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 this Lab :

- You're now on your fourth year of graduate study. We won't correct missing semicoloms.
- At the first session, your setup must be working, and you must be able to compile and execute a simple "Hello World" program (see next section).
- Your backup, your problem.
- Use a tool to follow your software modification. Git is the one you're looking for.

2 Software requirement / Homework

This series of Labs runs on the plain old Java assorted to the Swing librairies. This library is available for all kind of Operating Systems. The screenshot made here we'll be done using IntelliJ IDE. While you're free to use any IDE you want, we strongly advise you against plain old Geany (or any other IDE without refactoring or code completion ability).

First, we'll need a basic setup :

- Download the last stable version of the Java Development Kit.
- Download the community edition of IntelliJ.

[*]This labs originally was designed by David Picard and his team at the ENSEA back in the 2010's. It has been rewrited numerous time since.

Then we'll need a working Hello world console application.

```java
public class Main {

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

Listing 1: Java Basic ”Hello world”

Create and test a simple ”Hello world” empty application.

3 Overview of the global project

We'll build the application describe by the following class diagram (cf figure 1 page 3). Let's analyse some of it :

- The Point class serve as a base to construct the origin point (upper left) of each figure.
- The Figure class is an abstract class that is the super class of every kind of figures. It encloses a Color.
- The Square class is a child of the Rectangle class that implements Figure. The Square class override the setLength et setWidth methods.
- The Drawing class contains mainly an ArrayList that contains all the Figure. It extends the JPanel class. The Color in Drawing class is the current color for any new figure.
- The Window class consist of the GUI. It uses the Swing API for is display.
- The Serializable class is used to store and retrieve the different drawing.

You don't have to understand at first all the relation between the different classes. The object oriented conception subject (that leads to this conception) is not part of this courses. The IS section in your senior year at the ENSEA will teach you how to do this conception.

