTimeFormatter.ofPattern("yyyyMMdd"));
    File directorio = new File(nombreDir);
    directorio.mkdirs();

    try
    {
        pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(new
         FileOutputStream(
                new File(esqueleto.dirTrabajo
                        + nombreDir
                        + System.getProperty("file.separator")
                        + palanquilla
                        + ".dat"), true), "ISO8859_1")));
        String strRegistro = actual.format(DateTimeFormatter.ofPattern("dd/MM/yyyy
         HH:mm:ss")) + ";";
        for (double dato : datos)
        {
            strRegistro += String.format("%.1f", dato) + ";";
        }

        pw.println(strRegistro);
        pw.close();
    }
    catch (IOException ex)
```

```
    {
        Global.info.Registra(ex);
    }
}

Pane getVentana()
{
    pane=new Pane(root);
    VBox.setVgrow(pane, Priority.ALWAYS);
    //pane.setStyle("-fx-background-color:white");
    pane.setBackground(
            new Background(
                    new BackgroundFill(
                            linearGrad, null, new Insets(0)
                    )
            )
    );
    return pane;
}

public void update(double width, double height)
{
    prueba = new VistasHorno(width, height);
    root.getChildren().addAll(prueba.getHorno());

    root.getStyleClass().add("fondo");
    /*Button btnOrdena = new Button("No pulsar");
    Button btnDialogo = new Button("Cuadro de diálogo");

    btnOrdena.setLayoutX(300);
    btnOrdena.setLayoutY(100);
    labelContador = new Text();
    labelContador.setFont(Font.font("Verdana", FontWeight.BOLD, 70));
    labelContador.setFill(Color.RED);

    labelContador.setLayoutX(400);
    labelContador.setLayoutY(900);

    btnDialogo.setLayoutX(150);
    btnDialogo.setLayoutY(900);

    root.getChildren().addAll(btnOrdena, labelContador, btnDialogo);
    btnOrdena.setOnAction(new EventHandler<ActionEvent>()
    {

        @Override
        public void handle(ActionEvent event)
        {
            cuenta = 0;
            Global.info.Error("¡Eh! Alguien ha pulsado el botón que no debía");
        }
    });

    btnDialogo.setOnAction(new EventHandler<ActionEvent>()
    {
```

```
    @Override
    public void handle(ActionEvent event)
    {
        VBox vbox = new VBox();
        HBox p = new HBox(10);
            //img= new ImageView(new File(".\\Niño_Puñeta.gif").toURI().toString());
        //img= new ImageView(new
     File("src\\Images\\Niño_Puñeta.gif").toURI().toString());
        Image image = new
     Image(getClass().getResourceAsStream("/Images/Niño_Puñeta.gif"));
        img = new ImageView(image);

        p.setAlignment(Pos.CENTER);
        TextField tf;
        p.getChildren().addAll(new Label(""));
        vbox.getChildren().addAll(p, img);

        ColorAdjust colorAdjust = new ColorAdjust();
        colorAdjust.setContrast(0);
        colorAdjust.setHue(0);
        colorAdjust.setBrightness(0.3);
        colorAdjust.setSaturation(-0.75);

        root.setEffect(colorAdjust);
        Paint paint = scene.getFill();
        Background bg=HOT.pane.getBackground();
        HOT.pane.setBackground(null);

        ModalDialog dialogo = new ModalDialog(esqueleto, vbox, true);
        dialogo.boxButtons.getChildren().remove(dialogo.btnCancelar);
        if (dialogo.ShowModal())
        {
            root.setEffect(null);
            HOT.pane.setBackground(bg);
        }
        else
        {
            //Pulso cancela
        }
    }
});*/

zonaSuperior.Carga();
zonaInferior.Carga();

    //Transferencia_Calor Palanquilla = new Transferencia_Calor(19, col, fil, dim, hc,
     emisividad);
//TEMPORIZADOR
x = 10000;

temporizador = new Timeline(new KeyFrame(Duration.ZERO, (ActionEvent actionEvent)
     ->
{
    if (cuenta >= 0)
```

```
{
   labelContador.setText("Cuenta atrás " + String.valueOf(5 - (cuenta)));
   if (cuenta == 5)
   {

      Platform.runLater(new Runnable()
      {
         @Override
         public void run()
         {
            cuenta = -1;
            new MessageDialog(esqueleto, "¡Estás despedido! Te dije que no pulsaras",
MessageDialog.TipoMensaje.ERROR).Show();
            labelContador.setText("");
         }
      });

   }
   cuenta++;
}

try
{
   DataValue[] valores = laminacion.opc.leerGrupoOPC(0);

   caudalAireCalentamientoSuperior =
Double.parseDouble(laminacion.getValor(valores, "Caudal Aire Calentamiento
Superior"));
   temperaturaBoveda1CalentamientoSuperior =
Double.parseDouble(laminacion.getValor(valores, "Temperatura Boveda 1
Calentamiento Superior"));

   prueba.temperaturaChimenea.setText(laminacion.getValor(valores, "Temperatura
Salida Horno") + " ºC");

prueba.temperaturaTermoparPrecalentamientoSuperior.setText(laminacion.getValor(val
ores, "Temperatura Recuperación Superior") + " ºC");

prueba.temperaturaTermoparPrecalentamientoInferior.setText(laminacion.getValor(val
ores, "Temperatura Recuperación Inferior") + " ºC");


prueba.zonasHornoCalentamientoSuperior.consigna.setText(laminacion.getValor(valor
es, "Temperatura Consigna Calentamiento Superior"));

prueba.zonasHornoCalentamientoSuperior.medida.setText(laminacion.getValor(valores
, "Temperatura Medida Calentamiento Superior"));

prueba.zonasHornoCalentamientoSuperior.regul_salida.setText(laminacion.getValor(va
lores, "Salida Regulador Calentamiento Superior"));

prueba.zonasHornoCalentamientoSuperior.TC_regulacion.setText(laminacion.getValor(
valores, "Selección PARED/BOVEDA regulación Calentamiento Superior"));
```

```
prueba.zonasHornoCalentamientoSuperior.Temp_pared1.setText(laminacion.getValor(
valores, "Temperatura Pared 1 Calentamiento Superior"));

prueba.zonasHornoCalentamientoSuperior.Temp_pared2.setText(laminacion.getValor(
valores, "Temperatura Pared 2 Calentamiento Superior"));

prueba.zonasHornoCalentamientoSuperior.Temp_boveda1.setText(laminacion.getValor
(valores, "Temperatura Boveda 1 Calentamiento Superior"));

prueba.zonasHornoCalentamientoSuperior.Temp_boveda2.setText(laminacion.getValor
(valores, "Temperatura Boveda 2 Calentamiento Superior"));

prueba.zonasHornoCalentamientoSuperior.caudalGas.setText(laminacion.getValor(valo
res, "Caudal Gas Calentamiento Superior"));


prueba.zonasHornoCalentamientoInferior.consigna.setText(laminacion.getValor(valore
s, "Temperatura Consigna Calentamiento Inferior"));

prueba.zonasHornoCalentamientoInferior.medida.setText(laminacion.getValor(valores,
"Temperatura Medida Calentamiento Inferior"));

prueba.zonasHornoCalentamientoInferior.regul_salida.setText(laminacion.getValor(val
ores, "Salida Regulador Calentamiento Inferior"));

prueba.zonasHornoCalentamientoInferior.TC_regulacion.setText(laminacion.getValor(
valores, "Selección PARED/BOVEDA regulación Calentamiento Inferior"));

prueba.zonasHornoCalentamientoInferior.Temp_pared1.setText(laminacion.getValor(v
alores, "Temperatura Pared 1 Calentamiento Inferior"));

prueba.zonasHornoCalentamientoInferior.Temp_pared2.setText(laminacion.getValor(v
alores, "Temperatura Pared 2 Calentamiento Inferior"));

prueba.zonasHornoCalentamientoInferior.Temp_boveda1.setText(laminacion.getValor(
valores, "Temperatura Boveda 1 Calentamiento Inferior"));

prueba.zonasHornoCalentamientoInferior.Temp_boveda2.setText(laminacion.getValor(
valores, "Temperatura Boveda 2 Calentamiento Inferior"));

prueba.zonasHornoCalentamientoInferior.caudalGas.setText(laminacion.getValor(valor
es, "Caudal Gas Calentamiento Inferior"));
  prueba.zonasHornoCalentamientoInferior.cteA.setText(laminacion.getValor(valores,
"Cte. caudal A Calentamiento Inferior"));
  prueba.zonasHornoCalentamientoInferior.cteB.setText(laminacion.getValor(valores,
"Cte. caudal B Calentamiento Inferior"));

  prueba.zonasHornoIgualacionSuperior.consigna.setText(laminacion.getValor(valores,
"Temperatura Consigna Igualación Superior"));
  prueba.zonasHornoIgualacionSuperior.medida.setText(laminacion.getValor(valores,
"Temperatura Medida Igualación Superior"));

prueba.zonasHornoIgualacionSuperior.regul_salida.setText(laminacion.getValor(valore
s, "Salida Regulador Igualación Superior"));
```

```
prueba.zonasHornoIgualacionSuperior.TC_regulacion.setText(laminacion.getValor(val
ores, "Selección PARED/BOVEDA regulación Igualación Superior"));

prueba.zonasHornoIgualacionSuperior.Temp_pared1.setText(laminacion.getValor(valor
es, "Temperatura Pared 1 Igualación Superior"));

prueba.zonasHornoIgualacionSuperior.Temp_pared2.setText(laminacion.getValor(valor
es, "Temperatura Pared 2 Igualación Superior"));

prueba.zonasHornoIgualacionSuperior.Temp_boveda1.setText(laminacion.getValor(val
ores, "Temperatura Boveda 1 Igualación Superior"));

prueba.zonasHornoIgualacionSuperior.Temp_boveda2.setText(laminacion.getValor(val
ores, "Temperatura Boveda 2 Igualación Superior"));

prueba.zonasHornoIgualacionSuperior.caudalGas.setText(laminacion.getValor(valores,
"Caudal Gas Igualación Superior"));

  prueba.zonasHornoIgualacionInferior.consigna.setText(laminacion.getValor(valores,
"Temperatura Consigna Igualación Inferior"));
  prueba.zonasHornoIgualacionInferior.medida.setText(laminacion.getValor(valores,
"Temperatura Medida Igualación Inferior"));

prueba.zonasHornoIgualacionInferior.regul_salida.setText(laminacion.getValor(valores,
"Salida Regulador Igualación Inferior"));

prueba.zonasHornoIgualacionInferior.TC_regulacion.setText(laminacion.getValor(valo
res, "Selección PARED/BOVEDA regulación Igualación Inferior"));

prueba.zonasHornoIgualacionInferior.Temp_pared1.setText(laminacion.getValor(valor
es, "Temperatura Pared 1 Igualación Inferior"));

prueba.zonasHornoIgualacionInferior.Temp_pared2.setText(laminacion.getValor(valor
es, "Temperatura Pared 2 Igualación Inferior"));

prueba.zonasHornoIgualacionInferior.Temp_boveda1.setText(laminacion.getValor(valo
res, "Temperatura Boveda 1 Igualación Inferior"));

prueba.zonasHornoIgualacionInferior.Temp_boveda2.setText(laminacion.getValor(valo
res, "Temperatura Boveda 2 Igualación Inferior"));

prueba.zonasHornoIgualacionInferior.caudalGas.setText(laminacion.getValor(valores,
"Caudal Gas Igualación Inferior"));
  prueba.zonasHornoIgualacionInferior.cteA.setText(laminacion.getValor(valores,
"Cte. caudal A Igualación Inferior"));
  prueba.zonasHornoIgualacionInferior.cteB.setText(laminacion.getValor(valores, "Cte.
caudal B Igualación Inferior"));

  //Obtener las temperaturas de termopar de precalentameinto, calentamiento e
igualación, superior e inferior
  double TC1_Superior = Double.parseDouble(laminacion.getValor(valores,
"Temperatura Recuperación Superior"));
  double TC2_Superior = Double.parseDouble(laminacion.getValor(valores,
"Temperatura Medida Calentamiento Superior"));
```

```
  double TC3_Superior = Double.parseDouble(laminacion.getValor(valores,
"Temperatura Medida Igualación Superior"));
  double C3_Superior = Double.parseDouble(laminacion.getValor(valores, "Caudal
Aire Igualación Superior"));
  double C2_Superior = C3_Superior +
Double.parseDouble(laminacion.getValor(valores, "Caudal Aire Calentamiento
Superior"));
  double C1_Superior = C2_Superior;

  double TC1_Inferior = Double.parseDouble(laminacion.getValor(valores,
"Temperatura Recuperación Inferior"));
  double TC2_Inferior = Double.parseDouble(laminacion.getValor(valores,
"Temperatura Medida Calentamiento Inferior"));
  double TC3_Inferior = Double.parseDouble(laminacion.getValor(valores,
"Temperatura Medida Igualación Inferior"));
  double C3_Inferior = Double.parseDouble(laminacion.getValor(valores, "Caudal Aire
Igualación Inferior"));
  double C2_Inferior = C3_Inferior +
Double.parseDouble(laminacion.getValor(valores, "Caudal Aire Calentamiento
Inferior"));
  double C1_Inferior = C2_Inferior;

  //Calculamos las temperaturas de los gases
  double[] tempGases_Superior = zonaSuperior.TemperaturaGas(new double[]
  {
    TC1_Superior, TC2_Superior, TC3_Superior
  },
        new double[]
        {
          C1_Superior / 3600, C2_Superior / 3600, C3_Superior / 3600
        });

  double[] tempGases_Inferior = zonaSuperior.TemperaturaGas(new double[]
  {
    TC1_Inferior, TC2_Inferior, TC3_Inferior
  },
        new double[]
        {
          C1_Inferior / 3600, C2_Inferior / 3600, C3_Inferior / 3600
        });

  //Calculamos las potencias de radiacion superior e inferior en los volúmenes medidos
  double[] EgTc_Superior = zonaSuperior.getEgTc(tempGases_Superior);
  double[] EgTc_Inferior = zonaInferior.getEgTc(tempGases_Inferior);

  double[][] EgTc_Sup =
  {
    {
      EgTc_Superior[0]
    },
    {
      EgTc_Superior[1]
    },
    {
      EgTc_Superior[2]
```

```
        }
    };
    double[][] EgTc_Inf =
    {
        {
            EgTc_Inferior[0]
        },
        {
            EgTc_Inferior[1]
        },
        {
            EgTc_Inferior[2]
        }
    };

    Matrix EgTc_Up = new Matrix(EgTc_Sup);
    Matrix EgTc_Down = new Matrix(EgTc_Inf);

        //Calculamos las potencias de radiacion en todos los volúmenes y las radiosidades
en el techo y en el suelo
    //Eg.
    Matrix[] EgJ_Superior = zonaSuperior.getEg_J(EgTc_Superior);
    Matrix Eg_Superior = EgJ_Superior[0];
    Matrix J_Superior = EgJ_Superior[1];

