Introduction to Programming

© Martin Gonzalez-Rodriguez, PhD.

www.martin-gonzalez.es

Module 1

Programming Essentials

Lecture 1

The Person Class

"Writing code has a place in the human hierarchy worth somewhere above grave robbing and beneath managing"

Gerald Weinberg

The Person Class

Design a class to model a Person. It must contain information regarding his/her:

- Name.
- Surname.
- Age.
- Gender.

Include comments for every method. They must be compatible with JavaDoc.

The Person Class

Use the following data types and classes for the fields:

```
public String name;
public String surname;
public int age;
public boolean gender;
```

Add methods to modify the value of the fields.

```
// Example
public void changeName (String newName)
```

Add methods to obtain the value of the fields.

```
// Example
public String giveMeYourName()
```

The Person Class

Methods that change the value of a field are called setters

```
// Example
public void setName (String newName)
```

Methods that return the value of a field are called getters

```
// Example
public String getName()
```

The Person Class

Documenting a class is as important as its implementation.

There are three types of comments in Java

```
// Single statement comment.

/* Multiple statement
comment */

/**
* JavaDoc comment
*/
```

JavaDoc commands

The @param and @return JavaDoc commands

1. @param is used to document each parameter for a given method.

2. @return is used to document the method's return value.

JavaDoc commands

Example

```
/**
 * Description of the method.
 *
 * @param parameter1 Description of the first parameter.
 * @param parameter2 Description of the second parameter.
 * @return Description of the return value.
 */

public ReturnType methodName (ParameterType parameter1,
 ParameterType parameter 2)
{
    // magic;
}
```

HOMEWORK

Complete the Person class to include:

1. Name, Surname, Gender and Age fields.
2. Setters and Getters for all these fields.
3. The Person class and all its methods must include descriptive comments compatible with JavaDoc.

Lecture 2

Conditions and Constants

"First, solve the problem. Then, write the code"

John Johnson

Development approach

Programmers implement their methods just after completing their documentation and tests.

1. Document the method.
2. Design the test units.
3. Develop the method.

Example. Create a Math class and develop a method to return the area of a triangle

```
/**
* Computes the area of a triangle from its base and height
* …
*/
public ... computeArea (...)
```

The 'this' operator

Who does 'name' refer to in this setter?

```
private String name;

public void setName (String name)
 {
  name = name;
 }
```

The If command

Modify the setAge(int age) method to exclude negative values in 'age'.

Exceptions

Throw a RuntimeException if the setAge is not invoked properly.

```
public void setAge (int age)
```

Throwing Exceptions

Use auxiliary methods to deal with exceptions.

```
public void checkParam(boolean condition, String message) {
 if (!condition)
    throw new RuntimeException(message);
}

public void setAge (int age)
{
    checkParam (age >=0, "The age can not be negative");

    this.age = age;
}
```

Semantics

Implement the following method:

```
public boolean isAGirl();
```

Represent the gender using constants like:

```
// We define that...
// gender == false; -> girl!
// gender == true; -> boy!
```

Constants

Classic constant definition.

```
public final static boolean FEMALE_VALUE = false;
public final static boolean MALE_VALUE = true;
```

What does 'static' mean?

Static properties

Name declared as an object's field.

```
public String name;

Person a = new Person();
Person b = new Person();

a.setName ("Peter");
b.setName ("Mary");

// compare the name obtained in both person objects…
// Is it the same?
a.getName();
b.getName();
```

Static properties

Name declared as a Class field.

```
public static String name; // <- STATIC field!

Person a = new Person();
Person b = new Person();

a.setName (“Peter”);
b.setName (“Mary”);

// compare the name obtained in both person objects…
// Is it the same?
a.getName();
b.getName();
```

HOMEWORK

Modify the setAge() method to accept values in the range [0..118].

a. Define 118 as the value of the MAX_AGE_VALUE constant.

b. If the value provided is out of range, the age will not be updated.

Lecture 3

Data Output

“Optimism is an occupational hazard of programming; feedback is the treatment”

Kent Beck

The print() Method

Implement the print() method in the Class Person to obtain an output similar to the one used below.

```
public void print()


Name: Yuri
Surname: Gagarin
Age: 34
Gender: false.
```

The print() Method

Modify the print() method to use text when displaying info about the gender.

```
public void print()


Name: Yuri
Surname: Gagarin
Age: 34
Gender: male.
```

The toString() Method

Develop a method called toString() to return a String that contains the values of the fields in the Person class.

```
public String toString()
```

```
[Name: Yuri - Surname: Gagarin - Age: 34 - Gender: male]
```

The getHashCode() Method

Develop a getHashCode() method[1] to return a String containing the value of every field in Person.