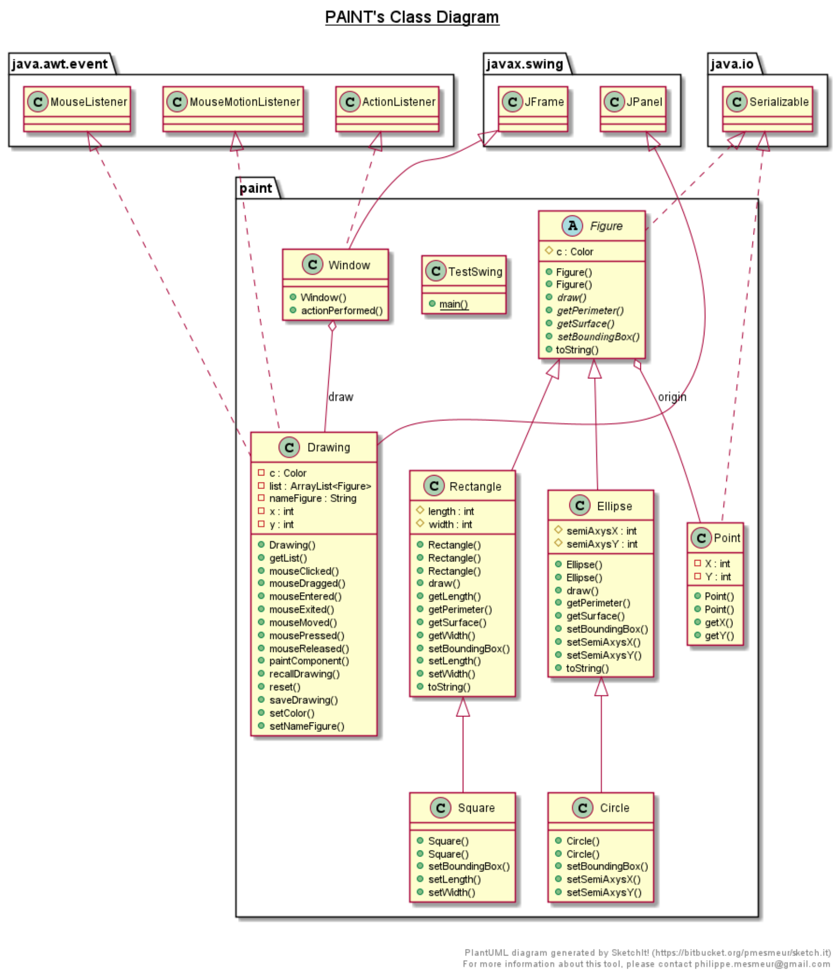

Figure 1: Our application class diagram

4 The different Figures classes

4.1 Our base : a point

Java offers many ways to define a point. In order to start at the lowest point possible, we'll define a Point class that will serve as origin to all of our figures.

Create 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 defines two abstract method : `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 great features of abstract class is that you won't be able to build a new *Figure* that doesn't implement *draw* or *setBoundingBox*. The other great feature is that this allows us to use polymorphic version of Draw.

Some attributes can be final (we won't modified them after their creation). Which are them ? Are you sure about the Origin ?
Why is it wise to make this attributes protected, instead of public or private ?

Create the Figure class with every field. Create getter methods for all attributes, except the width (y size) and the length(x size).
The constructor accepts two parameters : a color (use java.awt.Color) and a Point.
Create a setBoundingBox method that sets value for width and length.
As always, this class overrides the toString method.

4.3 The rectangle and the ellipse classes

Those basic classes are more or less the same, they'll just differ in the way we will draw them afterwards.

Create and test these two classes. Here are the two constructors :

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

At the beginning width and length are sets to 0.
Add the `setBoundingBox` and the `draw` methods. For now, `draw` is an empty method.

4.4 The square and the circle classes

Why must the setBoundingBox be overridden by the Square and Circle Classes ?
What value should you use when calling setBoundingBox ? Only x ? Only y ? the minimum ? the maximum ?

Create and test those 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");
```

```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

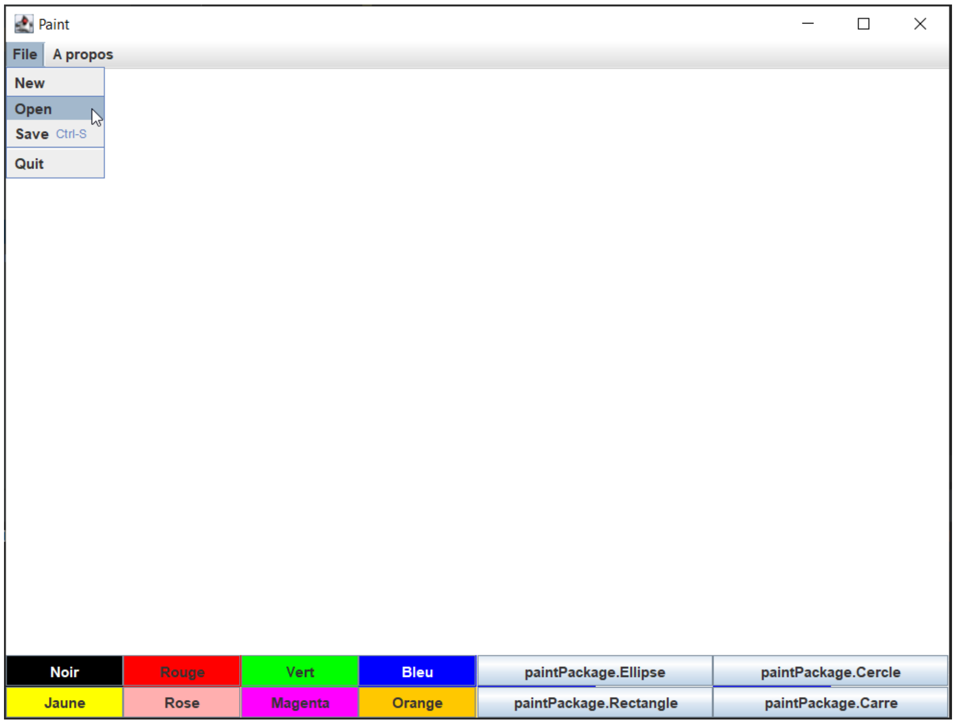

Figure 2: Application screenshot

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

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 MouseAction-Listener 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.

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.

```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