ÉTS Le génie pour l'industrie	**INF-111** Practical work #3 **Group: 2 students maximum:**(one report). **Delivery**:December 9, 2022. **Author :**Frederic Simard and Pierre Belisle

1 - Introduction

1.1 - Academic background

This third assignment aims to complete the course by leading you to develop the fundamental concepts associated with creating a graphical user interface (GUI). It aims to acquire and consolidate the following knowledge:

- The use of inheritance concepts,
- The Observer Boss
- The MVC boss
- The use of libraries associated with the Java Swing framework.

1.2 - Description of the problem

In the previous assignment, we implemented a solution method for a graph connection optimization problem.

The problems with homework #2 are:

- The parameterization of the problem is done with constants and a change requires recompiling and restarting the program.
- The visualization of the solution in the form of console printouts does not allow us to appreciate the result.

To remedy the situation, we are therefore adding a graphics component to our program.

2 - Overview of the provided code

To prevent you from remaining bogged down in the program developed during assignment 2. A functional solution has been provided to you. In addition, the first elements of the program have also been put in place to help you get started on the right foot.
In the provided files you have:

- The MVC model, ie the solution of TP2, whose operation is encapsulated in a class called ProblemeManager. Note that this class is implemented following the Singleton pattern.

- The main frame, i.e. an empty frame. Note in passing the lines in comments in the run() method. You can uncomment these lines when you have created your main panel.

- There is also the main program, look at it, but you will not have to modify it.

- Finally, there is a class UtilitaireSwing, to which the statement refers during the lab.

3 - MonObservable and MonObserver

As explained in class, the Observable class and the Observer interface provided by Java do not work in a multitasking context. So you need to create your own Observer pattern.

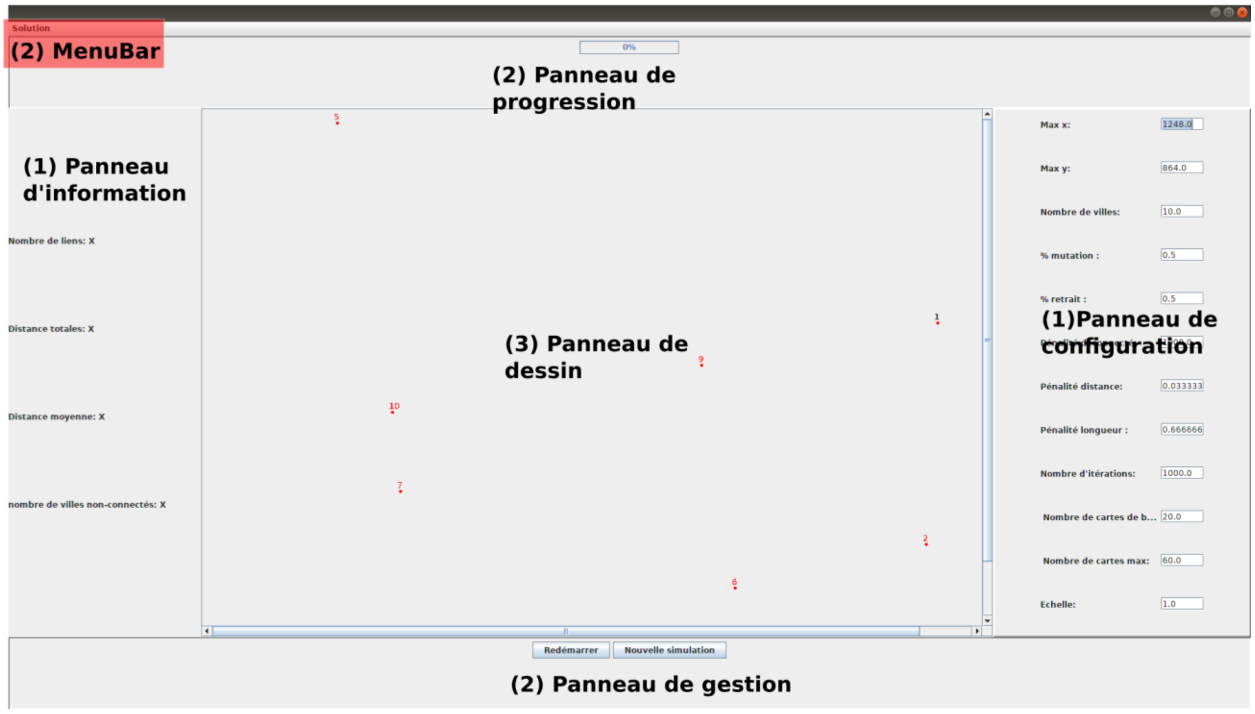
Use the examples given in class to implement your own Observer. The name of the abstract class must be MonObservable and the interface MonObserver. Both must be in an Observer Package.

