Java Labs 2021
No Paint, no gain

Christophe Barès, Nicolas Papazoglou,
Sylvain Reynal, Antoine Tauvel, ENSEA [*]

October 2021

1 Introduction

Welcome to this lab. This lab has been thought to be worked on individually.
Here are some basic rules for it:

- As you are now at graduate entry level, we shall not correct missing semi-colons or similar stuff. Definitely not.
- Before the end of the first class session you should have a **working setup** and you should be able to **compile and execute** a simple "Hello World" program (see next section for more on this).
- Your backup, your problem.
- Please use a professional tool to monitor modifications made to your code. Git is the one you are looking for.

2 Software requirement / Homework

This series of Labs will run on plain old Java[1] alongside the Swing library. Swing is available on all major Operating Systems (Windows, Linux, MacOS). Screenshots provided below are those obtained using IntelliJ IDE, though you are free to use any IDE you want (yet we strongly advise you against plain old Geany or any other IDE without refactoring or code completion ability as they swiftly prove useless when it comes to serious coding).

[*]This lab was initially designed by David Picard and his team at ENSEA back in the 2010's. It has been rewritten numerous times since then.

[1]That is, anything from Java 1.8 to Java 17. Please observe how the general versioning scheme has changed when going from Java 1.8 to Java 9, which is actually Java 1.9, but was renamed Java 9 for strategic reasons pertaining to Oracle.

First thing we need is a basic setup to write code, compile and execute Java code[2]:

- Download the last stable version of the Java Development Kit (aka JDK)
- Download the community edition of IntelliJ.

Then we need a working Hello world console application[3]:

```java
public class Main {

    public void main (String[] args){
        System.out.println("Hello world");
    }
}
```

Listing 1: A basic ”Hello world” in Java tongue.

Write and test this simple ”Hello world” application.

3 Overview of the global project

The application we are going to create is described by the class diagram shown in Figure 1 (page 4). Let us analyse some of these classes:

- The Point class serve as a basis class to construct the origin point (upper left corner) of each figure.
- The Figure class is an *abstract* class that acts as the super class of every kind of figure we may want to draw. It has a Color field.

[2]Beware: the Java Development Kit (JDK) is not the same as the Java Runtime Exectution (JRE). JDK is somehow a JRE plus a compiler and a bunch of additional developer tools like a profiler, a decompiler, a tool that generates glue code when linking C code with Java, etc

[3]A console application is an application that can be run from a Console (which some operating systems call a Shell, or a Terminal for that matter, but these are all synonyms for one unique function: interacting with your computer using words and sentences, and as this is greek to most people on earth, then if you become fluent with it, this will immediately propel you as the next genius hacker to your friends...)

- The Rectangle class extends the abstract Figure class and implements its abstract methods[4].
- The Square class is a *child* (or a *subclass*) of the Rectangle class (we say that the Square class "derives" from it). The Square class overrides the setLength() et setWidth() methods.
- The Drawing class is a container for all the Figure's that have been drawn. It uses an ArrayList object for this purpose. The Drawing class extends the JPanel class. The Color field in the Drawing class is the current color for any newly created figure.
- The Window class is the Graphical User Interface (GUI) main container. It uses the Swing API for rendering its content.
- Finally the Serializable class is used to store and retrieve the different drawings.

Panic disclaimer: you do not have to understand every single relationship between the different classes exposed here. Just take things for granted, at least to begin with. There is an *Object Oriented Design* (OOD) approach to software design that would indeed help you a lot here, assisting you in building a global class diagram like the one which is being shown in Figure 1, but it is not part of this course. Of course in case you want to know, there is a more specialized 3IS track during your senior year at ENSEA during which you would learn how to practically implement this OOD approach using e.g. UML methods.

[4]Abstract methods are methods without body (no implementation). Abstract classes have at least one abstract method, and interfaces are classes with only abstract methods. All these serve as a kind of *blueprint* or *mould* for subclasses. More on this in the next section