    Matrix[] EgJ_Inferior = zonaInferior.getEg_J(EgTc_Inferior);
    Matrix Eg_Inferior = EgJ_Inferior[0];
    Matrix J_Inferior = EgJ_Inferior[1];

    Matrix I = Matrix.identity(EgTc_Superior.length, EgTc_Superior.length);

    //Comprobación superior
    Matrix resultado =
((zonaInferior.k4vtc.times(I).times(EgTc_Down)).minus(zonaInferior.gtcf.times(J_Infer
ior))).minus(zonaInferior.gtcg.times(Eg_Inferior));

    //Comprobación inferior
    List<Registro> listaHorno =
laminacion.horno.getRegistros(TrazaLaminacion.ordenTraza.PRIMERO_MAS_ANTI
GUO,
            new ArrayList<>(Arrays.asList("Horno")));

    boolean existe = false;

    //System.out.print("Abcisas:");
    for (int i = 0; i < listaHorno.size(); i++)
    {
        double abcisa = 0.0;
        double tcs=0.0;
        double tci=0.0;
        double tcc=0.0;
        Registro palanquillaTracking = listaHorno.get(i);
        try
        {
```

```
            abcisa =
Double.valueOf(ObtieneVarPalanquilla(DatoLaminacion.tipoVar.ABCISA,
palanquillaTracking.var));
            //System.out.print(String.valueOf(abcisa/1000.)+", ");
        }
        catch (NumberFormatException ex)
        {
            continue;
        }

        PalanquillaHOT palanquillaHOT = null;

        existe = false;

        for (int j = 0; j < listaTC.size(); j++)
        {
            if (listaHorno.get(i).ID.equals(listaTC.get(j).getID()))
            {
                existe = true;
                palanquillaHOT = listaTC.get(j);
                break;
            }
        }

        if (!existe)
        { //No existe así que la añado

            double longitud =
Integer.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.L
ONGITUD_PALANQUILLA, palanquillaTracking.getDat()));
            double a =
Integer.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.A
NCHO_PALANQUILLA, palanquillaTracking.getDat()));
            int tipo_acero =
Integer.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.TI
PO_ACERO, palanquillaTracking.getDat()));

            int indice =
(laminacion.getLote(laminacion.horno.getIDLote(palanquillaTracking.getID()), false) -
1) % 20;
            javafx.scene.paint.Color color;
            if (indice < 0)
            {
                color = Color.WHITE;
            }
            else
            {
                color = colorAWT2Scene(coloresFondo[indice]);
            }
            palanquillaHOT = new PalanquillaHOT(palanquillaTracking, color,
getCodigoBisra(tipo_acero));
            //PalanquillaHOT.setID(listaHorno.get(i).ID)==palanquillaTracking.getID();

            listaTC.add(palanquillaHOT);
```

```
            palanquillaHOT.setPathPlanta(prueba.getPalanquillaPlanta(a, longitud, color,
0));
            palanquillaHOT.setPathPerfil(prueba.getPalanquillaPerfil(a, color, 0));
            //palanquillaHOT.setAbcisa(abcisa, prueba.escalaLargo);
            root.getChildren().add(palanquillaHOT.getPathPerfil());
            root.getChildren().add(palanquillaHOT.getPathPlanta());
            palanquillaHOT.Inicializar();

        }
        //Obtenemos la temperatura ambiente aparente superior e inferior para esta abcisa

        double Ta_sup = zonaSuperior.TemperaturaAmbiente(abcisa / 1000, Eg_Superior,
J_Superior) + 273;
        double Ta_inf = zonaInferior.TemperaturaAmbiente(abcisa / 1000, Eg_Inferior,
J_Inferior) + 273;
        palanquillaHOT.Calcular_Proceso(new double[]
        {
           Ta_sup, (Ta_sup + Ta_inf) / 2, Ta_inf, (Ta_sup + Ta_inf) / 2
        });
        palanquillaHOT.setAbcisa(abcisa, prueba.kx * prueba.escalaLargo);
        tcs = palanquillaHOT.Valor_Temperatura(0.1, 0);
        tcc= palanquillaHOT.Valor_Temperatura(0.1, 0.1);
        tci = palanquillaHOT.Valor_Temperatura(0.1, 0.2);
        String var =
DatoLaminacion.ModificaVarPalanquilla(DatoLaminacion.tipoVar.TEMPERATURA_
SUPERIOR, palanquillaTracking.getVar(),String.format("%.1f",tcs));
        var =
DatoLaminacion.ModificaVarPalanquilla(DatoLaminacion.tipoVar.TEMPERATURA_
CORAZON,var, String.format("%.1f",tcc));
        var =
DatoLaminacion.ModificaVarPalanquilla(DatoLaminacion.tipoVar.TEMPERATURA_
INFERIOR,var,String.format("%.1f",tci));

       palanquillaTracking.setVar(var);

        //if (false)//palanquillaHOT.ID.equals("12L0LJ400"))
        {
           ImprimeRegistro(palanquillaHOT.ID, new double[]
           {
              abcisa / 1000, Ta_sup - 273, Ta_inf - 273, palanquillaHOT.T[1][0]-273,
palanquillaHOT.T[7][0]-273, palanquillaHOT.T[4][0]-273
           });
        }

     }
     //System.out.println("");

     //Hacer los cálculos térmicos
     //Ubicar los path de las palanquillas en perfil y planta
     //palanquillaHOT.Calcular_Proceso();
     for (int k = 0; k < listaTC.size(); k++)
     { //Recorremos la listaTC y vemos cúal/cuales sobran
        boolean no_uso = true;
        PalanquillaHOT palHot = listaTC.get(k);
        for (int j = 0; j < listaHorno.size(); j++)
```

```
                {
                    if (listaHorno.get(j).ID.equals(listaTC.get(k).getID()))
                    {
                        no_uso = false;
                    }
                }
                if (no_uso)
                {
                        //panePlanta.getChildren().remove(listaTC.get(i).pathPlanta);
                    //panePerfil.getChildren().remove(listaTC.get(i).pathPerfil);
                    root.getChildren().remove(palHot.getPathPerfil());
                    root.getChildren().remove(palHot.getPathPlanta());
                    listaTC.remove(k);
                    k--;
                }

            }
        }
        catch (ServiceException ex)
        {
            Global.info.Registra(ex);
        }
    }), new KeyFrame(Duration.seconds(60)));

    temporizador.setCycleCount(Timeline.INDEFINITE);
    temporizador.play();
}

public HOT(Esqueleto esqueleto)
{
    this.esqueleto=esqueleto;
    coloresFondo = new java.awt.Color[20];
    coloresFondo[0] = new java.awt.Color(0x90ee90); //lightgreen
    coloresFondo[1] = new java.awt.Color(0x20b2aa); //lightseagreen
    coloresFondo[2] = new java.awt.Color(0xdda0dd); //plum
    coloresFondo[3] = new java.awt.Color(0xbc8f8f); //rosybrown
    coloresFondo[4] = new java.awt.Color(0x1e90ff); //dodgerblue
    coloresFondo[5] = new java.awt.Color(0xee82ee); //violet
    coloresFondo[6] = new java.awt.Color(0xf4a460); //sandybrown
    coloresFondo[7] = new java.awt.Color(0xffff00); //yellow
    coloresFondo[8] = new java.awt.Color(0xffa500); //orange
    coloresFondo[9] = new java.awt.Color(0x00ffff); //aqua
    coloresFondo[10] = new java.awt.Color(0xd2691e); //chocolate
    coloresFondo[11] = new java.awt.Color(0xbdb76b); //darkkhaki
    coloresFondo[12] = new java.awt.Color(0x00ced1); //darkturquoise
    coloresFondo[13] = new java.awt.Color(0xfa8072); //salmon
    coloresFondo[14] = new java.awt.Color(0x00fa9a); //mediumspringgreen
    coloresFondo[15] = new java.awt.Color(0xffd700); //gold
    coloresFondo[15] = new java.awt.Color(0x4682b4); //steelblue
    coloresFondo[16] = new java.awt.Color(0xff6347); //tomato
    coloresFondo[17] = new java.awt.Color(0x708090); //slategrey
    coloresFondo[18] = new java.awt.Color(0x6495ed); //cornflowerblue
    coloresFondo[19] = new java.awt.Color(0x9acd32); //yellowgreen

    try
```

```
            {
                laminacion = new Laminacion();
            }
            catch (ServiceException | StatusException | ServiceResultException | ExcepcionOPC |
                URISyntaxException | SecureIdentityException | IOException ex)
            {
                Global.info.Registra(ex);
            }

            root = new Group();
        }

        public void updateWidth(double newWidth)
        {

            root.getChildren().clear();
            prueba.kx = newWidth / 1264;
            root.getChildren().addAll(prueba.getHorno());
            for (PalanquillaHOT palanquilla : listaTC)
            {
                palanquilla.setPathPlanta(prueba.getPalanquillaPlanta(palanquilla.anchoPalanquilla,
                 palanquilla.longitudPalanquilla, palanquilla.color, 0));
                palanquilla.setPathPerfil(prueba.getPalanquillaPerfil(palanquilla.anchoPalanquilla,
                 palanquilla.color, 0));
                //palanquillaHOT.setAbcisa(abcisa, prueba.escalaLargo);
                root.getChildren().add(palanquilla.getPathPerfil());
                root.getChildren().add(palanquilla.getPathPlanta());
                palanquilla.setAbcisa(palanquilla.abcisa, prueba.escalaLargo);
            }
            setSizeFuente((int) (newWidth * 12 / 1264 + 0.5), root.getChildren());
        }

        public void updateHeight(double newHeight)
        {
            root.getChildren().clear();
            prueba.ky = newHeight / 946;
            root.getChildren().addAll(prueba.getHorno());
            for (PalanquillaHOT palanquilla : listaTC)
            {

                palanquilla.setPathPlanta(prueba.getPalanquillaPlanta(palanquilla.anchoPalanquilla,
                 palanquilla.longitudPalanquilla, palanquilla.color, 0));
                palanquilla.setPathPerfil(prueba.getPalanquillaPerfil(palanquilla.anchoPalanquilla,
                 palanquilla.color, 0));
                palanquilla.setAbcisa(palanquilla.abcisa, prueba.escalaLargo);
                //palanquillaHOT.setAbcisa(abcisa, prueba.escalaLargo);
                root.getChildren().add(palanquilla.getPathPerfil());
                root.getChildren().add(palanquilla.getPathPlanta());

            }
            setSizeFuente((int) (newHeight * 12 / 946 + 0.5), root.getChildren());
        }
    }
```

1.5.4 Clase PalanquillaHOT

```
package hot;

import com.DatoLaminacion;
import static com.DatoLaminacion.ObtieneVarPalanquilla;
import com.DatoLaminacion.tipoVar;
import com.estado.Registro;
import com.interfaz.ModalDialog;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.effect.ColorAdjust;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Background;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.Paint;
import javafx.scene.shape.Path;
import javafx.scene.text.Font;
import javafx.scene.text.Text;

/**
 *
 * @author SONIA
 */
public class PalanquillaHOT extends Transferencia_Calor
{

    public double anchoPalanquilla;
    public double longitudPalanquilla;
    public String ID;
    public javafx.scene.paint.Color color;
    public Path pathPalanquillaPlanta;
    public Path pathPalanquillaPerfil;
    private int codigoBisra;
    double abcisa;
    Registro registroTraza;

    static public int getCodigoBisra(int tipo_acero)
    {
        switch (tipo_acero)
        {
            case 900:
            case 917:
            case 500:
            case 550:
            case 502:
            case 512:
```

```
            case 520:
            case 510:
            case 519:
            case 529:
            case 300:
            case 406:
            case 600:
            case 23:
            case 910:
            case 911:
            case 903:
            case 915:
            case 916:
                return 21;
            default:
                return 19;
        }
    }

    public PalanquillaHOT(Registro registro, javafx.scene.paint.Color color, int codigoBisra)
    {
        super(codigoBisra, 3, 3,
        Integer.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.ANCHO_PAL
        ANQUILLA, registro.getDat()))/1000.0, new double[]
        {
            0, 0, 0, 0
        },60);
        this.registroTraza = registro;
        this.ID = registro.ID;
        this.color = color;
        this.codigoBisra = codigoBisra;

        this.anchoPalanquilla =
        Integer.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.ANCHO_PAL
        ANQUILLA, registro.getDat()));
        this.longitudPalanquilla =
        Integer.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.LONGITUD_P
        ALANQUILLA, registro.getDat()));

    }

    public String getID()
    {
        return ID;
    }

    public Path getPathPlanta()
    {
        return pathPalanquillaPlanta;

    }

    public void setPathPlanta(Path pathPalanquillaPlanta)
    {
        this.pathPalanquillaPlanta = pathPalanquillaPlanta;
```

```
this.pathPalanquillaPlanta.setOnMouseClicked(new EventHandler<MouseEvent>()
{

    @Override
    public void handle(MouseEvent event)
    {
        Image image = new Image(getClass().getResourceAsStream("/Images/Niño_Puñeta.gif"));
        ImageView img = new ImageView(image);
        VBox vbox = new VBox(20);
        vbox.setAlignment(Pos.CENTER);

        GridPane grid = new GridPane();
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setPadding(new Insets(0, 0, 25, 0));
        grid.setAlignment(Pos.CENTER);
        EtiquetaColor l;

        int peso =
        Integer.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.PESO,
        registroTraza.getDat()));
        double longitud =
        Double.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.LONGIT
        UD_PALANQUILLA, registroTraza.getDat())) / 1000;
        double espesor_ancho =
        Double.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.ANCHO_
        PALANQUILLA, registroTraza.getDat())) / 1000.0;
        int tipo_acero =
        Integer.valueOf(DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.TIPO_AC
        ERO, registroTraza.getDat()));
        String txtFecha =
        DatoLaminacion.ObtieneDatoPalanquilla(DatoLaminacion.tipoDato.HORA_MOVIMIENTO,
        registroTraza.getDat());
        int perm = (int) ChronoUnit.SECONDS.between(LocalDateTime.parse(txtFecha,
        DateTimeFormatter.ofPattern("yyyyMMddHHmm")), LocalDateTime.now()) / 60;
        double abcisa = Double.valueOf(ObtieneVarPalanquilla(DatoLaminacion.tipoVar.ABCISA,
        registroTraza.getVar()).replace(",", "."))/1000;
        double tempSuperior =
        Double.valueOf(ObtieneVarPalanquilla(DatoLaminacion.tipoVar.TEMPERATURA_SUPERIO
        R, registroTraza.getVar()).replace(",", "."))-273;
        double tempInferior =
        Double.valueOf(ObtieneVarPalanquilla(DatoLaminacion.tipoVar.TEMPERATURA_INFERIO
        R, registroTraza.getVar()).replace(",", "."))-273;
        double tempCorazon =
        Double.valueOf(ObtieneVarPalanquilla(DatoLaminacion.tipoVar.TEMPERATURA_CORAZO
        N, registroTraza.getVar()).replace(",", "."))-273;