Set up a small program to test your observer.

4 - Overview of the graphic window

At the end of the assignment, this is what your graphics window should look like:

(1) CadrePrincipal::PanneauPrincipal



Nombre de liens: X

Distance totales: X

Distance moyenne: X

nombre de villes non-connectés: X

Max x:	1248.0
Max y:	864.0
Nombre de villes:	10.0
% mutation :	0.5
% retrait :	0.5
[illegible]	[illegible]
Pénalité distance:	0.033333
Pénalité longueur :	0.666666
Nombre d'itérations:	1000.0
Nombre de cartes de b...	20.0
Nombre de cartes max:	60.0
Echelle:	1.0

The different elements of the window are as follows:

1) Main Frame
(JFrame) main which defines the behavior of the window and which contains the main panel.

2) Main panel
(JPanel) which groups all the other panels.

3) Control Panel
(JPanel) which allows to read/write the configurations of the problem.

4) Information panel
(JPanel) which presents the information relating to the solution found.

5) Progress panel
(JPanel) which indicates the progress of the algorithm.

6) Menu Bar
(JMenuBar) used to obtain the list of solution links.

7) Management panel
(JPanel) which allows the operation of the algorithm.

8) Drawing panel

(JPanel) which displays the solution graphically.

5 - Creation of panels

5.1 Main panel

The main panel is a container that contains all the other panels.

- Create a new class, MainPanel, inheriting from the JPanel class
- The class constructor must receive the screen size, provided by the main frame.

- The constructor must also initialize a member field with a reference to the ProblemeManager. (hint: look at the mainframe constructor)

5.2 Other Panels

- Create all the other panels, except the Drawing panel, as classes deriving from JPanel. Leave the definitions empty except for the constructor, which must receive a parameter of type Dimension and called size.
    - progress panel
    - management panel
    - config panel
    - info panel
- Now add a reference for each of these panels in the main panel.

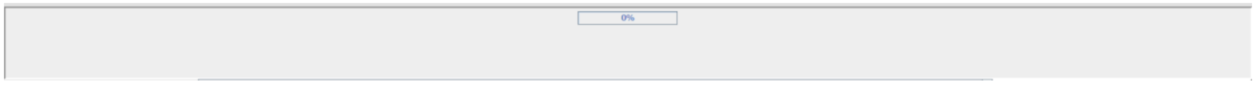
- Add a method to the main panel which is used to initialize the panels and add them to the Layout. The panel sizes are as follows:
    - progress panel
        - width = main panel width
        - height = 0.1 x height main panel
    - management panel
        - width = main panel width
        - height = 0.1 x height main panel
    - config panel
        - width = 0.2 x width main panel
        - height = main panel height
    - info panel
        - width = 0.15 x width main panel
        - height = main panel height

The layout to use is the BorderLayout (https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html)

Place each of the panels in their respective place.

5.3 Mechanism for indicating progress

- In the progress panel, add a java swing progress bar.



- To refresh the progress bar, we will use the Observer boss you made. In our pattern, the MonObservable class is the problemManager and the MonObserver interface is the progress panel.
- The link between the two classes must be made in the Main panel.
- Each time the progress member variable of the problemManager class changes, the problemManager must notify the progress panel, which must make a call to getProgres(), to obtain the value of the member field. This value should be used to change the progress bar value.

Note:in the constructor of the progress panel, add the initialization of a reference to the problemManager.