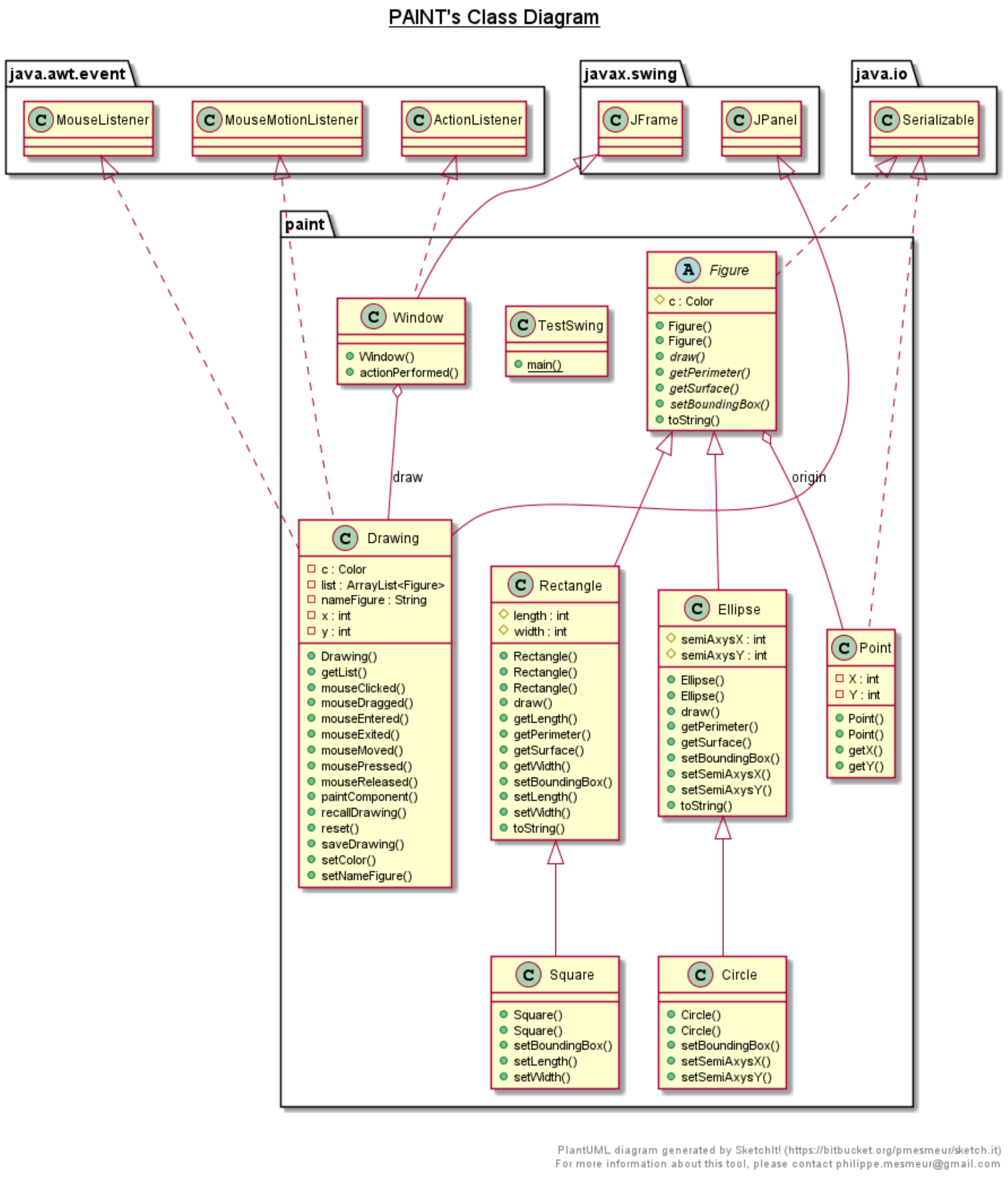

Figure 1: Our application class diagram

4 The different Figures classes

4.1 Our base: a point

Java offers many ways to define a geometric point. In order to start at the lowest point possible (SR: je comprend pas cette phrase), we are going to define a Point class that will serve as origin to all of our figures.

Write and test a simple Point class with every field having a getter and a setter method.
This class overrides the toString method to display `"(X,Y)"` when invoked.
Two constructors are concurrents: a `public Point ()` one, initializing X and Y to 0, and a `public Point (int a, int b)` one.

4.2 The Figure class

Figure is an abstract class that declares two abstract methods:

- `public abstract void setBoundingBox (int heightBB, int widthBB);`
- and `public abstract void draw (Graphics g);`.

An abstract class is a class that you cannot instanciate as such. It can only be used as a mould to create new and more complex objects. You cannot test an abstract class.

One of the powers of abstract classes is that you cannot build a new *Figure* without implementing both *draw* and *setBoundingBox*. The compiler just will not let you do it. Think of it as trying to live in a house that has some rooms missing: they are on the blueprint, but have not been built yet, so you just cannot use them! That is why we say they are **abstract**.