        Text ID_pal = new Text(registroTraza.ID);
        ID_pal.setFill(Color.RED);
        ID_pal.setFont(Font.font("Arial", 28));
        grid.add(ID_pal, 0, 0, 2, 1);

        EtiquetaColor ancho = new EtiquetaColor("Espesor/Ancho:", Color.WHITE);
        grid.add(ancho, 0, 1);
```

```
EtiquetaColor valor_ancho = new EtiquetaColor(Double.toString(espesor_ancho) + " m",
Color.TURQUOISE);
grid.add(valor_ancho, 1, 1);

EtiquetaColor labelLongitud = new EtiquetaColor("Longitud:", Color.WHITE);
grid.add(labelLongitud, 0, 2);
EtiquetaColor valor_longitud = new EtiquetaColor(Double.toString(longitud) + " m",
Color.TURQUOISE);
grid.add(valor_longitud, 1, 2);

EtiquetaColor labelPeso = new EtiquetaColor("Peso:", Color.WHITE);
grid.add(labelPeso, 0, 3);
EtiquetaColor valor_peso = new EtiquetaColor(Integer.toString(peso) + " Kg",
Color.TURQUOISE);
grid.add(valor_peso, 1, 3);

EtiquetaColor labelPosicion = new EtiquetaColor("Posicion:", Color.WHITE);
grid.add(labelPosicion, 0, 4);
EtiquetaColor valorPosicion = new EtiquetaColor(Double.toString(abcisa) + " m",
Color.TURQUOISE);
grid.add(valorPosicion, 1, 4);

EtiquetaColor labelTiempo = new EtiquetaColor("Tiempo en el horno:", Color.WHITE);
grid.add(labelTiempo, 0, 5);
EtiquetaColor valorTiempo = new EtiquetaColor(/*Integer.toString(perm) + */"37 min",
Color.TURQUOISE);
grid.add(valorTiempo, 1, 5);

EtiquetaColor labelTipoAcero = new EtiquetaColor("Tipo de acero:", Color.WHITE);
grid.add(labelTipoAcero, 0, 6);
EtiquetaColor valorTipoAcero = new EtiquetaColor(Integer.toString(tipo_acero),
Color.TURQUOISE);
grid.add(valorTipoAcero, 1, 6);

EtiquetaColor labelTemperaturaSuperior=new EtiquetaColor("Temperatura Superior:",
Color.WHITE);
grid.add(labelTemperaturaSuperior, 0, 7);
EtiquetaColor valorTemperaturaSuperior=new
EtiquetaColor(String.format("%.1f",tempSuperior)+" ºC",Color.TURQUOISE);
grid.add(valorTemperaturaSuperior,1,7);

EtiquetaColor labelTemperaturaInferior=new EtiquetaColor("Temperatura Inferior:",
Color.WHITE);
grid.add(labelTemperaturaInferior, 0, 8);
EtiquetaColor valorTemperaturaInferior=new
EtiquetaColor(String.format("%.1f",tempInferior)+" ºC",Color.TURQUOISE);
grid.add(valorTemperaturaInferior,1,8);

EtiquetaColor labelTemperaturaCorazon=new EtiquetaColor("Temperatura Corazón:",
Color.WHITE);
grid.add(labelTemperaturaCorazon, 0, 9);
EtiquetaColor valorTemperaturaCorazon=new
EtiquetaColor(String.format("%.1f",tempCorazon)+" ºC",Color.TURQUOISE);
grid.add(valorTemperaturaCorazon,1,9);
```

```
            //vbox.getChildren().addAll(grid,img);
            vbox.getChildren().add(grid);

            ColorAdjust colorAdjust = new ColorAdjust();
            colorAdjust.setContrast(0);
            colorAdjust.setHue(0);
            colorAdjust.setBrightness(0.3);
            colorAdjust.setSaturation(-0.75);

            HOT.root.setEffect(colorAdjust);
            Paint paint = Principal.scene.getFill();
            Background bg=HOT.pane.getBackground();
            HOT.pane.setBackground(null);

            ModalDialog dialogo = new ModalDialog(Principal.esqueleto, vbox, true);
            dialogo.boxButtons.getChildren().remove(dialogo.btnCancelar);
            if (dialogo.ShowModal()) //Se pulsa aceptar
            {
               HOT.root.setEffect(null);
               HOT.pane.setBackground(bg);
            }
            else
            {
               //Pulso cancela
            }

         }
      });
   }

   public Path getPathPerfil()
   {
      return pathPalanquillaPerfil;

   }

   public void setPathPerfil(Path pathPalanquillaPerfil)
   {
      this.pathPalanquillaPerfil = pathPalanquillaPerfil;
   }

   public void setAbcisa(double abcisa, double escala)
   {
      this.abcisa = abcisa;
      pathPalanquillaPerfil.setTranslateX(abcisa * escala);
      pathPalanquillaPlanta.setTranslateX(abcisa * escala);
   }

   @Override
   public double getEmisividad(double T, int S)
   {
      double emisividad = 0.0;
      double factorSombra = 0.85;
      if (S == 1 || S == 3)
      {
```

```
            return 0;
        }

        if (T <= 750 + 273)
        {
            emisividad = 0.65;
        }
        else if (T >= 800 + 273)
        {
            emisividad = 0.85;
        }
        else
        {
            emisividad = (0.85 - 0.65) / (800 - 750) * (T - 750) + 0.65;
        }

        if (S == 0)
        {
            return emisividad;
        }

        if (S == 2)
        {
            return emisividad * factorSombra;
        }

        return -1;
    }
}
```

1.5.5 Clase Principal

```
package hot;

import com.Global;
import com.interfaz.Esqueleto;
import com.interfaz.grafica.CategoriaGrafica;
import com.interfaz.grafica.SimpleConfigSerie;
import java.util.ArrayList;
import java.util.List;
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Rectangle2D;
import javafx.scene.Scene;
import javafx.stage.Screen;
import javafx.stage.Stage;

/**
 *
 * @author SONIA
 */
public class Principal extends Application
{

    static public Esqueleto esqueleto;

    static public Scene scene;

    @Override
    public void start(Stage primaryStage)
    {

        esqueleto = new Esqueleto(System.getProperty("user.dir"), "sonia", primaryStage,
                null);

//        try
        {

            primaryStage.setTitle("Path");

            Screen screen = Screen.getPrimary();

            double ancho = screen.getBounds().getWidth();
            double alto = screen.getBounds().getHeight();
            Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
            primaryStage.setX(primScreenBounds.getMinX());
            primaryStage.setY(primScreenBounds.getMinY());
            primaryStage.setWidth(primScreenBounds.getWidth());
            primaryStage.setHeight(primScreenBounds.getHeight());

            HOT hot = new HOT(esqueleto);
            scene = new Scene(hot.getVentana());

            //scene.setFill(hot.linearGrad);
```

```
primaryStage.setScene(scene);
primaryStage.show();
hot.update(scene.getWidth(), scene.getHeight());

scene.widthProperty().addListener(new ChangeListener<Number>()
{

   @Override
   public void changed(ObservableValue<? extends Number> observable, Number
   oldValue, Number newValue)
   {
      hot.updateWidth(newValue.doubleValue());
   }
});

scene.heightProperty().addListener(new ChangeListener<Number>()
{

   @Override
   public void changed(ObservableValue<? extends Number> observable, Number
   oldValue, Number newValue)
   {
      hot.updateHeight(newValue.doubleValue());
   }
});


   scene.getStylesheets().add(Esqueleto.class.getResource("general.css").toExter
   nalForm());
esqueleto.scene = scene;

Global.info.Info("Inicio");

int col = 3;
int fil = 3;
double dim = 0.150;
/* double hc[] =
 {
 0, 0, 0, 0
 };

 double emisividad[] =
 {
 0.95, 0.8, 0.9, 0.8
 };
 */

List<CategoriaGrafica> categorias = new ArrayList<CategoriaGrafica>();
CategoriaGrafica cg;
categorias.add(cg = new CategoriaGrafica("Temperatura"));
cg.listaConfigSerie.add(new SimpleConfigSerie("Pared"));
cg.listaConfigSerie.add(new SimpleConfigSerie("Corazón"));

/*GraficaSwingTiempoAuto grafica = new GraficaSwingTiempoAuto("Prueba", null,
   categorias, "Prueba", "dd/MM/yy HH:mm:ss", Duration.millis(10))
```

```
                {

                @Override
                public void addMuestra()
                {
                Palanquilla.Calcular_Proceso(new double[]
                {
                1200, 1200, 1200, 1200
                });
                addMuestra(LocalDateTime.now(), Palanquilla.T[0][0], "Pared");
                addMuestra(LocalDateTime.now(), Palanquilla.T[4][0], "Corazón");
                }
                };

                grafica.setMaxItemAge(60000);
                raiz.getChildren().add(grafica.getGrafica());*/
               //primaryStage.show();
            }
            /*catch (URISyntaxException | SecureIdentityException | IOException |
                    ServiceException | StatusException | ServiceResultException | ExcepcionOPC
                    ex)
             {
             Global.info.Registra(ex);
             }*/
        }

        public static void main(String[] args)
        {

            launch(args);
        }

    }
```

1.5.6 Clase TransferenciaCalor

```
package hot;

import Jama.Matrix;
import static java.lang.Math.abs;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.event.ActionEvent;
import javafx.util.Duration;

/**
 *
 * @author SONIA
 */
public abstract class Transferencia_Calor
{

    private int num_col;
    private int num_fil;
    private double a; //No es la dimensión de la palanquilla. Se refiere a la región en la que
    //subdividimos los elementos de la malla. Elemento malla= 2ax2b y hemos fijado que a=b
    private double dim;
    public double hc[];
    public double hr[];
    public double Ta[];
    public double den = 7850.0; //Densidad acero: 7850 kg/m^3
    // public double cp; //460J/(Kg*K)
    public double conductividad;
    public double constante_Stefan = 5.67 * Math.pow(10, -8);//
    public double C[][];
    public double K[][];
    public double f[][];
    public double T[][];
    public int g_i[][];
    public int num_nodos;
    public double t = 1.0;
    private final int codigoBisra;

    /**
     * ** CONSTRUCTOR *****
     */
    public Transferencia_Calor(int codigoBisra, int num_col, int num_fil, double dim, double hc[],
                double incrementoTiempo)
    {
        this.codigoBisra = codigoBisra;
        this.num_col = num_col;
        this.num_fil = num_fil;
        this.hc = hc;
        this.dim = dim;
```

```
    this.t = incrementoTiempo;
    a = dim / (2 * (num_fil - 1)); //num_fil=num_col
    Ta = new double[4];
    num_nodos = num_fil * num_col;
    C = new double[num_nodos][num_nodos];
    K = new double[num_nodos][num_nodos];
    f = new double[num_nodos][1];
    T = new double[num_nodos][1];
    g_i = new int[num_fil][num_col];

}

double getConductividad(double T)
{
    return BISRA.getConductividad(T, codigoBisra);
}

double getCalorEspecifico(double T)
{
    return BISRA.getCalorEspecifico(T, codigoBisra);
}

/* getTemperatura: Para obtener la temperatura del nodo i*/
double getTemperatura(int i)
{
    return T[i][0];
}

double getCoeficienteRadiacion(double T, int S)
{
    double hr;

    hr = getEmisividad(T, S) * constante_Stefan
            * (Math.pow(T, 2) + Math.pow(Ta[S], 2)) * (Math.pow(T, 1) + Math.pow(Ta[S], 1));

    return hr;
}

void Inicializar()
{
    int i, j;

    for (i = 0; i < num_nodos; i++)
    {
        T[i][0] = 293; //Inicialmente la palanquilla está a temperatura ambiente.
    }
    /* Inicialización de matrices */
    for (i = 0; i < num_nodos; i++)
    {
        for (j = 0; j < num_col * num_fil; j++)
        {
            C[i][j] = 0;
            K[i][j] = 0;
        }
    }
```

```
}

void Actualizar_Valores(double[] Ta)
{

   for (int i = 0; i < 4; i++)
   {
      this.Ta[i] = Ta[i];
   }

}

void Construir_Componentes()
{

   Implementacion_C();
   Matriz_K();
   Vector_f();

}

void Calcular_T()
{
   int i;
   Matrix Capacitancia = new Matrix(C);
   Matrix Conductividad = new Matrix(K);
   Matrix Fuerza = new Matrix(f);
   Matrix Temperatura = new Matrix(T);
   Matrix I = Matrix.identity(num_nodos, num_nodos);

   Matrix Part1;
   Matrix Part2;
   Matrix Prueba;
   Part1 = (I.minus(Capacitancia.inverse().times(Conductividad).times(t))).times(Temperatura);
   Part2 = Capacitancia.inverse().times(Fuerza).times(t);
   //Prueba=Conductividad.inverse().times(Fuerza);

   Temperatura = Part1.plus(Part2);
   //Temperatura=Prueba;

   for (i = 0; i < num_nodos; i++)
   {
      T[i][0] = Temperatura.get(i, 0);

   }

}

void Calcular_Proceso(double[] Ta)
{

   Actualizar_Valores(Ta);
   Construir_Componentes();
   Calcular_T();
```

```
}


double Valor_Temperatura(double x_ref, double y_ref)
{
   int num=1;
   int fil = 0,col = 0;
   int nfil = -1,ncol = -1;
   try
   {
      int n = num_col - 1;//Número de elementos por fila
      int m = num_fil - 1; //Número de elementos por columna
      double i, x0 = 0, y0 = 0;

      double x = 0, y = 0;
      //En nuestro caso, si trabajamos con matrices cuadradas n=m

      if (x_ref== dim){
         x0=dim-(dim/n/2);
         col=n-1;
      }
      else
         for (i = 0, num = 1; i < dim; i = i + dim / n, num++)
         {
            ncol++;
            if (i <= x_ref && x_ref <num * dim / n)
            {
               x0 = (i + num * dim / n) / 2;
               col=ncol;
               break;
            }


         }

      if (y_ref== dim){
         y0=dim-(dim/m/2);
         fil=m-1;
      }
      else
         for (i = 0, num = 1; i <dim; i = (i + dim / m), num++)
         {
             nfil++;
             if (i <= y_ref && y_ref <num * dim / m)
            {
               y0 = (i + num * dim / m) / 2;
               fil=nfil;
               break;
            }
         }

      x = x_ref - x0;
      y = -y_ref + y0;
```

```
        //FUNCIONES DE FORMA
        double Nsd = ((a + x) * (a + y)) / (4 * a * a);
        double Nsi = ((a - x) * (a + y)) / (4 * a * a);
        double Nid = ((a + x) * (a - y)) / (4 * a * a);
        double Nii = ((a - x) * (a - y)) / (4 * a * a);

        int Tsi = fil * num_col + col;
        int Tsd = fil * num_col + col + 1;
        int Tii = (fil + 1) * num_col + col;
        int Tid = (fil + 1) * num_col + col + 1;

        double Temp = Nsd * T[Tsd][0] + Nsi * T[Tsi][0] + Nid * T[Tid][0] + Nii * T[Tii][0];

        return Temp;
    }
    catch (Exception ex)
    {
        Logger.getLogger(Transferencia_Calor.class.getName()).log(Level.SEVERE, null, ex);
        return 0;
    }
}