6 - Operation of the problem

6.1 Implementation

- In the management panel, add two java swing buttons.
- The first button must be used to reset the problemManager. That is, call the resetPopulation() method of problemeManager.
- The second button must be used to start problem solving. That is, call the go() method of problemManager.

In both cases, you must use an internal event handler, as explained in class.

Note:in the constructor of the management panel, add the initialization of a reference to the problemManager.

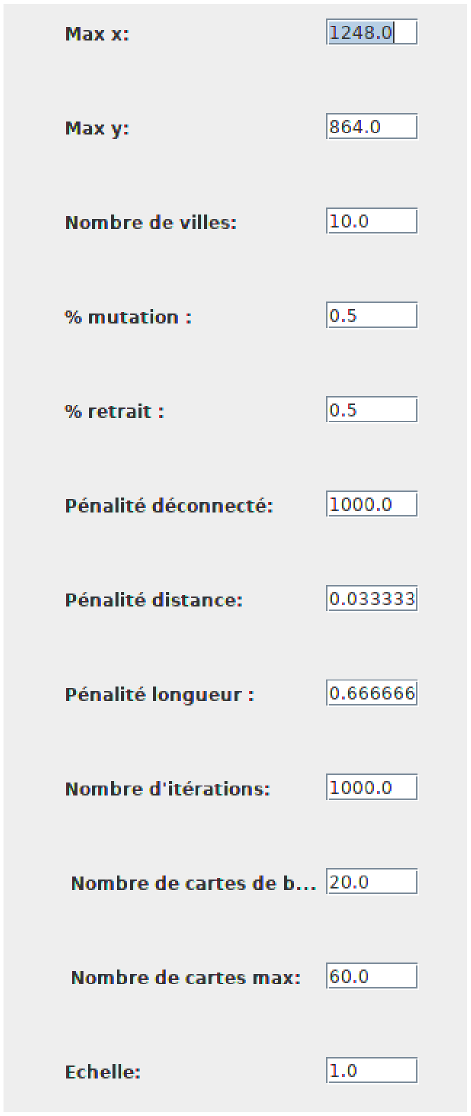
6.2 Approval

Validate the operation of your program. With the buttons you should be able to control the problemManager. And the progress status should be reflected in the progress panel.

7 - Modification of configurations

7.1 Initialization of the configuration panel

The configuration panel must provide access to the model configurations. All configurations must be available.



Max x:	1248.0
Max y:	864.0
Nombre de villes:	10.0
% mutation :	0.5
% retrait :	0.5
Pénalité déconnecté:	1000.0
Pénalité distance:	0.033333
Pénalité longueur :	0.666666
Nombre d'itérations:	1000.0
Nombre de cartes de b...	20.0
Nombre de cartes max:	60.0
Echelle:	1.0

For each of the configuration fields, the presentation of the value must be done using two elements: a JLabel and a JTextField.

The JLabel must indicate the name of the field, while the JTextField must represent the value of the field and allow its editing.

The initial values presented must be the default values for the fields. After initialization, on the other hand, the communication only takes place from the view to the model.

You have 2 options to make this panel:

1) Either you add the number of element pairs needed and create the number of ActionListener needed. This is the long way here.

2) Alternatively, you can set another panel type class which contains 2 elements {JLabel and JTextField} which can be configured in order to operate each of the configurations.

Version (2) requires a little more thought, but is much shorter to program.

A hint: study the setConfig method, in the configuration class.

7.2 other actions

Each time a configuration is changed, the model must be reset, by calling resetPopulation().