1. Fields are separated by dashes.
2. Name and Surname are written in uppercase.

```
public String getHashCode()
```

```
YURI-GAGARIN-34-false
```

[1] Every class in Java owns a public *int hashCode()* method.

The HashCode() Method

Modify the HashCode() to use only the first 3 letters of each String field.

```
public String getHashCode()
```

```
YUR-GAG-34-false
```

HOMEWORK

1. Develop a public int getCriticalAge() to return the number of years until adulthood.

   a. If the person has reached adulthood, the method returns the number of years <u>until</u> retirement.

   b. If the person has reached retirement, the method returns the number of years <u>after</u> retirement.

   c. An ADULTHOOD_AGE and RETIREMENT_AGE constants must be defined.

Lecture 4

Software Quality

“I don’t care if it works on your machine! We are not shipping your machine!”

Vidiu Platon

The Test Class

Create a Test class to test the gender related methods in the Person class

```
/**
* Returns true if the basic gender related methods in the
* Person class pass a series of tests (setGender, getGender).
*
* @return boolean True passes the tests, false otherwise
*/
public boolean testGender()
{
  //Set true as the value for the gender field.
  //Retrieve the value for the gender field and check that
 // it is actually true.
}
```

The Test Class

Add a second test to the Test class to test the isAGirl method

```
/**
* Returns true if the isAGirl passes a series of tests
*
* @return boolean True pass the tests, false otherwise
*/
public boolean testIsAGirl()
{
  //Sets different values for the gender field.
  //Verifies that the isAGirl method returns the right values

 // test the isAGirl() method using the MALE_VALUE and
 //FEMALE_VALUE constants.
 }
```

The Test Class

Add a new method to launch every testing method in the class

```
/**
* Returns true if ALL the test methods return true. False
* if at least one method returns false.
*
* @return True when all the testing methods return true.
* otherwise it returns false
*/
public boolean testAll()
{
}
```

The Java Unit Test Framework

Automates the testing process replicating previous defined tests.

It is based on the *assertEquals* command.

Testing infinite set of values

Add a testAge method to the JUnit to perform positive and negative test on the age field.

Positive tests: [0..118]

Negative test: (-∞, -1] and [119, ∞)

The Test Class

Testing methods that throw exceptions

```
dummy.setAge(0);

try // Handle exception
{
    dummy.setAge(-1); // wrong parameter!
    fail(); // makes sure the exception is thrown!
}
catch (Exception e)
{
  assertEquals(0, dummy.getAge());
   assertEquals("The age can not be negative",
    e.getMessage()); // The error must be the expected one!
}
```

The Test Class

Add this code to the header of your test file to test the methods in alphabetical order

```
import org.junit.runners.MethodSorters;
import org.junit.FixMethodOrder;
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
```

Naming convention of the test methods: <name of the method/field><name of the test>

```
nameBasic;  // positive (“Luke”) - negative test (null)
nameSpecialChars; // eg. (@)
```

Do not use the word “test” as part of the name of a testing methods

HOMEWORK

1. Design exhaustive JUnit tests for all the methods designed so far.

2. Pay special attention to the getCriticalAge() method.

    a. Include positive and negative tests for every subset of feasible age values for Age.

Lecture 5

Comparing & Constructors

"Reusability is key in reducing bugs and coding quickly"

Robert Duchnik

Comparing in Java

Write a method areYou(int age) to return true if the age of the user is the same as the one provided.

```
public boolean areYou (int age)
```

Comparing in Java

Write a method areYou(String name) to return true if the name of the user is the same as the one provided.

```
public boolean areYou(String name)
```

Comparing in Java

Write a method areYou(Person person) that return true if the person is the same as the one provided.

a. A person is the same if the values of name, surname, age and gender are the same.

```
public boolean areYou(Person person)
```

Comparing in Java

Write a method int compareTo(Person person) to compare the age of the person with the one of the parameter.