/**
 * MATRIZ DE CAPACITANCIA*
 */
void Implementacion_C()
{
    double num = 0.0;

    Implementacion_G_i();

    for (int i = 0; i < num_nodos; i++)
    {

        for (int j = 0; j < num_nodos; j++)
        {

            if (i == j)
            {
                C[i][j] = g_i[i / num_col][i % num_col] * getDdensidad(getTemperatura(i)) *
                    getCalorEspecifico(getTemperatura(i)) * a * a;
            }
            else
            {
                C[i][j] = 0;
            }

        }

    }

}

/**
 * getConductividad: El siguiente método es abstracto (aparece en el mismo
```

```
 * sitio en el que se creó un objeto de la clase Transferencia_Calor. En
 * nuestro caso y, por ahora, en el main) y se encarga de devolver el valor
 * de la conductividad del material en funición de la temperatura a la que
 * se encuentre sometido el nodo de ese elemento en el instante t**
 */
/**
 * * Implementacion_K: Implementación de la matriz de resistencia*
 */
double Implementacion_K(int i, int j)
{
   double num = 0.0, sum = 0.0;
   //int num_nodos;
   double h_Ta[] =
   {
      0.0, 0.0
   };
   //num_nodos = num_col * num_fil;
   if (i == j)
   {
      for (int k = 0; k < num_nodos; k++)
      {
         h_Ta = Implementacion_h(i, k);
         sum = sum + alpha(i, k) * 2 * a / 3.0 * h_Ta[0];
      }

      num = Implementacion_G_i_Otra_version(i) * 2 * getConductividad(getTemperatura(i)) /
            3.0 + sum;
   }

   else if (i != j)
   {

      if (alpha(i, j) == 1)
      {  //Son contiguos

         num = -1 * Implementacion_G_i_j(i, j) * getConductividad(getTemperatura(i)) / 6.0
               + (h_Ta[0] * a / 3.0);

      }

      else if (beta(i, j) == 1)
      { //Son opuestos

         num = -getConductividad(getTemperatura(i)) / 3.0;
      }

      else if (alpha(i, j) == 0 && beta(i, j) == 0)
      { //Ni son contiguos ni son opuestos

         num = 0;
      }

   }

   return num;
```

```
}

///*** Función que rellena la matriz K***///
void Matriz_K()
{
   int i, j;
   double sum;
   double h_Ta[] =
   {
      0.0, 0.0
   };

   for (i = 0; i < num_nodos; i++)
   {
      for (j = 0; j < num_nodos; j++)
      {
         if (i == j)
         {
            sum = 0.0;
            for (int k = 0; k < num_nodos; k++)
            {
               h_Ta = Implementacion_h(i, k);
               sum = sum + alpha(i, k) * 2 * a / 3.0 * h_Ta[0];
            }

            K[i][j] = Implementacion_G_i_Otra_version(i) * 2 *
              getConductividad(getTemperatura(i)) / 3.0 + sum;
         }

         else if (i != j)
         {

            if (alpha(i, j) == 1)
            {  //Son contiguos
               h_Ta = Implementacion_h(i, j);

               K[i][j] = -1 * Implementacion_G_i_j(i, j) * getConductividad(getTemperatura(i)) /
              6.0
                     + (h_Ta[0] * a / 3.0);

            }

            else if (beta(i, j) == 1)
            { //Son opuestos

               K[i][j] = -getConductividad(getTemperatura(i)) / 3.0;
            }

            else if (alpha(i, j) == 0 && beta(i, j) == 0)
            { //Ni son contiguos ni son opuestos

               K[i][j] = 0;
            }
```

```
            }

        }

    }

}

/**
 * ** Implementacion f: Vector de fuerza externa ***
 */
double Implementacion_f(int i)
{

    double sum = 0.0;
    double h_Ta[] =
    {
        0.0, 0.0
    };

    for (int k = 0; k < num_nodos; k++)
    {
        h_Ta = Implementacion_h(i, k);
        sum = sum + alpha(i, k) * h_Ta[0] * h_Ta[1] * a;

    }

    return sum;

}

///***Función que rellena el vector f ***///
void Vector_f()
{

    int i;
    double h_Ta[] =
    {
        0.0, 0.0
    };

    double sum;
    for (i = 0; i < num_nodos; i++)
    {
        sum = 0.0;
        for (int k = 0; k < num_nodos; k++)
        {

            h_Ta = Implementacion_h(i, k);
            sum = sum + alpha(i, k) * h_Ta[0] * h_Ta[1] * a;

        }
        f[i][0] = sum;
    }
}
```

```
/**
 * ** G_i: Función que devuelve el número de elementos compartidos por
 * elnodo i ****
 */
// Dos versiones: 1)se genera una matriz con los valores de G_i
//                2)se le pasa un nodo y devuelve su número de elementos compartidos.
void Implementacion_G_i()
{

   int j, i;

   /* En los extremos esta función siempre vale 1 */
   g_i[0][0] = 1;
   g_i[0][num_col - 1] = 1;
   g_i[num_fil - 1][0] = 1;
   g_i[num_fil - 1][num_col - 1] = 1;

   //Fila 0 y fila última
   for (j = 1; j < num_col - 1; j++)
   {
      g_i[0][j] = 2;
      g_i[num_fil - 1][j] = 2;
   }
   //Columna 0 y columna última
   for (i = 1; i < num_fil - 1; i++)
   {
      g_i[i][0] = 2;
      g_i[i][num_col - 1] = 2;

   }

   // Elementos centrales cuyo valor siempre es 4
   for (i = 1; i <= num_fil - 2; i++)
   {
      for (j = 1; j <= num_col - 2; j++)
      {
         g_i[i][j] = 4;
      }
   }

}

int Implementacion_G_i_Otra_version(int i)
{

   int num = 0;
   int i1, j1;
   i1 = i / num_col;
   j1 = i % num_col;

   if ((i1 == 0 && j1 == 0) || (i1 == 0 && j1 == num_col - 1) || (j1 == 0 && i1 == num_fil - 1)
         || (i1 == num_fil - 1 && j1 == num_col - 1))
   {
      //Es extremo
```

```
      num = 1;

   }
   else if (i1 == 0 || i1 == num_fil - 1 || j1 == 0 || j1 == num_col - 1)
   {
      num = 2;
   }

   else
   {
      //Está en el centro
      num = 4;
   }

   return num;

}

/**
 * G_i_j: Función que devuelve el número de elementos compartidos por la *
 * línea que une i,j ***
 */
int Implementacion_G_i_j(int i, int j)
{
   int num = 0;
   int i1, i2, j1, j2;
   i1 = i / num_col;
   j1 = i % num_col;
   i2 = j / num_col;
   j2 = j % num_col;

   if ((i - j) == 0)
   {
      //Llamada a Implementacion G_i
      num = g_i[i / num_col][i % num_col];

   }

   //¿Es contiguo?
   else if (i1 == i2 && abs(j1 - j2) == 1)
   {
      //Es contiguo en fila
      if (i1 == 0 || i1 == (num_fil - 1))
      {
         //Está en la periferia, bien en la fila 0 o en la última fila.
         num = 1;
      }

      else
      { //Es contiguo pero no está en la periferia.
         num = 2;
      }

   }
   else if (j1 == j2 && abs(i1 - i2) == 1)
```

```
    {

        //También es contiguo pero ahora en columna
        if (j1 == 0 || j1 == (num_col - 1))
        {
            //Está en la periferia también pero en este caso en la col 0 o col final.
            num = 1;
        }
        else
        { //Es contiguo pero no está en la periferia.
            num = 2;
        }
    }

    else
    { // No es contiguo
        num = 0;
    }

    return num;
}

/* ALPHA: Función que vale 1 si los nodos i,j son contiguos y 0 en caso contrario*/
int alpha(int i, int j)
{
    int num = 0;
    int i1, i2, j1, j2;
    i1 = i / num_col;
    j1 = i % num_col;
    i2 = j / num_col;
    j2 = j % num_col;

    if ((i1 == i2 && abs(j1 - j2) == 1) || (j1 == j2 && abs(i1 - i2) == 1))
    {
        num = 1;
    }

    else
    {
        num = 0;
    }

    return num;
}

/* BETA: Función que vale '1' si los nodos i,j son opuestos Y pertenecen al mismo elemnto*/
int beta(int i, int j)
{

    int num = 0;
    int i1, i2, j1, j2;
    i1 = i / num_col;
    j1 = i % num_col;
    i2 = j / num_col;
    j2 = j % num_col;
```

```
    if (abs(i1 - i2) == 1 && abs(j1 - j2) == 1)
    { //Es opuesto
        num = 1;

    }

    else
    {
        num = 0;
    }

    return num;

}

/**
 * *Implementacion_h: Implementación del coeficiente de transferencia
 * térmica por convección y radiacion para los nodos i, j**
 */
double[] Implementacion_h(int i, int j)
{
    double num[] =
    {
        0.0, 0.0
    };

    int S;

    int i1, j1, i2, j2;
    i1 = i / num_col;
    j1 = i % num_col;
    i2 = j / num_col;
    j2 = j % num_col;

    if (i1 == i2 && j1 != j2) //Están en la misma fila
    {
        if (i1 == 0) //Superficie superior
        {
            S = 0;
            num[0] = hc[0] + getCoeficienteRadiacion(getTemperatura(i), S);
            num[1] = Ta[0];

        }
        else if (i1 == num_fil - 1)//Superficie inferior
        {
            S = 2;
            num[0] = hc[2] + getCoeficienteRadiacion(getTemperatura(i), S);
            num[1] = Ta[2];

        }
        else
        {
            num[0] = 0;
            num[1] = 0;
```

```
            }
        }

        else if (j1 == j2 && i1 != i2) //Están en la misma columna
        {

            if (j1 == 0) //Superficie izquierda
            {
                S = 1;
                num[0] = hc[1] + getCoeficienteRadiacion(getTemperatura(i), S);
                num[1] = Ta[1];

            }
            else if (j1 == num_col - 1) //Superficie derecha
            {
                S = 3;
                num[0] = hc[3] + getCoeficienteRadiacion(getTemperatura(i), S);
                num[1] = Ta[3];
            }
            else
            {
                num[0] = 0;
                num[1] = 0;
            }

        }

        else
        {
            num[0] = 0;
            num[1] = 0;
        }

        return num;
    }
```

1.5.7 Clase VistasHorno

```
package hot;

import com.interfaz.BorderedTitledPane;
import java.util.ArrayList;
import java.util.List;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.EventHandler;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.effect.Bloom;
import javafx.scene.effect.ColorAdjust;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.Light;
import javafx.scene.effect.Lighting;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Background;
import javafx.scene.layout.ColumnConstraints;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;
import javafx.scene.shape.HLineTo;
import javafx.scene.shape.LineTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.scene.shape.VLineTo;
import javafx.scene.text.Text;

/**
 *
 * @author SONIA
 */
public class VistasHorno
{

    //ATRIBUTOS
      Stop[] stops = new Stop[]
    {
       new Stop(0, Color.web("FFFF00", 0.3)), new Stop(0.5, Color.web("FFA500", 0.6)), new
     Stop(1, Color.web("FF0000", 0.6))
    };
    LinearGradient linearGrad = new LinearGradient(0, 0, 1, 0, true, CycleMethod.NO_CYCLE,
     stops);
```

```
double x0 = 150;
double y0_perfil = 200;
double y0_planta = 656;

double largoHorno = 14945.0;
double altoSuperior = 1480;
double altoSuperiorPrecalentamiento = 1080;
double altoInferior = 1820;
double anchoPlantaHorno = 7900;

double altoHorno = altoSuperior + altoInferior;
double altoHorno2 = altoSuperiorPrecalentamiento + altoInferior;

public double escalaLargo = 1080.0 / largoHorno;
public double escalaPlanta = escalaLargo / 3;
public double escalaPerfil = escalaLargo / 2;

double offsetInicial = 182.5;
double offsetFinal = 202.5;
double distanciaPrecalentamiento = 4960;
double distanciaCalentamiento = 9772.5;
double margenPaloma = 600;
double anchoChimenea = 1000;

double anchoMuro = 340;
double altoMuro = 1300;
double altoMuro2 = 1100;

//Variables Distancia Termopares
//Termopares de Bóveda
double termoBovedaPrecalentamiento = 3345;
double termoBovedaCalentamiento = 9675;
double termoBovedaIgualacion = 13318;
//Termopares de Solera
double termoSoleraPrecalentamiento = 3400;
double termoSoleraCalentamiento = 8681;
double termoSoleraIgualacion = 12560;

private double Bx_Perfil; //En principio, estaban como locales pero son necesarias para poder
 ubicar los cuatro pane de informacion.
private double Dx_Perfil;
private double Dy_Perfil;
private double Ay_Perfil;
private double Fy_Perfil;


private Text labelContador;
private final double ancho_escena;
private final double alto_escena;
public double kx;
public double ky;
```

```
public EtiquetaColor temperaturaTermoparPrecalentamientoSuperior = new
 EtiquetaColor(Color.BLUE);
public  EtiquetaColor temperaturaTermoparPrecalentamientoInferior = new
 EtiquetaColor(Color.BLUE);
public  EtiquetaColor temperaturaChimenea = new  EtiquetaColor(Color.BLUE);

ZonaHorno zonasHornoCalentamientoSuperior = new ZonaHorno(true, "Calentamiento
 Superior");
ZonaHorno zonasHornoCalentamientoInferior = new ZonaHorno(false, "Calentamiento
 Inferior");
ZonaHorno zonasHornoIgualacionSuperior = new ZonaHorno(true, "Igualación Superior");
ZonaHorno zonasHornoIgualacionInferior = new ZonaHorno(false, "Igualación Inferior");


//***CONSTRUCTOR
public VistasHorno(double ancho_escena, double alto_escena)
{
   this.ancho_escena = ancho_escena;
   this.alto_escena = alto_escena;

   kx = ancho_escena / 1264;
   ky = alto_escena / 946;

}

public Pane getCuadroValores(double x1, double x2, double y, ZonaHorno zona)
{
   //Pane pane=new Pane();
   EtiquetaColor l;
   VBoxCuadro pane = new VBoxCuadro(5);
   HBox hbox1 = new HBox(5);
   HBox hbox2 = new HBox(5);
   HBox hbox3 = new HBox(5);
   HBox hbox4 = new HBox(5);