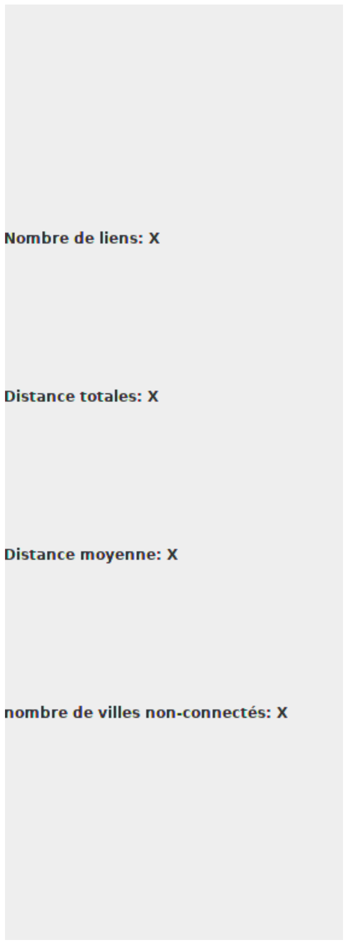
NOTE: A revision of the problemManager file has been released where a reset breaks the solving task, make sure you have this in your program.

7.3 Approval

Validate the operation of your program. With the buttons you should be able to control the problemManager. And the progress status should be reflected in the progress panel.

The most obvious configurations to change are the number of towns and the number of iterations, both of which should have an effect on progress speed.

8 - Information panel



8.1 Information panel initialization

The information panel should display information about the solution found. The panel only displays information from the model, but does not allow editing. Therefore, we only use JLabels.

Note that information should only be displayed when the best solution has been found. In cases where there is no active solution, display 'X's

Considering the knowledge you have acquired so far, the realization of this panel should not create any problem for you. The figure shows an image from the solution, but I leave you carte blanche for the realization of this element.

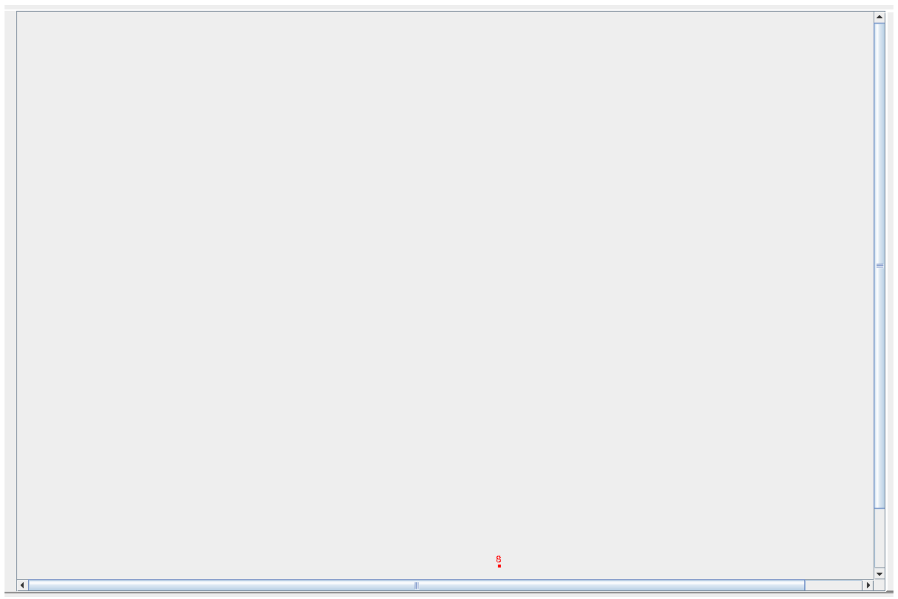
8.2 Approval

Make sure the information is updated when the progress bar reaches 100%. When resetting the resolution, we must find the 'X's again.

9 - Drawing Panel

9.1 Drawing panel initialization

In section 5 we left the drawing panel aside, it's time to add it to our project. The drawing panel is special because it includes two Scrollers, allowing you to explore the connection map:



The first step is to define a class that inherits from JPanel and instantiate it in the main panel, just like you did for the other panels. Here is the drawing panel size:

- drawing panel
  - width = 0.65 x width main panel
  - height = 0.8 x height main panel

Then, we must attach the panel to a JScrollPane, and it is this ScrollPane that we add to the main panel.

Here is the link to a representative example of what you need to do, I'll let you interpret it to achieve your ends.

http://www.java2s.com/Code/Java/Swing-JFC/AsimpleJScrollPanedemonstration.htm

9.2 Definition of the drawing panel

Constructor and initialization

The constructor must receive by parameter a reference to the population of cities and the size of the window. The reference to the city population must be copied into a member of the class. Size should be used with**UtilitySwing.setDimension**, to set the size of the window.