1. Returns 0 if the age of both persons is the same.
2. Returns -1 if the host person is younger than the person provided as the parameter.
3. Returns +1 if the host person is older than the one provided as the parameter.

```
public int compareTo(Person person)
```

Constructors

Modify the default constructor to set default values for all the fields in Person

```
public Person()


// Default values
Name: Yuri
Surname: Gagarin
Age: 34
Gender: male.
```

Constructors

Add a second constructor to set up a different default age.

```
public Person(int age)

//it can use any of the constructors defined previously
```

Constructors

Add a third constructor to set up values for the name and surname of Person.

```
public Person(String name, String surname)

//it can also use any of the constructors defined previously
```

Constructors

Add a fourth constructor to set up values for all the fields in Person.

```
public Person(String name, String surname, int age, boolean
 gender)

//it can also use any of the constructors defined previously
```

HOMEWORK [1/3]

Create an Airplane class to model aircrafts in a Air Controller simulator.

a. Airplanes flight in a game board of dimensions [X_WEST_BORDER, X_EAST_BORDER] and [Y_NORTH_BORDER, Y_SOUTH_BORDER].

   i. Define the constants with those values
      X_WEST_BORDER = 0
      X_EAST_BORDER = 10
      Y_NORTH_BORDER = 0
      Y_SOUTH_BORDER = 10

HOMEWORK [2/3]

Add the following properties to the Airplane class:

```
char ID  // identificator for the airplane (eg 'a', 'b', 'c'...).
double fuel     // current amount of fuel in tons.
double altitude // current altitude in Km.
int xPos // x-coordinate for the latitude or abscissa ([0, 10])
int yPos // y-coordinate for the longitude or ordinate ([0, 10])
int xSpeed // x-speed for the plane or abscissa ([-1, 0, 1])
int ySpeed // y-speed for the plane or ordinate ([-1, 0, 1])
```

Add public getters and setters.

1. Verify that the new values provided in the setters comply with the range of valid values.
2. Testing double data types requires to set the accuracy level or the assertEquals

```
eg. assertEquals(3.14, Math.PI, 0.01);
```

HOMEWORK [3/3]

Design different constructors for the class including at least the following:

```
Airplane(char ID); // all the fields but the ID must be set to 0

Airplane(char ID, double fuel, double altitude, int xPos, int
yPos, int xSpeed, int ySpeed);
```

Add the corresponding toString() and print() methods.

```
// USE THIS FORMAT PLEASE!
ID: z - Fuel: 12.0 - Altitude: 7.2 - Pos[2,5] - Speed[-1,0]
```

Module 2

Interaction between Objects

Lecture 6

Object Interaction

"Today, most software exists, not to solve a problem, but to interface with other software"

I. O. Angell

Debugger

Use the debugger to observe the course of action in the execution of this method:

```
new Person("Yuri", "Gagarin", 34, false);
```

a. Pay attention to changes in the value of the fields.

Importing Packages

Modify the default constructor in Person to assign random values to the gender field.

The Random class

Modify the default constructor in Person to assign random values to the age field.

a. Values must be in the range [0, MAX_AGE_VALUE].

The Airplane's Landing Gear System

Integrated by three components: Wheels, Wheel Struts and Landing Gear Controls.

The Airplane's Landing Gear

Wikipedia. Public Domain

The Airplane's Landing Gear

Control panel.

The Wheel Class

Create a Wheel class. It has a 'size' field that represents its (max) size (measured in Bars)[1].

a. The 'size' represents the MAXIMUM air pressure of the wheel (its value must be >= 0).

b. Add a getter for this field.

c. The size of the wheel is defined during its construction, thus it will not be modified later.

d. Design a constructor for the wheel and provide the wheel's size as a parameter.

[1]The size of the wheel of a Boeing 737 aircraft is 20.7 bars.

Inflate the wheel

The current air 'pressure' is represented by a second field called 'pressure'.

a. Its value must be within [0, size].

b. Build setters and getters for this field.

c. Modify the wheel's constructor to use the 'pressure' as its second parameter.

```
public Wheel (double size, double pressure);
```

Safety first

To be operational, a wheel must contain a minimum air pressure.

Define a SAFETY THRESHOLD to define the minimum safety ratio between the pressure and the size.

```
// a safe value is usually over 80%
```

Create an 'isOperational' method that returns true if it is safe to use the wheel.

```
public boolean isOperational ();

// returns true if the ratio between the size and the
// pressure is at least SAFETY THRESHOLD
```

The Wheel Class

Complete the Wheel class adding the following methods:

```
public String toString();
public void print();
```

a. The methods must inform about the state of the wheel as well as its operational level.

b. Use String.format("%.2f", decimalNumber) to format decimal numbers with 2 decimal positions.

```
// USE THIS FORMAT PLEASE!
Size: 20,70 bars - Pressure: 19,30 bars - Ratio: 0,93 - Op: true
```

Testing the wheel

Fully test the wheel using JUnit.

a. Use different values for the size and pressure fields.

b. The toString() method can be used to test the class.

```
//Example:
Wheel a = new Wheel (20.7, 19.3);

assertEquals ("Size: 20,70 bars - Pressure: 19,30 bars - Ratio:
0,93 - Op: true", a.toString());

a.setPressure (2.3);
assertEquals ("Size: 20,70 bars - Pressure: 2,30 bars - Ratio:
0,11 - Op: false", a.toString());
```

The Wheel Strut class

The wheels are grouped in struts (small aircrafts only include a nose and two wing struts).