   GridPane grid1 = new GridPane();
   grid1.setHgap(5);
   grid1.setVgap(5);
   grid1.add(l = new EtiquetaColor("Consigna:", Color.BLACK), 0, 0);
   grid1.add(zona.consigna, 1, 0);
   grid1.add(l = new EtiquetaColor("ºC", Color.BLUE), 2, 0);
   grid1.add(l = new EtiquetaColor("del Hot:", Color.BLACK), 3, 0);
   grid1.add(zona.consignaHOT, 4, 0);
   grid1.add(l = new EtiquetaColor("Medida:", Color.BLACK), 0, 1);
   grid1.add(zona.medida, 1, 1);
   grid1.add(l = new EtiquetaColor("ºC", Color.BLUE), 2, 1);

   hbox1.getChildren().addAll(l = new EtiquetaColor("Regulador salida:", Color.BLACK),
 zona.regul_salida, l = new EtiquetaColor("%", Color.CORAL));

   hbox2.getChildren().addAll(l = new EtiquetaColor("TC de regulación:", Color.BLACK),
 zona.TC_regulacion);

   GridPane grid2 = new GridPane();
   grid2.setHgap(7);
```

```
   grid2.setVgap(5);
   grid2.add(l = new EtiquetaColor("Pared:", Color.BLACK), 0, 0);
   grid2.add(zona.Temp_pared1, 1, 0);
   grid2.add(l = new EtiquetaColor("ºC", Color.ORANGERED), 2, 0);
   grid2.add(zona.Temp_pared2, 3, 0);
   grid2.add(l = new EtiquetaColor("ºC", Color.ORANGERED), 4, 0);
   grid2.add(l = new EtiquetaColor(zona.isZonaSuperior() ? "Bóveda:" : "Solera:",
Color.BLACK), 0, 1);
   grid2.add(zona.Temp_boveda1, 1, 1);
   grid2.add(l = new EtiquetaColor("ºC", Color.ORANGERED), 2, 1);
   grid2.add(zona.Temp_boveda2, 3, 1);
   grid2.add(l = new EtiquetaColor("ºC", Color.ORANGERED), 4, 1);

   hbox3.getChildren().addAll(l = new EtiquetaColor("Caudal gas:", Color.BLACK),
zona.caudalGas, l = new EtiquetaColor("Nm3/h", Color.GRAY));

   if (zona.isZonaSuperior() == false)
   {
      hbox4.getChildren().addAll(l = new EtiquetaColor("A:", Color.BLACK), zona.cteA,
            l = new EtiquetaColor("B:", Color.BLACK), zona.cteB);
   }

   pane.getChildren().addAll(grid1, hbox1, hbox2, grid2, hbox3, hbox4);
   //pane.getChildren().add(pane);

   pane.heightProperty().addListener(new ChangeListener<Number>()
   {

      @Override
      public void changed(ObservableValue<? extends Number> observable, Number oldValue,
Number newValue)
      {
         if (zona.isZonaSuperior())
         {
            pane.setLayoutY(y - newValue.doubleValue() - 10 * ky);
         }
         else
         {
            pane.setLayoutY(y + 10 * ky);
         }
      }
   });

   pane.widthProperty().addListener(new ChangeListener<Number>()
   {

      @Override
      public void changed(ObservableValue<? extends Number> observable, Number oldValue,
Number newValue)
      {
         pane.setLayoutX((x1 + x2) / 2 - newValue.doubleValue() / 2);
      }
   });

   DropShadow ds1 = new DropShadow();
```

```
    ds1.setOffsetY(2.0f);
    ds1.setOffsetX(2.0f);
    ds1.setColor(Color.MAROON);
    pane.setStyle("-fx-background-color:LEMONCHIFFON;");
    pane.setPadding(new Insets(10*ky,10*kx,10*kx,10*ky));
    pane.setEffect(ds1);
    return pane;
}

public Node ubicaLabel(String texto, EtiquetaColor etiqueta, double x, double y, boolean
 boveda)
{
    GridPane grid = new GridPane();
    grid.setHgap(5);
    grid.add(new EtiquetaColor(texto,Color.BLACK), 0, 0);
    grid.add(etiqueta, 0, 1);

    ColumnConstraints column1 = new ColumnConstraints();
    column1.setHalignment(HPos.CENTER);
    grid.getColumnConstraints().add(column1);
    grid.heightProperty().addListener(new ChangeListener<Number>()
    {

        @Override
        public void changed(ObservableValue<? extends Number> observable, Number oldValue,
 Number newValue)
        {
            if (boveda)
            {

                grid.setLayoutY(y - newValue.doubleValue() - 10 * ky);
            }
            else
            {

                grid.setLayoutY(y + 10 * ky);
            }
        }
    });

    grid.widthProperty().addListener(new ChangeListener<Number>()
    {

        @Override
        public void changed(ObservableValue<? extends Number> observable, Number oldValue,
 Number newValue)
        {
            grid.setLayoutX(x - newValue.doubleValue() / 2);
        }
    });

    DropShadow ds1 = new DropShadow();
    ds1.setOffsetY(2.0f);
    ds1.setOffsetX(2.0f);
    ds1.setColor(Color.MAROON);
```

```
    grid.setStyle("-fx-background-color:LEMONCHIFFON;");
    grid.setPadding(new Insets(10*ky,10*kx,10*kx,10*ky));
    grid.setEffect(ds1);
    return grid;
}

//OTROS MÉTODOS

//Función para dibujar los distintos termopares que existen en el horno.
public Path termoparHorno(double abcisa, double y, boolean arriba)
{

    Path termo = new Path();
    double t_x = x0 + (offsetInicial + abcisa) * escalaLargo;
    double t_y = y;

    MoveTo move_termo = new MoveTo();
    move_termo.setX(kx * (t_x - 7));
    move_termo.setY(ky * t_y);
    HLineTo hLineTo1_termo = new HLineTo();
    hLineTo1_termo.setX(kx * (t_x + 7));

    MoveTo move2_termo = new MoveTo();
    move2_termo.setX(kx * (t_x));
    move2_termo.setY(ky * t_y);
    VLineTo vLineTo_termo = new VLineTo();

    vLineTo_termo.setY(ky * (y + (arriba ? 7 : -7)));

    termo.getElements().addAll(move_termo, hLineTo1_termo, move2_termo, vLineTo_termo);
    termo.setStroke(Color.FIREBRICK);
    termo.setStrokeWidth(5);
    return termo;

}

//Función que dibuja los muros que hay en el horno.
public Path murosHorno(double distancia, double y, double alto, double ancho)
{

    Path muro = new Path();
    double m0_x = x0 + (offsetInicial + distancia - ancho / 2) * escalaLargo;
    double m0_y = y;
    double m1_x = m0_x;
    double m1_y = y - alto * escalaPerfil;
    double m2_x = m1_x + ancho * escalaLargo;
    double m2_y = m1_y;
    double m3_x = m2_x;
    double m3_y = m0_y;
    double m4_x = m0_x;
    double m4_y = m0_y;

    MoveTo move_muro = new MoveTo();
    move_muro.setX(kx * m0_x);
    move_muro.setY(ky * m0_y);
```

```
        LineTo line_muro1 = new LineTo();
        line_muro1.setX(kx * m1_x);
        line_muro1.setY(ky * m1_y);

        LineTo line_muro2 = new LineTo();
        line_muro2.setX(kx * m2_x);
        line_muro2.setY(ky * m2_y);

        LineTo line_muro3 = new LineTo();
        line_muro3.setX(kx * m3_x);
        line_muro3.setY(ky * m3_y);

        LineTo line_muro4 = new LineTo();
        line_muro4.setX(kx * m4_x);
        line_muro4.setY(ky * m4_y);

        muro.getElements().addAll(move_muro, line_muro1, line_muro2, line_muro3, line_muro4);
        muro.setStroke(Color.BLACK);
        muro.setStrokeWidth(3);
        muro.setFill(Color.BLACK);

        return muro;
    }

    //Función que dibuja las líneas divisorias de las distintas zonas de calentamiento.
    public Path lineasVerticales(double distancia, double y, double long_vertical)
    {

        Path linea_vertical = new Path();
        double l_x = x0 + (offsetInicial + distancia) * escalaLargo;
        double l_y = y;

        MoveTo move_l = new MoveTo();
        move_l.setX(kx * l_x);
        move_l.setY(ky * l_y);

        VLineTo vLineTo = new VLineTo();
        vLineTo.setY(ky * long_vertical);

        linea_vertical.getElements().addAll(move_l, vLineTo);
        linea_vertical.setStroke(Color.RED);
        linea_vertical.setStrokeWidth(3);

        return linea_vertical;
    }

    public Node[] getPerfil()
    {
        List<Node> paths = new ArrayList<>();
        Path currentPath;

        //******PATH PERFIL******
        MoveTo move_perfil = new MoveTo();
        move_perfil.setX(kx * x0);
```

```
    move_perfil.setY(ky * y0_perfil);

    //Coordenadas Perfil
    Ay_Perfil = y0_perfil + 1.5 * altoHorno * escalaPerfil;
    double Ax = x0;
    Bx_Perfil = x0 + largoHorno * escalaLargo;
    double By = Ay_Perfil;
    double Cx = Bx_Perfil;
    double Cy = Ay_Perfil - altoHorno * escalaPerfil;
    Dy_Perfil = Cy;
    Dx_Perfil = Bx_Perfil - (largoHorno - (offsetInicial + distanciaPrecalentamiento +
margenPaloma)) * escalaLargo;
    double Ex = Dx_Perfil - margenPaloma * 2 * escalaLargo;
    double Ey = Ay_Perfil - (altoInferior + altoSuperiorPrecalentamiento) * escalaPerfil;
    Fy_Perfil = Ey;
    double Fx = x0 + anchoChimenea * escalaLargo;
    double Gx = Fx;
    double Gy = y0_perfil;

    LineTo line_perfil1 = new LineTo();
    line_perfil1.setX(kx * Ax);
    line_perfil1.setY(ky * Ay_Perfil);

    LineTo line_perfil2 = new LineTo();
    line_perfil2.setX(kx * Bx_Perfil);
    line_perfil2.setY(ky * By);

    LineTo line_perfil3 = new LineTo();
    line_perfil3.setX(kx * Cx);
    line_perfil3.setY(ky * Cy);

    LineTo line_perfil4 = new LineTo();
    line_perfil4.setX(kx * Dx_Perfil);
    line_perfil4.setY(ky * Dy_Perfil);

    LineTo line_perfil5 = new LineTo();
    line_perfil5.setX(kx * Ex);
    line_perfil5.setY(ky * Ey);

    LineTo line_perfil6 = new LineTo();
    line_perfil6.setX(kx * Fx);
    line_perfil6.setY(ky * Fy_Perfil);

    LineTo line_perfil7 = new LineTo();
    line_perfil7.setX(kx * Gx);
    line_perfil7.setY(ky * Gy);

    currentPath = new Path();
    paths.add(currentPath);
    currentPath.getElements().addAll(move_perfil, line_perfil1, line_perfil2, line_perfil3,
line_perfil4, line_perfil5, line_perfil6, line_perfil7);
    currentPath.setStroke(Color.RED);
    currentPath.setStrokeWidth(3);
    currentPath.setFill(linearGrad);
```

```
    // PATH LÍNEA DIVISORIA ZONA PRECALENTAMIENTO-CALENTAMIENTO
    paths.add(lineasVerticales(distanciaPrecalentamiento, (Cy + Ey) / 2, Ay_Perfil));
    //PATH LÍNEA DIVISORIA ZONA CALENTAMIENTO-IGUALACIÓN
    paths.add(lineasVerticales(distanciaCalentamiento, Cy, Ay_Perfil));

    //********PATH MURO PRECALENTAMIENTO-CALENTAMIENTO*******
    paths.add(murosHorno(distanciaPrecalentamiento, Ay_Perfil, altoMuro, anchoMuro));
    //*******PATH MURO CALENTAMIENTO-IGUALACIÓN*******
    paths.add(murosHorno(distanciaCalentamiento, Ay_Perfil, altoMuro2, anchoMuro));

    //*******PATH LÍNEA DISCONTINUA: Plano circulación de barras
    //Coordenadas línea discontinua
    double l0_x = x0;
    double l0_y = Ay_Perfil - altoInferior * escalaPerfil;

    MoveTo move_dis = new MoveTo();
    move_dis.setX(kx * l0_x);
    move_dis.setY(ky * l0_y);

    HLineTo hLineTo = new HLineTo();
    hLineTo.setX(kx * Bx_Perfil);

    currentPath = new Path();
    paths.add(currentPath);
    currentPath.getElements().addAll(move_dis, hLineTo);
    currentPath.setStroke(Color.BLACK);
    currentPath.getStrokeDashArray().addAll(15.0, 5.0, 5.0, 5.);
    currentPath.setStrokeWidth(1);

    //*******TERMOPARES********//
    //Bóveda
    paths.add(termoparHorno(termoBovedaPrecalentamiento, Fy_Perfil, true));
    paths.add(termoparHorno(termoBovedaCalentamiento, Dy_Perfil, true));
    paths.add(termoparHorno(termoBovedaIgualacion, Dy_Perfil, true));
    //Solera
    paths.add(termoparHorno(termoSoleraPrecalentamiento, Ay_Perfil, false));
    paths.add(termoparHorno(termoSoleraCalentamiento, Ay_Perfil, false));
    paths.add(termoparHorno(termoSoleraIgualacion, Ay_Perfil, false));

    paths.add(ubicaLabel("Tª Recuperación Superior",
 temperaturaTermoparPrecalentamientoSuperior, (x0 + (offsetInicial +
 termoBovedaPrecalentamiento) * escalaLargo) * kx, ky * Fy_Perfil, true));
    paths.add(ubicaLabel("Tª Recuperación Inferior",
 temperaturaTermoparPrecalentamientoInferior, (x0 + (offsetInicial +
 termoBovedaPrecalentamiento) * escalaLargo) * kx, ky * Ay_Perfil, false));
    paths.add(ubicaLabel("Tª Humos", temperaturaChimenea, (x0 + (anchoChimenea / 2) *
 escalaLargo) * kx, ky * y0_perfil, true));

    return paths.toArray(new Node[paths.size()]); //Convertimos la lista a un array de Paths.
  }

  public Path[] getPlanta()
  {

    //Paths usados
```

```
Path planta = new Path();
Path l1 = new Path();
Path l2 = new Path();
Path offsetIni = new Path();
Path offsetFin = new Path();

//Coordenadas
double Ax = x0 + largoHorno * escalaLargo;
double Ay = y0_planta;
double Bx = Ax;
double By = y0_planta + anchoPlantaHorno * escalaPlanta;
double Cx = x0;
double Cy = By;
double Dx = x0;
double Dy = y0_planta;

// PATH LÍNEA PRECALENTEAMIENTO-CALENTAMIENTO
//*******PATH LÍNEA CALENTAMIENTO-IGUALACIÓN
//*******PATH PLANTA*******
MoveTo moveTo = new MoveTo();
moveTo.setX(kx * x0);
moveTo.setY(ky * y0_planta);

LineTo lineTo = new LineTo();
lineTo.setX(kx * Ax);
lineTo.setY(ky * Ay);

LineTo lineTo2 = new LineTo();
lineTo2.setX(kx * Bx);
lineTo2.setY(ky * By);