The other great feature of abstract classes is that this *abstraction* mechanism allows us to use **polymorphic** versions of *draw* for example.

Some attributes can be final (we cannot modify them after they have been created). Which are they? Are you sure about the Origin? Why is it wise to make these attributes *protected*, instead of *public* or *private*?

Write the Figure class with every field you deem necessary. Create getter() methods for all attributes, except for the width (y size) and the length(x size).
The constructor may accept two parameters: a java.awt.Color and a Point.
Create a setBoundingBox method that sets value for width and length.
As always this class must override the toString method to display appropriate information regarding the state of the Figure object.

4.3 The Rectangle and the Ellipse classes

These classes are quite similar, they differ only in the way their respective shapes are drawn in the end.

Write and test these two classes using the following constructors:

- `public Rectangle (int px, int py, Color c);`
- `public Ellipse (int px, int py, Color c);`

width and length are initially sets to 0.
Implement the **`setBoundingBox`** and **`draw`** methods. For now the **`draw`** method body should be left empty.

4.4 The square and the circle classes

Why should the setBoundingBox method be overridden in the Square and Circle classes?
What value should you use when calling setBoundingBox? Only x? Only y? the minimum? the maximum?

Write and test these two classes.

5 The Graphical User Interface

5.1 Let's draw it !

In order to do this part, you should watch the video on Moodle about the Swing package.

The Swing package enclosed many objects to create graphical applications. While he has aged a little bit (it's main problem is that there is no support for responsiveness), it's still in use for many application. Here's a sample of code for a simple Hello World code.

As you can see, a window must extend JFrame in order to run. One of the main concept we'll be using is the JPanel object. Those objects are displayed inside the JFrame. Each has its own stack method and position. In order to draw our final GUI (see the look of if on figure 2 page 7), we'll be using three JPanel: two to stack the figure button and the color button, and one with our Drawing, the next classes that will extend JPanel.

Here's the code to create one ("South" obviously reffers to the place of the JPanel in the Window):

```
JPanel southPanel = new JPanel();
southPanel.setLayout(new GridLayout(1,2));
southPanel.add(/*add your button there*/);
contentPanel.add(SouthPanel,"South");
```