The Wheel Strut class

Create a 'WheelStrut' class for small planes[1] that use two wheels ('leftWheel' and 'rightWheel') per strut.

a. Initialize the 'leftWheel' and 'rightWheel' fields in the WheelStrut constructor.

b. Include a 'ID' char field to identify the WheelStruct.

c. Add a setter and getter for the ID field.

d. Implement a WheelStrut (char ID, double wheelSize) constructor.

[1]Boeing 737 and Airbus 319-20-21 series uses two wheels per strut.

The Wheel Strut class

Wheel struts are operational if ALL the wheels in the strut are operational too.

a. Implement the boolean isOperational() method for the WheelStrut class.

    i. Make use of the isOperational() method for every wheel contained in the strut.

b. Fully test this method using JUnits.

    i. How to modify the pressure on the wheels for testing?

The Wheel Strut class

Testing isOperational.

To modify the pressure of the wheels, a getter is required…

```
public Wheel getLeftWheel()
```

…but the getter is only required for testing, it is not part of the functionality of the class. Therefore….

…it must be declared protected!

```
protected Wheel getLeftWheel()
```

```
WheelStrut dummyStrut = new WheelStrut ('N', 100); //Nose strut;
dummyStruct.getLeftWheel().setPressure(5.8);
```

HOMEWORK

Add a new default constructor to the Airplane class to assign random values to all of its fields except the ID.

a. The ID must be 'X'.

```
// default constructor for Airplane
public Airplane();
```

Lecture 7

Delegation of control

“Think (design) globally, act (code) locally”

Anonymous

The Wheel Strut class

Add a 'deployed' boolean field to represent whether the struct is deployed or retracted.

a. Implement a setter for the deployed field.

b. Add a boolean isDeployed() method.

The default value for this field is true (the strut is deployed).

The Wheel Strut class

Complete the WheelStrut adding the next methods:

```
public String toString();

// USE THIS FORMAT PLEASE!
ID: N - Deployed: true - Op: true [L: true][R: true]
```

```
public void print();

// USE THIS FORMAT PLEASE!
ID: N - Deployed: true - Op: true [L: true][R: true]
L: Size: 50,00 bars - Pressure: 50,00 bars - Ratio: 1,00 - Op:
  true
R: Size: 50,00 bars - Pressure: 50,00 bars - Ratio: 1,00 - Op:
  true
```

The Landing Gear class

Models the lever (handle) used by the crew to deploy the wheel struts.

Also models the status lights of the struts.

The Landing Gear class

Create a LandingGear class

a. Add three struts (called nose, left and right) with IDs 'N', 'L' and 'R' (nose, left and right).

b. Include a boolean lever field to model the handle.

   i. Add LEVER_UP and LEVER_DOWN constants to provide semantic meaning to this field.

   ii. Default value for the lever is LEVER_DOWN.

   iii. When the handle is down all three struts must be deployed. Otherwise they must be retracted.

The Landing Gear class

Add print()/toString() methods to display the status of the system. The status includes

a. Status of the handle (DOWN/UP).

b. Operational status of the system (ON / FAILURE).

c. Status of each struct (ON = deployed / OFF = retracted / PRESS = pressure failure).

```
Lever: DOWN Status: ON      Nose: ON    Left: ON    Right: ON
Lever: DOWN Status: FAILURE Nose: ON    Left: OFF   Right: ON
Lever: UP   Status: FAILURE Nose: ON    Left: OFF   Right: OFF
Lever: UP   Status: ON      Nose: OFF   Left: OFF   Right: OFF
Lever: UP   Status: FAILURE Nose: PRESS Left: OFF   Right: PRESS[1]
```

[1]At least one of the wheels does not have enough pressure.