LineTo lineTo3 = new LineTo();
lineTo3.setX(kx * Cx);
lineTo3.setY(ky * Cy);

LineTo lineTo4 = new LineTo();
lineTo4.setX(kx * Dx);
lineTo4.setY(ky * Dy);

planta.getElements().addAll(moveTo, lineTo, lineTo2, lineTo3, lineTo4);
planta.setStroke(Color.RED);
planta.setStrokeWidth(3);
planta.setFill(linearGrad);

/* LinearGradient linearGrad = new LinearGradient(0,0, 0.5, 0, true,
 CycleMethod.REPEAT,
 new Stop(0.1f, Color.web("51E9FF"))
 ,new Stop(1.0f, Color.web("000000")));*/
//Color c = Color.BLUE;   //use the blue constant
// PATH LÍNEA DISCONTINUA OFFSET INICIAL
//Coordenadas
double Ix = x0 + offsetInicial * escalaLargo;
double Iy = y0_planta;
MoveTo move_offsetInicial = new MoveTo();
move_offsetInicial.setX(kx * Ix);
```

```
    move_offsetInicial.setY(ky * Iy);

    VLineTo vLine_offsetInicial = new VLineTo();
    vLine_offsetInicial.setY(ky * By);

    offsetIni.getElements().addAll(move_offsetInicial, vLine_offsetInicial);
    offsetIni.setStroke(Color.BLACK);
    offsetIni.getStrokeDashArray().addAll(15.0, 5.0, 5.0, 5.);
    offsetIni.setStrokeWidth(1);

    // PATH LÍNEA DISCONTIUA OFFSET FINAL
    //Coordenadas
    double Fx = x0 + (largoHorno - offsetFinal) * escalaLargo;
    double Fy = y0_planta;
    MoveTo move_offsetFin = new MoveTo();
    move_offsetFin.setX(kx * Fx);
    move_offsetFin.setY(ky * Fy);

    VLineTo vLine_offsetFin = new VLineTo();
    vLine_offsetFin.setY(ky * By);

    offsetFin.getElements().addAll(move_offsetFin, vLine_offsetFin);
    offsetFin.setStroke(Color.BLACK);
    offsetFin.getStrokeDashArray().addAll(15.0, 5.0, 5.0, 5.);
    offsetFin.setStrokeWidth(1);

    return new Path[]
    {
        planta, lineasVerticales(distanciaPrecalentamiento, y0_planta, By),
 lineasVerticales(distanciaCalentamiento, y0_planta, By), offsetIni, offsetFin
    };
}


// Palanquilla en Planta.
public Path getPalanquillaPlanta(double anchoPalanquilla, double longitudPalanquilla, Color
 color, double abcisa)
{

    Path palanquillaPlanta = new Path();
    double x1 = abcisa * escalaLargo + x0;
    double y1 = y0_planta + (anchoPlantaHorno / 2 - longitudPalanquilla / 2) * escalaPlanta;
    double y2 = y1 + longitudPalanquilla * escalaPlanta;
    double x2 = x1 + anchoPalanquilla * escalaLargo;

    MoveTo moveTo = new MoveTo();
    moveTo.setX(kx * x1);
    moveTo.setY(ky * y1);

    LineTo lineTo = new LineTo();
    lineTo.setX(kx * x1);
    lineTo.setY(ky * y2);

    LineTo lineTo2 = new LineTo();
    lineTo2.setX(kx * x2);
```

```
lineTo2.setY(ky * y2);

LineTo lineTo3 = new LineTo();
lineTo3.setX(kx * x2);
lineTo3.setY(ky * y1);

LineTo lineTo4 = new LineTo();
lineTo4.setX(kx * x1);
lineTo4.setY(ky * y1);

palanquillaPlanta.getElements().addAll(moveTo, lineTo, lineTo2, lineTo3, lineTo4);

palanquillaPlanta.setStroke(Color.BLACK);
palanquillaPlanta.setStrokeWidth(2);
palanquillaPlanta.setFill(color);

palanquillaPlanta.setOnMouseEntered(new EventHandler<MouseEvent>()
{

   @Override
   public void handle(MouseEvent event)
   {

      Bloom bloom = new Bloom();
      bloom.setThreshold(1);

      ColorAdjust colorAdjust = new ColorAdjust();
      colorAdjust.setContrast(0.1);
      colorAdjust.setHue(-0.05);
      colorAdjust.setBrightness(0.1);
      colorAdjust.setSaturation(0.7);

      Light.Distant light = new Light.Distant();
      light.setAzimuth(45.0);
      light.setElevation(30.0);

      Lighting lighting = new Lighting();
      lighting.setLight(light);
      lighting.setSurfaceScale(5.0);

      // palanquillaPlanta.setEffect(lighting);
      palanquillaPlanta.setStroke(Color.WHITE);
      palanquillaPlanta.setStrokeWidth(5);

      //System.out.println("Dentro");
   }
});

palanquillaPlanta.setOnMouseExited(new EventHandler<MouseEvent>()
{

   @Override
   public void handle(MouseEvent event)
   {
      palanquillaPlanta.setStroke(Color.BLACK);
```

```
            palanquillaPlanta.setStrokeWidth(2);
            palanquillaPlanta.setEffect(null);

        }
    });

    return palanquillaPlanta;
}

//Palanquilla en Perfil
public Path getPalanquillaPerfil(double anchoPalanquilla, Color color, double abcisa)
{

    double x1 = abcisa * escalaLargo + x0;
    double Ay = y0_perfil + 1.5 * altoHorno * escalaPerfil;
    double y1 = Ay - altoInferior * escalaPerfil;

    double y2 = y1 - anchoPalanquilla * escalaLargo;
    double x2 = x1 + anchoPalanquilla * escalaLargo;
    Path palanquillaPerfil = new Path();

    MoveTo moveTo = new MoveTo();
    moveTo.setX(kx * x1);
    moveTo.setY(ky * y1);

    LineTo lineTo = new LineTo();
    lineTo.setX(kx * x1);
    lineTo.setY(ky * y2);

    LineTo lineTo2 = new LineTo();
    lineTo2.setX(kx * x2);
    lineTo2.setY(ky * y2);

    LineTo lineTo3 = new LineTo();
    lineTo3.setX(kx * x2);
    lineTo3.setY(ky * y1);

    LineTo lineTo4 = new LineTo();
    lineTo4.setX(kx * x1);
    lineTo4.setY(ky * y1);

    palanquillaPerfil.getElements().addAll(moveTo, lineTo, lineTo2, lineTo3, lineTo4);

    palanquillaPerfil.setStroke(Color.BLACK);
    palanquillaPerfil.setStrokeWidth(1);
    palanquillaPerfil.setFill(color);

    return palanquillaPerfil;
}



public Group getHorno()
{
    double x1 = (x0 + (offsetInicial + distanciaCalentamiento) * escalaLargo);
```

```
        Group Horno = new Group();
        Horno.getChildren().addAll(getPerfil());
        Horno.getChildren().addAll(getPlanta());
        Horno.getChildren().addAll(
                getCuadroValores(kx * Dx_Perfil, kx * x1, Dy_Perfil * ky,
    zonasHornoCalentamientoSuperior),
                getCuadroValores(kx * Dx_Perfil, kx * x1, Ay_Perfil * ky,
    zonasHornoCalentamientoInferior),
                getCuadroValores(kx * x1, kx * Bx_Perfil, Dy_Perfil * ky,
    zonasHornoIgualacionSuperior),
                getCuadroValores(kx * x1, kx * Bx_Perfil, Ay_Perfil * ky,
    zonasHornoIgualacionInferior));
        return Horno;
    }
}
```

1.5.8 Clase Superficie

```
package hot;

/**
 *
 * Clase de una superficie discretizada perpendicular a algún eje coordenado
 *
 * @author SONIA
 */
public class Superficie
{

   /**
    * Diferenciales de longitud en X
    */
   double[] x;
   /**
    * Diferenciales de longitud en Y
    */
   double[] y;
   /**
    * Diferenciales de longitud en Z
    */
   double[] z;
   /**
    * Vector normal a la superficie
    */
   double[] normal;


   /**
    * Constructor de Superficie.
    * Uno de los vectores x, y ó z debería contener un único valor
    *
    * @param x Diferenciales de longitud en X
    * @param y Diferenciales de longitud en Y
    * @param z Diferenciales de longitud en Z
    * @param normal Vector normal a la superficie
    */
   public Superficie(double[] x,
         double[] y,
         double[] z,
         double[] normal)
   {
      this.x = x;
      this.y = y;
      this.z = z;
      this.normal = normal;
   }
}
```

1.5.9 Clase Volumen

```
package hot;


/**
 * Clase de un columen discretizado
 *
 * @author SONIA
 */

public class Volumen
{
   /**
    * Diferenciales de longitud en X
    */
   double[] x;
   /**
    * Diferenciales de longitud en Y
    */
   double[] y;
   /**
    * Diferenciales de longitud en Z
    */
   double[] z;
   /**
    * Volumen
    */
   double volumen;


   /**
    * Constructor de Volumen
    *
    * @param x Diferenciales de longitud en X
    * @param y Diferenciales de longitud en Y
    * @param z Diferenciales de longitud en Z
    */
   public Volumen(double[] x,
         double[] y,
         double[] z)
   {
      this.x = x;
      this.y = y;
      this.z = z;
      volumen =(x[x.length-1]-x[0])*(y[y.length-1]-y[0])*(z[z.length-1]-z[0]);
   }
}
```

1.5.10 Clase ZonaHorno

```
package hot;

import javafx.scene.paint.Color;

/**
 *
 * @author SONIA
 */
public class ZonaHorno
{
   public String nombre;
   //***ATRIBUTOS***
   public EtiquetaColor consigna = new EtiquetaColor(Color.BLUE);
   public EtiquetaColor consignaHOT = new EtiquetaColor(Color.PURPLE);
   public EtiquetaColor medida = new EtiquetaColor(Color.BLUE);
   public EtiquetaColor regul_salida= new EtiquetaColor(Color.LIGHTCORAL);
   public EtiquetaColor TC_regulacion= new EtiquetaColor(Color.GREEN);
   public EtiquetaColor Temp_pared1= new EtiquetaColor(Color.ORANGERED);
   public EtiquetaColor Temp_pared2 = new EtiquetaColor(Color.ORANGERED);
   public EtiquetaColor Temp_boveda1= new EtiquetaColor(Color.ORANGERED);
   public EtiquetaColor Temp_boveda2 = new EtiquetaColor(Color.ORANGERED);
   public EtiquetaColor caudalGas= new EtiquetaColor(Color.GRAY);
   public EtiquetaColor cteA = new EtiquetaColor(Color.BLUE);
   public EtiquetaColor cteB = new EtiquetaColor(Color.BLUE);

   private boolean zonaSuperior; //Zona superior= true; Zona inferior (solera) = false.

   //Constructor
   public ZonaHorno(boolean zona, String nombre){
   this.zonaSuperior=zona;
   this.nombre=nombre;
   }

   //Métodos
   public boolean isZonaSuperior()
   {
      return zonaSuperior;
   }

}
```

1.5.11 Calse ZonaRadiacion

```
package hot;

import Jama.Matrix;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import static java.lang.Math.PI;
import static java.lang.Math.abs;
import static java.lang.Math.exp;
import static java.lang.Math.pow;
import static java.lang.Math.sqrt;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * Clase que maneja los factores de intercambio directo para el cálculo de la
 * transferencia de calor radiativo
 *
 * @author SONIA
 */
public class ZonalRadiacion
{
    /**
     * Nombre de la parte del horno
     */
    final String nombre;

    /**
     * Incremento espacial
     */
    final double incremento;

    /**
     * Longitud del horno
     */
    final double longitudHorno;

    /**
     * Ancho del horno
     */
    final double anchoHorno;
```

```
/**
 * Altura de cada uan de las zonas del horno
 */
double[] altoHorno;

/**
 * Abcisa comienzo zona calentamiento
 */
final double comienzoCalentamiento;

/**
 * Abcisa comienzo zona igualación
 */
final double comienzoIgualacion;

/**
 * Anchura volumen de gas
 */
final double anchoVolumen;

/**
 * Posición termopares
 */
double[] posTC;

/**
 * Índice termopar Precalentameinto
 */
int kTC1;

/**
 * Índice termopar Calentameinto
 */
int kTC2;

/**
 * Índice termopar Igualación
 */
int kTC3;

/**
 * Matriz 4KV
 */
Matrix k4v;

/**
 * Matriz intercambio directo fuente-gas
 */
Matrix fg;

/**
 * Matriz intercambio directo gas-gas
 */
Matrix gg;
```

```
/**
 * Matriz intercambio directo gasTermopar-gas
 */
Matrix gtcg;

/**
 * Matriz intercambio directo gasTermopar-fuente
 */
Matrix gtcf;

/**
 * Matriz 4KV termopar
 */
Matrix k4vtc;

/**
 * Matriz Intercambio directo sumidero-fuente
 */
Matrix sf;

/**
 * Matriz Intercambio directo sumidero-Gas
 */
Matrix sg;


/**
 * Diámetro termnopar
 */
private final double diametroTermopar = 0.5e-3; //400um
/**
 * Constante Stefan-Boltzmann
 */
public double constante_Stefan = 5.67e-8;


/**
 * Array temperaturas aire
 */
public static double[] temperaturaAire =
{
   100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400,
   1500, 1600, 1700, 1800, 1900, 2000, 2100, 2200, 2300, 2400, 2500
};

/**
 * Array densidad aire en función de la temperatura
 */
public static double[] densidad =
{
   3.6010, 1.7684, 1.1774, 0.8826, 0.7048, 0.5879, 0.5030, 0.4405, 0.3925, 0.3524,
   0.3204, 0.2947, 0.2707, 0.2515, 0.2355, 0.2211, 0.2082, 0.1970, 0.1858, 0.1762,
   0.1682, 0.1602, 0.1538, 0.1458, 0.1394
};
```

```
/**
 * Array calor específico aire en función de la temperatura
 */
public static double[] calorEspecifico =
{
   1.0266, 1.0061, 1.0057, 1.0140, 1.0295, 1.0551, 1.0752, 1.0978, 1.1212,
   1.1417, 1.160, 1.179, 1.197, 1.214, 1.230, 1.248, 1.267, 1.287, 1.309,
   1.338, 1.372, 1.419, 1.482, 1.574, 1.688
};

/**
 * Array conductividad térmica aire en función de la temperatura
 */
public static double[] conductividadTermica =
{
   0.009246, 0.01809, 0.02624, 0.03365, 0.04038, 0.04659, 0.05230,
   0.05779, 0.06279, 0.06752, 0.0732, 0.0782, 0.0837, 0.0891, 0.0946,
   0.100, 0.105, 0.111, 0.117, 0.124, 0.131, 0.139, 0.149, 0.161, 0.175
};