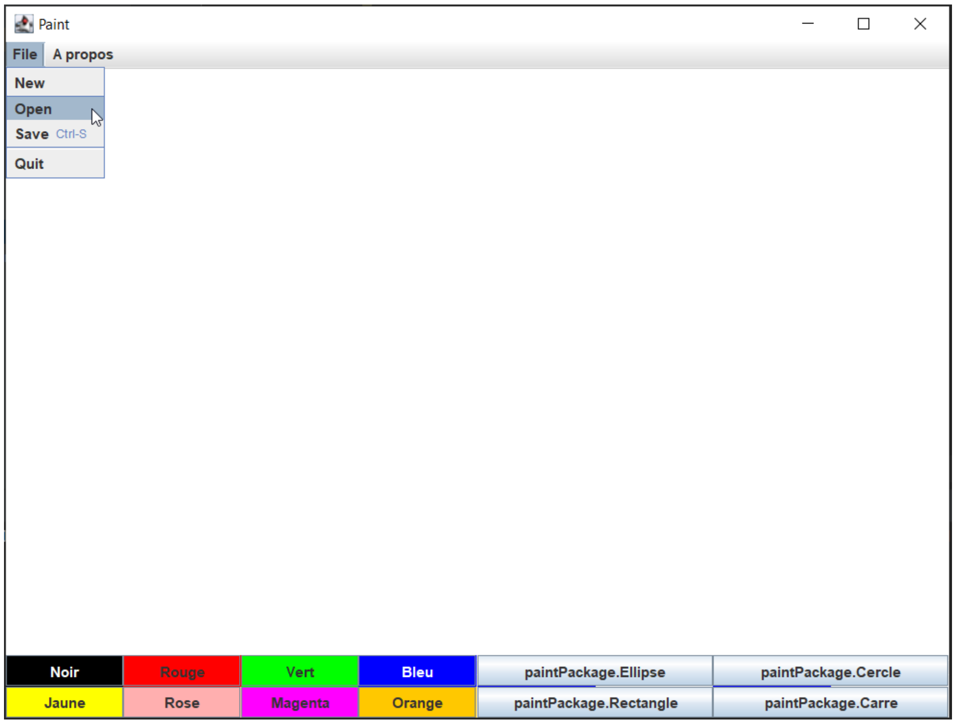

Figure 2: Application screenshot

Create and test the final look of our GUI. At this moment there is no interaction with the mouse.

```java
public class Window extends JFrame {

        public Window(String Title,int x,int y)
                {
                super(Title);
                this.setSize(x,y);
                this.setVisible(true);
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                Container contentPanel = this.getContentPane() ;

                JMenuBar m= new JMenuBar();

                JMenu menu1= new JMenu("File");
                JMenuItem open = new JMenuItem("Open") ;
                menu1.add(open);
                m.add(menu1);

                JButton OkButton= new JButton("Que viva ENSEA !");
                contentPanel.add(OkButton);

                this.setVisible(true);
                }

    public static void main (String args[]){
                Window win = new Window("Paint it black",800,600);
        }

}

```

Listing 2: The classical "hello world" application with the Swing API

5.2 The Drawing class

As seen on the previous work, the Drawing class extends the JPanel class. On the constructor, it fixes it's own background color to white (`this.setBackground(Color.white);`), the current color is set to black and the current figure to Rectangle.

Create the first draft of the Drawing class. For now, the methods is just a constructor with two setter methods for the current color and the current figure. Don't forget to implement an ArrayList of Figures to store our Figures.

5.3 Interraction between the Drawing class and the Window class

. The goal of this section is to build interactions between our button and the Drawing class. To that let's go back to our button on the Window class. We'll modified the class so that it implements the abstract class ActionListener. Let's see how to add some interaction with the mouse:

```
JButton OkButton= new JButton("Que viva ENSEA !");
contentPanel.add(OkButton);
OkButton.addActionListener(this);   // This because this class implements ActionListener.
```

Further away, you need to override the actionPerformed method in which you'll treat every click on the Buttons or in the Menu.

```
public void actionPerformed(ActionEvent e)
{
        String cmd=e.getActionCommand();

        switch (cmd)
                {
                case "Que viva ENSEA !":
                    System.out.println("I've been clicked !");
                    break;
                ...
                }
}
```

Listing 3: An example of button click / program interaction

Modify and test the Window class so that each click on the different button modify the current Drawing color and shape.
Also implements an Information popup that comes when click on the About / Authors menu to proudly add your name to your work.
Bellow is a sample of how to display such an information popup.

```
JOptionPane info = new JOptionPane();
info.showInternalMessageDialog( info, "Authors: Insert your name here",
           "information",JOptionPane.INFORMATION_MESSAGE);
```

5.4 Let's draw !

This is where things tends to get tricky. Don't panic but be aware that you might need the help of the debugger and the help of your teacher.

> Change the Drawing class so that it implements the MouseActionListener abstract class. Why is the class not compiling now?

As the *Drawing* class extends JPanel, it can override the paintComponent method, a methods that starts by calling the super class paintComponent method. After that, you'll just have to scan every object in the ArrayList to call the Draw method on them.

Of course, the Draw method must have been implemented in the Rectangle and Ellipse classes.

> Add handler for all the missing methods.
> We want that whenever the user click on the Drawing, we create a new Figure in the list. This figure has the color and the shape selected by the last click on the different buttons. The Origin of the Figure is set where the mouse was clicked.

> The Drawing tends to work well in down / right direction but not in other direction. How to correct this bug?

6 Saving our masterpieces

In order to do this section, you should watch the Moodle Video about File handling in Java.

6.1 Saving objects

To save an object, it must be a Serializable one. Once an object implements this abstract class, all the attributes can be serialised in order either to send them through a network or to save them.

> Add handler for all the missing methods.
> We want that whenever the user click on the Drawing, we create a new Figure in the list. This figure has the color and the shape selected by the last click on the different buttons. The Origin of the Figure is set where the mouse was clicked.

```java
        public void save(){
                try{
                        FileOutputStream fos = new FileOutputStream("sauveDessin");
                        ObjectOutputStream oos = new ObjectOutputStream(fos);

                        oos.writeInt(liste.size());
                        for(Figure f : liste){
                                oos.writeObject(f);
                                }
                        oos.close();
                }
                catch (Exception e){
                        System.out.println("Problemos !");
                }
        }
```

Listing 4: Sample code to save a File

You can get inspiration from this code sample:

7 Conclusion

Don't forget to upload in moodle your GIT repository adress. We'll monitor your progress through it. Most of your grade in this project depends on your implication and on your code quality.

All the team hope you'll enjoy this ride in the land of Object oriented programming and is eager to look at your future realisation in Java.