HOMEWORK

Fully test and hack failures[1] in all the components of the system (LandingGear, WheelStrut and Wheel).

```
//EXAMPLE.
LandingGear lg = new LandingGear();
lg.setLever(LandingGear.LEVER_UP);
assertEquals("Lever: UP   Status: ON      Nose: OFF Left: OFF
Right: OFF", lg.toString());

lg.getRight().setDeployed(true); //!hack
assertEquals("Lever: UP   Status: FAILURE Nose: OFF Left: OFF
Right: ON ", lg.toString());

lg.setLever(LandingGear.LEVER_UP);
lg.getRight().getLeftWheel().setPressure(0); // !damage
assertEquals("Lever: UP   Status: FAILURE Nose: OFF Left: OFF
Right: PRESS", lg.toString());
```

[1]The access methods to the wheel struct and its wheels must be declared as protected (they are not part of the functionality of the classes).

Module 3

Collections

Lecture 8

Arrays and Collections

"The art of programming is the art of organizing complexity, of mastering multitude and avoiding its bastard chaos"

E. Dijkstra

Landing Gear System revisited

Refactor the WheelStrut class to substitute the left and right wheels by an ArrayList of wheels.

```
ArrayList<Wheel> wheels;
```

a. Modify the WheelStrut's constructor to provide the number of wheels as a parameter.

```
public WheelStrut (char ID, double wheelSize, int numberOfWheels)
```

b. Initialize the wheels ArrayList in the WheelStrut's constructor.

Landing Gear System revisited

Rename the following methods as:

```
public void print() -> printOld
public String toString() -> toStringOld
public boolean isOperational() -> isOperationalOld;
```

Refactor the equivalent methods in the WheelStrut class to use the new ArrayList of wheels.

```
public void print()
public String toString()
public boolean isOperational();
```

Person Class revisited

Refactor the default constructor in the Person class to set a random value for the name and surname fields.

a. Hint: Use an array containing predefined values for these fields.

```
public final static String MALE_NAMES[] ={"Martin", "Peter",
"Steve", "Barry"}

public final static String FEMALE_NAMES[] ={"Klara", "Astrid",
"Mary", "Susan"}

public final static String SURNAMES[] ={"Cernan", "Armstrong",
"Aldrin", "Collins"}
```

The Seat Manager

Manages the booking system for a small Airplane[1]. The seat area is modeled using a bidimensional array.

```
Person seats[][];
```

Seat area is divided in two classes (First and Standard).

The constructor needs to know how many rows are selected for each class[2].

```
public SeatManager (int firstClassRows, int standardClassRows) {
    seats = new ...;
```

[1]Small aircrafts such as the Boeing 737 series and the Airbus 320 series have 6 seats in each row separated by an aisle.

[2]First rows in a plane are usually reserved for the First Class.

The Seat Manager

Calculate the number of seats in the aircraft and return this value as the plane's size.

```
//Virtual fields
public int getSize()

public int getSizeFirstClass()

public int getSizeStandardClass()
```

The Seat Manager

Add a bookSeat method using the following parameters.

```
public void bookSeat(Person person, int row, int col)
```

a. The method assigns the seat to the person when:

   i. The seat exists!

   ii. The seat is not already reserved.

Implement the next management related methods.

```
public boolean isReserved(int row, int col)
public Person getReservation(int row, int col)
public void releaseSeat(int row, int col)
```

The Seat Manager

Fully test all the methods in the Seat Manager.

```
SeatManager a = new SeatManager (2, 8);
a.bookSeat(new Person ("Neal", "Armstrong", 25,
Person.MALE_VALUE), 3, 4);
a.bookSeat(new Person (), 6, 2);

assertEquals (false, a.isReserved (0, 0));
assertEquals (true, a.isReserved (3, 4));
assertEquals (true, a.isReserved (6, 2));

assertEquals ("[Name: Neal - Surname: Armstrong - Age: 25 -
Gender: male]", a.getReservation (3, 4).toString());

a.releaseSeat(3, 4);
assertEquals (false, a.isReserved (3, 4));
```

HOMEWORK

Implement a getNumPax() method to return the number of passengers in the plane.

```
public int getNumPax()
```

a. Navigate across the different seats (row by row, column by column) to count the passengers.

Lecture 9

Reports and Queries

“Testing means learning. Learning requires faith in one’s ignorance combined with the confidence that it can be extinguished.”