/**
 * Array viscosidad aire en función de la temperatura
 */
public static double[] viscosidad =
{
   0.6924, 1.3289, 1.8462, 2.286, 2.671, 3.018, 3.332, 3.625, 3.899, 4.152, 4.44, 4.69, 4.93, 5.17,
   5.40, 5.63, 5.85, 6.07, 6.29, 6.50, 6.72, 6.93, 7.14, 7.35, 7.57
};

/**
 * Enumeración variable a interpolar
 */
public enum casoInterpolacion
{
   VISCOSIDAD, CALOR_ESPECIFICO, DENSIDAD, CONDUCTIVIDAD
};

/**
 * Método que discretiza un segmento
 *
 * @param desde Valor inicial del segmento
 * @param hasta Valor final del segmento
 * @param incrementa Diferencial de longitud
 * @return Array con segmento discretizado
 */
static public double[] Discretiza(double desde, double hasta, double incrementa)
{
   if (desde == hasta)
   {
      return new double[]
      {
         desde
      };
   }
   int n = (int) ((hasta - desde) / incrementa + 0.5);
```

```
    double[] v = new double[n];
    for (int i = 0; i < n; i++)
    {
        v[i] = desde + i * incrementa;
    }
    return v;
}

/**
 * Método realiza el producto escalar de dos vectores
 *
 * @param v1 Coordenadas vector 1
 * @param v2 Coordenadas vector 2    *
 * @return Producto escalar
 */
static public Double productoEscalar(double[] v1, double[] v2)
{
    if (v1.length != v2.length)
    {
        return null;
    }
    double producto = 0;
    for (int i = 0; i < v1.length; i++)
    {
        producto += v1[i] * v2[i];
    }
    return producto;
}

/**
 * Método calcula el módulo de un vector
 *
 * @param v Coordenadas vector
 * @return Módulo del vector
 */
static public double Modulo(double[] v)
{
    double modulo = 0;
    for (double c : v)
    {
        modulo += pow(c, 2);
    }
    return sqrt(modulo);
}

 /**
 * Método que calcula el factor de intercambio directo superficie-superficie
 *
 * @param desde Superficie 1
 * @param hasta Superficie 2
 * @param incremento Diferencial longitud
 * @param absorcion Coeficiente de absorción térmica del gas
 * @return Factor de intercambio directo superficie-superficie
 */
```

```
static public double Superficie2Superficie(Superficie desde, Superficie hasta, double incremento,
double absorcion)
        {
           double ss = 0;
           double[] v1 = new double[3];
           double superficie = 0;

           for (double x0 : desde.x)
           {
              for (double y0 : desde.y)
              {
                 for (double z0 : desde.z)
                 {
                    for (double x1 : hasta.x)
                    {
                       for (double y1 : hasta.y)
                       {
                          for (double z1 : hasta.z)
                          {
                             double xi = x0;
                             double yi = y0;
                             double zi = z0;
                             double xf = x1;
                             double yf = y1;
                             double zf = z1;
                             if (desde.x.length == 1)
                             {
                                yi += incremento / 2;
                                zi += incremento / 2;
                             }
                             else if (desde.y.length == 1)
                             {
                                xi += incremento / 2;
                                zi += incremento / 2;
                             }
                             else if (desde.z.length == 1)
                             {
                                xi += incremento / 2;
                                yi += incremento / 2;
                             }
                             if (hasta.x.length == 1)
                             {
                                yf += incremento / 2;
                                zf += incremento / 2;
                             }
                             else if (hasta.y.length == 1)
                             {
                                xf += incremento / 2;
                                zf += incremento / 2;
                             }
                             else if (hasta.z.length == 1)
                             {
                                xf += incremento / 2;
                                yf += incremento / 2;
                             }
```

```
                        v1[0] = xf - xi;
                        v1[1] = yf - yi;
                        v1[2] = zf - zi;

                        double pe1 = abs(productoEscalar(v1, desde.normal));
                        double pe2 = abs(productoEscalar(v1, hasta.normal));
                        double S = Modulo(v1);
                        if (S != 0)
                        {
                           ss += pe1 * pe2 * exp(-absorcion * S) * pow(incremento, 4) / (PI * pow(S,
4));
                        }
                     }
                  }
               }
            }
         }
         return ss;
      }

      /**
       * Método que calcula el factor de intercambio directo superficie-volumen
       *
       * @param desde Superficie
       * @param hasta Volumen
       * @param incremento Diferencial longitud
       * @param absorcion Coeficiente de absorción térmica del gas
       * @return Factor de intercambio directo superficie-volumen
       */
      static public double Superficie2Volumen(Superficie desde, Volumen hasta, double incremento,
double absorcion)
      {
         double ss = 0;
         double[] v1 = new double[3];
         double superficie = 0;

         for (double x0 : desde.x)
         {
            for (double y0 : desde.y)
            {
               for (double z0 : desde.z)
               {
                  for (double x1 : hasta.x)
                  {
                     for (double y1 : hasta.y)
                     {
                        for (double z1 : hasta.z)
                        {
                           double xi = x0;
                           double yi = y0;
                           double zi = z0;
                           double xf = x1;
                           double yf = y1;
```

```
                        double zf = z1;
                        if (desde.x.length == 1)
                        {
                           yi += incremento / 2;
                           zi += incremento / 2;
                        }
                        else if (desde.y.length == 1)
                        {
                           xi += incremento / 2;
                           zi += incremento / 2;
                        }
                        else if (desde.z.length == 1)
                        {
                           xi += incremento / 2;
                           yi += incremento / 2;
                        }

                        xf += incremento / 2;
                        yf += incremento / 2;
                        zf += incremento / 2;

                        v1[0] = xf - xi;
                        v1[1] = yf - yi;
                        v1[2] = zf - zi;

                        double pe1 = abs(productoEscalar(v1, desde.normal));

                        double S = Modulo(v1);
                        if (S != 0)
                        {
                           ss += pe1 * absorcion * exp(-absorcion * S) * pow(incremento, 5) / (PI *
pow(S, 3));
                        }
                     }
                  }
               }
            }
         }
         return ss;
      }

      /**
       * Método que calcula el factor de intercambio directo volumen-volumen
       *
       * @param desde Volumen 1
       * @param hasta Volumen 2
       * @param incremento Diferencial longitud
       * @param absorcion Coeficiente de absorción térmica del gas
       * @return Factor de intercambio directo volumen-volumen
       */
      static public double Volumen2Volumen(Volumen desde, Volumen hasta, double incremento,
double absorcion)
      {
         double ss = 0;
```

```
    double[] v1 = new double[3];
    double superficie = 0;

    for (double x0 : desde.x)
    {
       for (double y0 : desde.y)
       {
          for (double z0 : desde.z)
          {
             for (double x1 : hasta.x)
             {
                for (double y1 : hasta.y)
                {
                   for (double z1 : hasta.z)
                   {
                      double xi = x0;
                      double yi = y0;
                      double zi = z0;
                      double xf = x1;
                      double yf = y1;
                      double zf = z1;

                      xi += incremento / 2;
                      yi += incremento / 2;
                      zi += incremento / 2;

                      xf += incremento / 2;
                      yf += incremento / 2;
                      zf += incremento / 2;

                      v1[0] = xf - xi;
                      v1[1] = yf - yi;
                      v1[2] = zf - zi;

                      double S = Modulo(v1);
                      if (S != 0)
                      {
                            ss += pow(absorcion, 2) * exp(-absorcion * S) * pow(incremento, 6) / (PI
                            * pow(S, 2));
                      }
                   }
                }
             }
          }
       }
    }
    return ss;
}

/**
 * Método que guarda en fichero una matriz
 *
 * @param nombreFichero Nombre fichero
 * @param array Matriz
 */
```

```
static void Salva(String nombreFichero, double[][] array) throws IOException
{
   File temp;
   PrintWriter pw;
   FileWriter fichero;

   temp = new File(System.getProperty("user.dir") + "\\" + nombreFichero);
   fichero = new FileWriter(temp);
   pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(new
   FileOutputStream(temp), "ISO8859_1")));

   for (int i = 0; i < array.length; i++)
   {
      for (int j = 0; j < array[i].length; j++)
      {
         pw.print(String.valueOf(array[i][j]) + ";");
         pw.flush();
      }
      pw.println("");
      pw.flush();
   }
   pw.close();
}



/**
 * Método que guarda en ficheros las matrices de intercambio directo
 *
 * @throws java.io.IOException
 */
public void generaArchivos() throws IOException
{
   double absortividad = 0.267;

   Superficie[] fuente = new Superficie[3];

   fuente[0] = new Superficie(Discretiza(0.0, comienzoCalentamiento, incremento),
   Discretiza(0.0, anchoHorno, incremento), Discretiza(altoHorno[0], altoHorno[0], incremento),
   Normalizar(new double[]
   {
      0, 0, -1
   }));
   fuente[1] = new Superficie(Discretiza(comienzoCalentamiento, comienzoIgualacion,
   incremento), Discretiza(0.0, anchoHorno, incremento), Discretiza(altoHorno[1], altoHorno[1],
   incremento), Normalizar(new double[]
   {
      0, 0, -1
   }));
   fuente[2] = new Superficie(Discretiza(comienzoIgualacion, longitudHorno, incremento),
   Discretiza(0.0, anchoHorno, incremento), Discretiza(altoHorno[2], altoHorno[2], incremento),
   Normalizar(new double[]
   {
      0, 0, -1
   }));
```

```
int n = (int) (longitudHorno / anchoVolumen);

Volumen[] volumenes = new Volumen[n];
for (int i = 0; i < volumenes.length; i++)
{
   if (i * anchoVolumen < comienzoCalentamiento)
   {
      volumenes[i] = new Volumen(Discretiza(i * anchoVolumen, (i + 1) * anchoVolumen,
incremento), Discretiza(0, anchoHorno, incremento), Discretiza(0, altoHorno[0],
incremento));
   }
   else if (i * anchoVolumen < comienzoIgualacion)
   {
      volumenes[i] = new Volumen(Discretiza(i * anchoVolumen, (i + 1) * anchoVolumen,
incremento), Discretiza(0, anchoHorno, incremento), Discretiza(0, altoHorno[1],
incremento));
   }
   else
   {
      volumenes[i] = new Volumen(Discretiza(i * anchoVolumen, (i + 1) * anchoVolumen,
incremento), Discretiza(0, anchoHorno, incremento), Discretiza(0, altoHorno[2],
incremento));
   }
}

Superficie[] sumidero = new Superficie[n];
for (int i = 0; i < sumidero.length; i++)
{
   sumidero[i] = new Superficie(Discretiza(i * anchoVolumen, (i + 1) * anchoVolumen,
incremento), Discretiza(0, anchoHorno, incremento), Discretiza(0, 0, incremento),
Normalizar(new double[]
   {
      0, 0, 1
   }));
}

File temp;
PrintWriter pw;
FileWriter fichero;

temp = new File(System.getProperty("user.dir") + "\\" + "sg_" + nombre + ".dat");
fichero = new FileWriter(temp);
pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(temp), "ISO8859_1")));

double[][] sg_arriba = new double[n][n];
for (int i = 0; i < sumidero.length; i++)
{
   for (int j = 0; j < volumenes.length; j++)
   {
      System.out.println(LocalDateTime.now().toString() + ", i=" + String.valueOf(i) + ", " +
"j=" + String.valueOf(j));
      sg_arriba[i][j] = ZonalRadiacion.Superficie2Volumen(sumidero[i], volumenes[j],
incremento, absortividad);
```

```
            pw.print(String.valueOf(sg_arriba[i][j]) + ";");
            pw.flush();
        }
        pw.println("");
        pw.flush();
    }
    pw.close();

    temp = new File(System.getProperty("user.dir") + "\\" + "sf_" + nombre + ".dat");
    fichero = new FileWriter(temp);
    pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(new
    FileOutputStream(temp), "ISO8859_1")));

    double[][] sf_arriba = new double[n][3];
    for (int i = 0; i < sumidero.length; i++)
    {
        for (int j = 0; j < fuente.length; j++)
        {
            System.out.println(LocalDateTime.now().toString() + ", i=" + String.valueOf(i) + ", " +
    "j=" + String.valueOf(j));
            sf_arriba[i][j] = ZonalRadiacion.Superficie2Superficie(sumidero[i], fuente[j],
    incremento, absortividad);

            pw.print(String.valueOf(sf_arriba[i][j]) + ";");
            pw.flush();
        }
        pw.println("");
        pw.flush();
    }
    pw.close();

    temp = new File(System.getProperty("user.dir") + "\\" + "k4v_" + nombre + ".dat");
    fichero = new FileWriter(temp);
    pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(new
    FileOutputStream(temp), "ISO8859_1")));

    double[][] k4v_arriba = new double[n][n];
    for (int i = 0; i < volumenes.length; i++)
    {
        for (int j = 0; j < volumenes.length; j++)
        {
            System.out.println(LocalDateTime.now().toString() + ", i=" + String.valueOf(i) + ", " +
    "j=" + String.valueOf(j));
            if (i == j)
            {
                k4v_arriba[i][j] = 4 * absortividad * volumenes[i].volumen;
            }
            else
            {
                k4v_arriba[i][j] = 0;
            }
            pw.print(String.valueOf(k4v_arriba[i][j]) + ";");
            pw.flush();
        }
```

```
        pw.println("");
                pw.flush();
            }
            pw.close();

            temp = new File(System.getProperty("user.dir") + "\\" + "fg_" + nombre + ".dat");
            fichero = new FileWriter(temp);
            pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(temp), "ISO8859_1")));

            double[][] fg_arriba = new double[n][3];
            for (int i = 0; i < volumenes.length; i++)
            {
               for (int j = 0; j < fuente.length; j++)
               {
                  System.out.println(LocalDateTime.now().toString() + ", i=" + String.valueOf(i)
+ ", " + "j=" + String.valueOf(j));
                  fg_arriba[i][j] = ZonalRadiacion.Superficie2Volumen(fuente[j], volumenes[i],
incremento, absortividad);
                  pw.print(String.valueOf(fg_arriba[i][j]) + ";");
                  pw.flush();
               }
               pw.println("");
               pw.flush();
            }
            pw.close();

            temp = new File(System.getProperty("user.dir") + "\\" + "gg_" + nombre + ".dat");
            fichero = new FileWriter(temp);
            pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(temp), "ISO8859_1")));

            double[][] gg_arriba = new double[n][n];
            for (int i = 0; i < volumenes.length; i++)
            {
               for (int j = 0; j < volumenes.length; j++)
               {
                  System.out.println(LocalDateTime.now().toString() + ", i=" + String.valueOf(i)
+ ", " + "j=" + String.valueOf(j));
                  gg_arriba[i][j] = ZonalRadiacion.Volumen2Volumen(volumenes[i],
volumenes[j], incremento, absortividad);
                  pw.print(String.valueOf(gg_arriba[i][j]) + ";");
            pw.flush();
         }
         pw.println("");
         pw.flush();
      }

      pw.close();

   }

   /**
    * Método que carga desde ficheros las matrices de intercambio directo
    *
```