The drawing panel should also have a reference to the configuration and the problemManager. These references are defined in the constructor.

Additionally, the drawing panel size should be used to define the maxX and maxY configurations. This ensures that cities will not be placed outside the window.

9.3 Drawing panel operation

Here is the list of methods to define in your drawing panel:

setPopulation, this public method allows to replace the population of cities referenced by the panel. This method must also call the methods inherited from JPanel: **validate()**and**repaint()**, which are necessary for the proper operation of the graphics area.

setMap, the drawing panel must contain a reference to a map (to be added). This public method is used to replace the map referenced by the panel. This method must also call the method inherited from JPanel:**repaint()**, which is necessary for the proper operation of the graphics area.

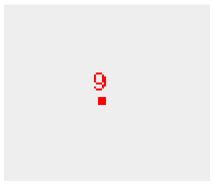
paintComponent, the drawing panel must override the paintComponent method inherited from JPanel and obtain a reference to a Graphics2D class, by casting. Refer to the example in class.

This method must call on two subroutines defined in the next section. First she has to call **drawCities()**, then in case the map is not null, it should also make a call to clearRect again, then call**drawLinks()**.

9.4 Graphical methods

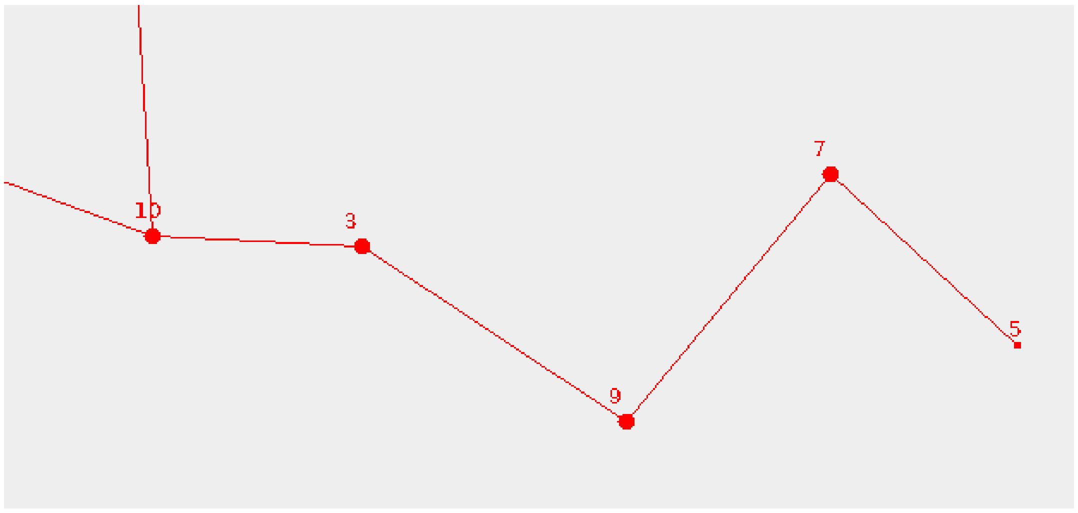
drawCities. For each city contained in the city population, this displays a RED circle of radius 5, at the position of the city. Additionally, it displays the city number above the circle. Here is an example of what is displayed for city number 9.

It is recommended to program the drawing of a city in a subroutine.



drawLinks. In the case where a map has been attached to the drawing panel, this method goes through the whole list of links, drawing a line between the source city and the destination city.

After commissioning, you should get something like this:



9.5 Commissioning

It is now a question of attaching our drawing panel to the rest of the program. The drawing panel must be a MonObserser of the problemManager. When a problem is solved, the Draw panel should fetch the best map and make a call to draw Links

Also, add a call to**setMap(null)**and**setPopulation**, in the event handlers of the two buttons in the management panel.

10 - Bonuses

In the solution, we would vary the size of cities based on the number of links they are connected to. This gives a nice effect, since the very connected cities become more imposing. If you're looking for an extra challenge, try replicating this feature.