James Bach

The Seat Manager

Implement a loadPax method to fill the airplane with a given number of virtual passengers.

```
public void loadPax(int paxNumber);


// The plane can be fully filled using the next code:

SeatManager b = new SeatManager(2,8);
b.loadPax(b.getSize());

assertEquals(b.getSize(), b.getNumPax());
```

Reports

Add a printManifest () method to get a detailed report about all the passengers on board

```
// USE THIS FORMAT PLEASE!

MANIFEST

FIRST CLASS
ROW: 0 SEAT: B - MARY SMITH - AGE: 53
ROW: 0 SEAT: C - SUSAN JOBS - AGE: 22
ROW: 1 SEAT: D - SUSAN HARPER - AGE: 63
ROW: 1 SEAT: E - PAUL DOWLAND - AGE: 87

STANDARD CLASS
ROW: 2 SEAT: F - BARRY FURNELL - AGE: 26
ROW: 3 SEAT: A - BARRY JOBS - AGE: 18
ROW: 3 SEAT: C - BRIDGITTE DOWLAND - AGE: 32
```

The Seat Manager

Implement the print method in the SeatManager class.

```
// USE THIS FORMAT PLEASE!

   A B C   D E F
 0 X X ?   ? ? X
 1 X X X   ? ? ?
 --------------
 2 ? X ?   ? ? ?
 3 X C ?   X X C
 4 X X X   X X X
 5 X X ?   X C X
 6 X X X   X X X

// X = seats booked  by adults
// C = seats booked by children (people under 18 years old)
// ? = free seats
```

Queries

Implement the following search method.

```
public int getYoungestAgeOnBoard()

// returns -1 if there is nobody on board
```

a. It can be tested using:

```
        SeatManager a = new SeatManager(2, 8);
        assertEquals(-1, a.getYoungestAgeOnBoard());

        a.bookSeat(new Person ("Yuri", "Gagarin", 44,
Person.MALE_VALUE), 0, 2);
        a.bookSeat(new Person ("Buzz", "Aldrin", 46,
Person.FEMALE_VALUE), 3, 1);
        a.bookSeat(new Person ("Sally", "Ride", 44,
Person.FEMALE_VALUE), 0, 4);

        assertEquals(44, a.getYoungestAgeOnBoard());
```

Queries

Design a method to return an ArrayList containing the youngest people on board.

```
public ArrayList<Person> getYoungestPeopleOnBoard()


SeatManager sm = new SeatManager(2, 6);
assertEquals(0, sm.getYoungestPeopleOnBoard().size());
Person yuri = new Person ("Yuri", "Gagarin", 44, Person.MALE_VALUE);
Person buzz = new Person ("Buzz", "Aldrin", 46, Person.MALE_VALUE);
Person sally = new Person ("Sally", "Ride", 44, Person.FEMALE_VALUE);

sm.bookSeat(yuri, 0, 2);
sm.bookSeat(buzz, 3, 1);
sm.bookSeat(sally, 0, 4);

assertEquals(2, sm.getYoungestPeopleOnBoard().size());
assertEquals(true, sm.getYoungestPeopleOnBoard().contains(yuri));
assertEquals(false, sm.getYoungestPeopleOnBoard().contains(buzz));
assertEquals(true, sm.getYoungestPeopleOnBoard().contains(sally));
```

[Optional] Queries

Implement the following query method

```
public int getNumPax(byte area);

// 'area' is defined as:
// FIRST_CLASS, STANDARD_CLASS, or ALL_CLASSES
```

[Optional] Queries

Test the getNumPax method

```
SeatManager a = new SeatManager(2, 8);

// First Class
a.bookSeat(new Person(), 0, 0); // Window
a.bookSeat(new Person(), 0, 2); // Aisle
a.bookSeat(new Person(), 0, 3); // Aisle
a.bookSeat(new Person(), 0, 5); // Window
// Std Class
a.bookSeat(new Person(), 4, 0); // Window
a.bookSeat(new Person(), 4, 2); // Aisle
a.bookSeat(new Person(), 4, 3); // Aisle
a.bookSeat(new Person(), 4, 5); // Window
a.bookSeat(new Person(), 5, 0); // Window
a.bookSeat(new Person(), 5, 2); // Aisle
a.bookSeat(new Person(), 5, 3); // Aisle
a.bookSeat(new Person(), 5, 5); // Window

assertEquals(4, a.getNumPax (SeatManager.FIRST_CLASS));
assertEquals(8, a.getNumPax (SeatManager.STANDARD_CLASS));
assertEquals(12, a.getNumPax (SeatManager.ALL_CLASSES));
```

[Optional] Queries

Implement the following query method

```
public int getNumPax(byte area, byte section);

/ ‘section’ is defined as:
// STARBOARD_WINDOW, BOARD_WINDOW, BOTH_WINDOWS, LEFT_AISLE
// RIGHT_AISLE, BOTH_AISLES or EVERYWHERE

// TIP:
// Set the STARBOARD_WINDOW to the row number corresponding
// to the starboard window’s column (5)...
```