```
 */
public void Carga()
{
   double[][] k4vArray = Carga("k4v_" + nombre + ".dat");;
   double[][] fgArray = Carga("fg_" + nombre + ".dat");
   double[][] ggArray = Carga("gg_" + nombre + ".dat");

   double[][] sfArray = Carga("sf_" + nombre + ".dat");
   double[][] sgArray = Carga("sg_" + nombre + ".dat");

   k4v = new Matrix(k4vArray);
   fg = new Matrix(fgArray);
   gg = new Matrix(ggArray);

   sf = new Matrix(sfArray);
   sg = new Matrix(sgArray);

   double[][] gtcgArray = new double[3][ggArray.length];
   double[][] gtcfArray = new double[3][3];
   double[][] k4vtcArray = new double[3][3];
   int[] indiceTermopar = new int[]
   {
      kTC1, kTC2, kTC3
   };
   for (int i = 0; i < indiceTermopar.length; i++)
   {
      for (int j = 0; j < indiceTermopar.length; j++)
      {
         if (i == j)
         {
            k4vtcArray[i][j] = k4vArray[indiceTermopar[i]][indiceTermopar[i]];
         }
         else
         {
            k4vtcArray[i][j] = 0;
         }
      }

   }
   for (int i = 0; i < indiceTermopar.length; i++)
   {
      for (int j = 0; j < ggArray.length; j++)
      {
         gtcgArray[i][j] = ggArray[indiceTermopar[i]][j];
      }
   }
   for (int i = 0; i < indiceTermopar.length; i++)
   {
      for (int j = 0; j < indiceTermopar.length; j++)
      {
         gtcfArray[i][j] = fgArray[indiceTermopar[i]][j];
      }
   }
   gtcg = new Matrix(gtcgArray);
   gtcf = new Matrix(gtcfArray);
```

```
        k4vtc = new Matrix(k4vtcArray);
    }

    /**
     * Método que obtiene un parámetro térmico del aire interpolando
     * con la temperatura
     *
     * @param temperatura Temperatura
     * @param caso      Variable a a interpolar
     * @return Valor interpolado
     */
    public double Interpola(double temperatura, casoInterpolacion caso)
    {

        double valorInterpolado = 0.0;

        if (temperatura < temperaturaAire[0])
        {
            temperatura = temperaturaAire[0];
        }

        if (temperatura > temperaturaAire[temperaturaAire.length - 1])
        {
            temperatura = temperaturaAire[temperaturaAire.length - 1];
        }

        int indice = 0;

        try
        {
            while (indice < temperaturaAire.length - 2
                    && !(temperatura >= temperaturaAire[indice] && temperatura <
        temperaturaAire[indice + 1]))
            {
                indice++;
            }
        }
        catch (ArrayIndexOutOfBoundsException ex)
        {
            int h = 01;
        }

        switch (caso)
        {
            case VISCOSIDAD:
                valorInterpolado = ((viscosidad[indice + 1] - viscosidad[indice]) /
          (temperaturaAire[indice + 1] - temperaturaAire[indice])) * (temperatura -
          temperaturaAire[indice]) + viscosidad[indice];
                valorInterpolado *= 1e5;
                break;
            case CALOR_ESPECIFICO:
                valorInterpolado = ((calorEspecifico[indice + 1] - calorEspecifico[indice]) /
          (temperaturaAire[indice + 1] - temperaturaAire[indice])) * (temperatura -
          temperaturaAire[indice]) + calorEspecifico[indice];
                break;
```

```
        case DENSIDAD:
            valorInterpolado = ((densidad[indice + 1] - densidad[indice]) / (temperaturaAire[indice +
 1] - temperaturaAire[indice])) * (temperatura - temperaturaAire[indice]) + densidad[indice];
            break;
        case CONDUCTIVIDAD:
            valorInterpolado = ((conductividadTermica[indice + 1] - conductividadTermica[indice]) /
           (temperaturaAire[indice + 1] - temperaturaAire[indice])) * (temperatura -
           temperaturaAire[indice]) + conductividadTermica[indice];
            break;

        default:
            valorInterpolado = -1; //Error en la interpolacion
            break;
    }

    return valorInterpolado;

}

/**
 * Método que obtiene la viscosidad del aire con la temperatura
 *
 * @param temperatura Temperatura
 * @return Viscosidad del aire
 */
public double getViscosidadAire(double temperatura)
{
    return Interpola(temperatura, casoInterpolacion.VISCOSIDAD);
}

 /**
 * Método que obtiene la conductividad térmica del aire con la temperatura
 *
 * @param temperatura Temperatura
 * @return Conductividad térmica del aire
 */
public double getConductividadAire(double temperatura)
{
    return Interpola(temperatura, casoInterpolacion.CONDUCTIVIDAD);
}

 /**
 * Método que obtiene la densidad del aire con la temperatura
 *
 * @param temperatura Temperatura
 * @return Densidad del aire
 */
public double getDensidadAire(double temperatura)
{
    return Interpola(temperatura, casoInterpolacion.DENSIDAD);
}

 /**
 * Método que obtiene el aclor específico del aire con la temperatura
 *
```

```
 * @param temperatura Temperatura
 * @return Calor específico del aire
 */
public double getCalorEspecificoAire(double temperatura)
{
   return Interpola(temperatura, casoInterpolacion.CALOR_ESPECIFICO);
}

 /**
 * Método que corrige las medidas de temperatura de los termopares
 *
 * @param TemperaturaTermopar Temperaturas termopar
 * @param caudal caudal de aire que pasa por el termopar
 * @return Temperaturas del gas
 */
public double[] TemperaturaGas(double[] TemperaturaTermopar, double[] caudal)
{
   int num_zonas = TemperaturaTermopar.length;
   double[] temperaturaGas = new double[num_zonas];
   double[] temperaturaGasTemporal = new double[num_zonas];
   for (double temperatura : TemperaturaTermopar)
   {
      temperatura += 273;
   }

   Arrays.fill(temperaturaGas, 0);
   Arrays.fill(temperaturaGasTemporal, 273);

   double numeroPrandtl = 0.0, numeroReynolds = 0.0, numeroNusselt = 0.0;
   double velocidadGas = 0.0, h = 0.0;

   for (int i = 0; i < num_zonas; i++)
   {
      while (true)
      {
         numeroPrandtl = getCalorEspecificoAire(temperaturaGasTemporal[i]) *
         getViscosidadAire(temperaturaGasTemporal[i]) /
         getConductividadAire(temperaturaGasTemporal[i]);
         velocidadGas = caudal[i] * getDensidadAire(273) / (7.9 * 1.480 *
         getDensidadAire(temperaturaGasTemporal[i]));
         numeroReynolds = getDensidadAire(temperaturaGasTemporal[i]) * velocidadGas *
         diametroTermopar / getViscosidadAire(temperaturaGasTemporal[i]);
         numeroNusselt = 2 + 0.6 * Math.pow(numeroReynolds, 1.0 / 2) *
         Math.pow(numeroPrandtl, 1.0 / 3);
         h = numeroNusselt * getConductividadAire(temperaturaGasTemporal[i]) /
         diametroTermopar;
         double tempGas = TemperaturaTermopar[i] + 0.11 * constante_Stefan *
         Math.pow(TemperaturaTermopar[i], 4) / h;

         if (abs(tempGas - temperaturaGasTemporal[i]) < 0.0001)
         {
            temperaturaGas[i] = tempGas;
            break;
         }
         temperaturaGasTemporal[i] = tempGas;
```

```
            }
        }

        for (double temperatura : temperaturaGas)
        {
            temperatura -= 273;
        }

        return temperaturaGas;
    }

    /**
     * Método que calcula una temperatura ambiente ficticia que produciría
     * el mismo flujo radiativo que el horno
     *
     * @param abcisa Abcisa del horno
     * @param EgtcArray Array de potencias radiativas medidas de gas
     * @return Temperatura ambiente ficticia en la abcisa
     */
    public double TemperaturaAmbiente(double abcisa, double[] EgtcArray)
    {
        int k = (int) (abcisa / anchoVolumen);
        Matrix[] t = getEg_J(EgtcArray);
        Matrix Eg = t[0];
        Matrix J = t[1];
        Matrix sg_i = new Matrix(1, Eg.getRowDimension());
        Matrix sf_i = new Matrix(1, J.getRowDimension());

        for (int i = 0; i < J.getRowDimension(); i++)
        {
            sf_i.set(0, i, sf.get(k, i));
        }

        for (int i = 0; i < Eg.getRowDimension(); i++)
        {
            sg_i.set(0, i, sg.get(k, i));
        }

        double E = ((sf_i.times(J)).plus(sg_i.times(Eg))).get(0, 0);

        return pow(E / (constante_Stefan * anchoVolumen * anchoHorno), 0.25) - 273;
    }

    /**
     * Método que calcula una temperatura ambiente ficticia que produciría
     * el mismo flujo radiativo que el horno
     *
     * @param abcisa Abcisa del horno
     * @param Eg Array de potencias radiativas del conjunto de volúmenes de gas
     * @param J Array de radiosidades de las paredes refractarias
     * @return Temperatura ambiente ficticia en la abcisa
     */
    public double TemperaturaAmbiente(double abcisa, Matrix Eg, Matrix J)
    {
        int k = (int) (abcisa / anchoVolumen);
```

```
    Matrix sg_i = new Matrix(1, Eg.getRowDimension());
    Matrix sf_i = new Matrix(1, J.getRowDimension());

    for (int i = 0; i < J.getRowDimension(); i++)
    {
       sf_i.set(0, i, sf.get(k, i));
    }

    for (int i = 0; i < Eg.getRowDimension(); i++)
    {
       sg_i.set(0, i, sg.get(k, i));
    }

    double E = ((sf_i.times(J)).plus(sg_i.times(Eg))).get(0, 0);

    return pow(E / (constante_Stefan * anchoVolumen * anchoHorno), 0.25) - 273;
}

/**
 * Método que calcula la distribución de potencias radiativas del gas
 * y las radiosidades de las paredes refractarias a partir de las potencias
 * radiativas medidas de gas
 *
 * @param EgArray Array de potencias radiativas medidas de gas
 * @return Array con las matrices de Potencias radiativas del gas y radiosidades refractario
 */
public Matrix[] getEg_J(double[] EgArray)
{
    Matrix Egtc;

    Egtc = new Matrix(EgArray, EgArray.length);

    Matrix t = k4v.minus(gg).inverse();
    Matrix J = (gtcf.plus(gtcg.times(t).times(fg))).inverse().times(k4vtc).times(Egtc);
    Matrix Eg = ((k4v.minus(gg)).inverse()).times(fg).times(J);
    Matrix r = k4vtc.inverse().times((gtcf.plus(gtcg.times(t).times(fg))).times(J));

    return new Matrix[]
    {
       Eg, J
    };

}

/**
 * Método que obtiene una array bidimensional de una matriz a partir de un fichero
 * @param nombreFichero Nombre del fichero que almacena una matriz
 * @return array de la matriz
 */
static public double[][] Carga(String nombreFichero)
{
    List<double[]> lista = new ArrayList<>();

    try
    {
```

```
            BufferedReader in = new BufferedReader(new FileReader(System.getProperty("user.dir") +
"\\" + nombreFichero));
            String linea;
            try
            {
               while ((linea = in.readLine()) != null)
               {
                  String[] tokens = linea.split(";");
                  double[] valorLista = new double[tokens.length];
                  for (int i = 0; i < tokens.length; i++)
                  {
                     valorLista[i] = Double.valueOf(tokens[i]);
                  }
                  lista.add(valorLista);
               }
            }
            finally
            {
               in.close();

               double[][] a = new double[lista.size()][];
               for (int i = 0; i < lista.size(); i++)
               {
                  a[i] = new double[lista.get(i).length];
                  for (int j = 0; j < lista.get(i).length; j++)
                  {
                     a[i][j] = lista.get(i)[j];
                  }
               }
               return a;
            }

         }
         catch (FileNotFoundException ex)
         {
            //info.Registra(ex);
         }
         catch (IOException ex)
         {
            //info.Registra(ex);
         }
         return null;
      }

      /**
       * Método que genera una array de potencias de radiación a partir
       * de las temperaturas en ºC
       * @param TempTC Array con las temperaturas en ºC
       * @return Array de potencias radiativas
       */
      public double[] getEgTc(double[] TempTC)
      {
         double[] EgTc = new double[TempTC.length];
         for (int i = 0; i < TempTC.length; i++)
         {
```

```
            EgTc[i] = constante_Stefan * pow(TempTC[i] + 273, 4);
        }

        return EgTc;

    }

/**
     * Método que normaliza un vector
     * @param v Vector
     * @return Vector normalizado
     */
    static public double[] Normalizar(double[] v)
    {
        double[] n = new double[v.length];
        double modulo = Modulo(v);
        for (int i = 0; i < v.length; i++)
        {
            n[i] = v[i] / modulo;
        }

        return n;
    }

    /**
     * Constructor de la clase que maneja los factores de intercambio directo
     * para el cálculo de la transferencia de calor radiativo
     *
     * @param nombre Nombre de la parte del horno
     * @param comienzoZona Array de abcisas de comienzo de zona
     * @param altura Array de alturas de zona
     * @param posTermopar Array de posiciones de termopar
     * @param longitudHorno Longitud del horno
     * @param anchoHorno Ancho del horno
     * @param incremento Diferencial de longitud
     * @param anchoVolumen  Ancho del volumen de gas
     */
    public ZonalRadiacion(String nombre, double[] comienzoZona, double[] altura, double[]
     posTermopar,
         double longitudHorno, double anchoHorno,
         double incremento, double anchoVolumen)
  {
     this.incremento = incremento;
     this.longitudHorno = longitudHorno;
     this.anchoHorno = anchoHorno;
     altoHorno = altura;
     comienzoCalentamiento = comienzoZona[1];
     comienzoIgualacion = comienzoZona[2];
     this.anchoVolumen = anchoVolumen;
     posTC = posTermopar;
     kTC1 = (int) (posTC[0] / anchoVolumen);
     kTC2 = (int) (posTC[1] / anchoVolumen);
     kTC3 = (int) (posTC[2] / anchoVolumen);
     this.nombre = nombre;
    }
```

```
    public static void main(String[] args) throws IOException
    {
       ZonalRadiacion zonaSuperior = new ZonalRadiacion("up",
             new double[]
             {
                0, 4.96, 4.96 + 4.8125
             },
             new double[]
             {
                1, 1.4, 1.4
             },
             new double[]
             {
                3.345, 9.675, 13.318
             },
             14.56, 7.9, 0.1, 0.26);

       ZonalRadiacion zonaInferior = new ZonalRadiacion("down",
             new double[]
             {
                0, 4.96, 4.96 + 4.8125
             },
             new double[]
             {
                1.820, 1.820, 1.820
             },
             new double[]
             {
                3.4, 8.68, 12.56
             },
             14.56, 7.9, 0.1, 0.26);

      // zonaSuperior.generaArchivos();
       // zonaInferior.generaArchivos();
    }
}
```