[Optional] Queries

Complete the previous test to check the proper working of this new method.

```
assertEquals(1, a.getNumPax (SeatManager.FIRST_CLASS,
SeatManager.STARBOARD_WINDOW));
assertEquals(1, a.getNumPax (SeatManager.FIRST_CLASS,
SeatManager.BOARD_WINDOW));
assertEquals(2, a.getNumPax (SeatManager.FIRST_CLASS,
SeatManager.BOTH_WINDOWS));
assertEquals(1, a.getNumPax (SeatManager.FIRST_CLASS,
SeatManager.LEFT_AISLE));
assertEquals(1, a.getNumPax (SeatManager.FIRST_CLASS,
SeatManager.RIGHT_AISLE));
assertEquals(2, a.getNumPax (SeatManager.FIRST_CLASS,
SeatManager.BOTH_AISLES));


assertEquals(2, a.getNumPax (SeatManager.STANDARD_CLASS,
SeatManager.STARBOARD_WINDOW));
...
```

HOMEWORK [1/3]

Implement a boolean fly() method in the Airplane class.

a. If there is enough fuel, it reduces its amount in 1.0 unit, returning true. Otherwise it returns false.

b. If there is enough fuel, it updates the position of the aircraft using the speed fields.

```
// eg. Given a plane located at (x,y) == (2,6)
// with speed (x, y) == (-1, 1) and fuel 3.5, a call to fly()
// would update the plane position to (1,7) and its fuel to
// 2.5, returning true.
```

c. If either the x or y coordinate is in the border of the gameboard, the coordinate will not be updated.

HOMEWORK [2/3]

Fully test the fly method using JUnits.

a. You may use the toString() method to facilitate the test process.

```
// EXAMPLE
Airplane a = new Airplane ('z', 12.0, 7.2, 2, 5, -1, 0);

assertEquals ("ID: z - Fuel: 12.0 - Altitude: 7.2 - Pos[2,5] -
Speed[-1,0]", a.toString());
a.fly();
assertEquals ("ID: z - Fuel: 11.0 - Altitude: 7.2 - Pos[1,5] -
Speed[-1,0]", a.toString());
a.fly();
assertEquals ("ID: z - Fuel: 10.0 - Altitude: 7.2 - Pos[0,5] -
Speed[-1,0]", a.toString());
a.fly();
assertEquals ("ID: z - Fuel: 9.0 - Altitude: 7.2 - Pos[0,5] -
Speed[-1,0]", a.toString());
```

HOMEWORK [3/3]

[OPTIONAL] Modify the loadPax method to distribute the passenger uniformly across the aircraft.

a. Don't place all of the passengers in the first rows!

```
public void loadPax (int paxNumber)
```

Lecture 10

Putting it all Together

"Everything is theoretically impossible, until it is done"

Robert A. Heinlein

SeatManager Class

Balanced distribution for loading pax.

```
…
int remainingFreeSeats = getSize() - getNumPax();

Random generator = new Random ();

for (int i=0; i<seats.length && paxNumber>0; i++)
 for (int j=0; j< seats[0].length && paxNumber>0; j++)
  if (seats[i][j] == null)
  {
    if (generator.nextInt(remainingFreeSeats) < paxNumber)
    {
      seats[i][j] = new Person();
      --paxNumber;
    }

    --remainingFreeSeats;
  }
```

FSim

Create a FSim class to emulate an Air Traffic Simulator

a. Use an ArrayList field to store objects of the Airplane class.

```
private ArrayList<Airplane> planes;
```

b. Use a bidimensional array of chars to emulate the air space map.

```
private char [][] map;
```

c. Initialize the data structures in the FSim 's constructor reserving memory for them!

FSim

Design a method to add planes to the simulation

```
public void add (Airplane plane)


// TEST
FSim fsim = new FSim();

fsim.add(new Airplane ('A', 15.0, 7.0, Airplane.X_WEST_BORDER,
Airplane.Y_NORTH_BORDER, +1, +1));
fsim.add(new Airplane ('B', 15.0, 7.0, Airplane.X_EAST_BORDER,
Airplane.Y_NORTH_BORDER, -1, +1));
fsim.add(new Airplane ('C', 15.0, 7.0, Airplane.X_WEST_BORDER,
Airplane.Y_SOUTH_BORDER, +1, -1));
fsim.add(new Airplane ('D', 15.0, 7.0, Airplane.X_EAST_BORDER,
Airplane.Y_SOUTH_BORDER, -1, -1));
```

FSim

The paint method displays the background and the planes flying over it

```
private void paint()
    {
        // Clear the map painting ground elements
        paintBackground();

        // Lays the planes IDs over the ground
        paintForeground();

        // Prints each cell on the map
        // use the System.out.print method
    }
```

The paintBackground method

Fills the map with the terrain symbol and the radar control strips.

```
// SAMPLE OUTPUT

\~~~~|~~~~/
~\~~~|~~~/~
~~\~~|~~/~~
~~~\~|~/~~~
~~~~\|/~~~~
-----/-----
~~~~/|\~~~~
~~~/~|~\~~~
~~/~~|~~\~~
~/~~~|~~~\~
/~~~~|~~~~\
```

The paintForeGround method

Browse the list of aircrafts adding their ID's to the map in the slot referenced by their own coordinates.

```
// SAMPLE OUTPUT

\~~~~|~~~~/
~\~~~|~~~/~
~~D~~|~~C~~
~~~\~|~/~~~
~~~~\|/~~~~
-----/-----
~~~~/|\~~~~
~~~/~|~\~~~
~~B~~|~~A~~
~/~~~|~~~\~
/~~~~|~~~~\
```

The animate method

Define an animate method to draw the next frames in the simulation.

```
public void animate (int frames)
```

1. Paints the frame (calls to the paint method).
2. Invokes the fly method in each airplane to update its position (to be painted in the next frame).

HOMEWORK

Fully test every class in the project (SeatManager, FSim, Person, Airplane, Landing Gear, etc).

Its proper operation might be verified by the instructor during the next test.

Extra Lectures

Exercises

“I do not fear computers. I fear lack of them”

Isaac Asimov

Exercise

Add this method to the SeatManager class.

```
public int countFirstClassElderlyPax (boolean gender)
// counts the number of passengers in the first
// class (by gender) whose age is over 65

// Example
assertEquals(3,  dummy.countFirstClassElderlyPax
 (Person.MALE_VALUE));


assertEquals(1,  dummy.countFirstClassElderlyPax
 (Person.FEMALE_VALUE));
```

Exercise

Add this method to the SeatManager class.

```
public void interchangeSeats (int colA, int colB)
// Interchange the seats of passengers in columns A and B


// example
 dummy.interchangeSeats(0, 5);
```

Exercise

Add this method to the Math class.

```
public boolean isPalindrome (String text)
// Returns true if the text is a Palindrome.
// BLANKS ARE NOT CONSIDERED!

// hint:
// the method… charAt(int index) returns the
// char value at the specified index.
//
// the method length() returns the String's length




// Example
assertEquals(true, g.isPalindrome("Never odd or even"));
assertEquals(false, g.isPalindrome("Once upon a time"));
```

Exercise

Add this method to the Math class.

```
public ArrayList<Integer> getPrimesUnder (int limit)
// Returns an Array List containing the prime numbers
// lower than 'limit'

// Example
assertEquals("[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]",
 dummy.getPrimesUnder (100).toString());
```

Exercise

Add this method to the Math class.

```
public int[] sortIntArray (int[] array)
// Sorts an array of int(s) using the Selection algorithm

// Swaps the minimum element of the collection with the
// one at position 0
// Swaps the minimum element of the remaining elements
// with the one at pos position 1
...

// Example
int input[] = { 100, 3, 25, 7, 14};
int output[] = {3, 7, 14, 25, 100};

assertArrayEquals(output, dummy.sortIntArray(input));
```

Exercise

Add this method to the Math class.

```
public int[][] sumMatrix (int[] a, b[][])
// Verifies that the size of both input arrays is the
// same


// if so, creates a new array with the sum of both
// (cell by cell)
...

// Example
int a[][] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int b[][] = {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};
int output[][] = {{11, 22, 33}, {44, 55, 66}, {77, 88, 99}};

assertArrayEquals(output, dummy.sumMatrix(a, b));
```

Exercise

Add this method to the Math class.

```
public int[][] randomMatrix (int row, int col)

// returns a matrix filled with non repeated random values
// for the given dimensions.
```

Exercise

Add this method to the Math class.

```
public boolean verifyUniqueElements (int[] input)
// Returns true if all the elements in the input
// matrix are different



// Example
int wrong[][] = {{77, 22, 33}, {44, 55, 66}, {77, 88, 99}};

 dummy.printMatrix(false, dummy.verifyUniqueElements
(wrong));
        dummy.printMatrix(true, dummy.verifyUniqueElements
(dummy.randomMatrix (4, 4)));
```