ITI 1121. Introduction to Computer Science II

Laboratory 4

Winter 2016

Plan

- Introduction to Graphical User Interfaces (GUIs)
- Exercises related to graphical user interfaces

Concepts: event-driven programming

- Any component can be the source of an event; a button generates an event when you click on it;
- Any class can be a listener for an event; it simply implements the method(s) of an interface.

  The object that handles the event generated by the click of a button needs to implement the interface **ActionListener**.

  The object that handles the event generated when a window is closed must implement the interface **WindowListener**;
- The event generated by the source component is sent to all the listeners who registered with the component.

  The source has a method **addActionListener** who needs as a parameter, an object that implements the interface **ActionListener**.

  Since the handler implements the interface **ActionListener**, the source knows that the handler has a method **actionPerformed(ActionEvent event)**.

1 Moving the red dot

Here is a simple application that has a display area and two buttons. When the left button is clicked the dot moves to the left; similarly for the right button. *Adapted from Decker & Hirshfield (2000) Programming.java*

1.1 Creating the graphical layout of the application

1.1.1 DisplayArea

Create a subclass of **JPanel**, called **DisplayArea**. "A JPanel component represents a blank rectangular area of the screen onto which the application can draw (...)". Later, we'll add the necessary implementation details so that a colored dot is displayed in the JPanel and moved to the left, right, top, and down whenever the user clicks on the proper buttons. Add a constructor that sets the size of the **DisplayArea** to 500 by 500 pixels.

1.1.2 GUI

Create a subclass of **JFrame**, called **GUI**. This will be the main window of the application.

- Add an instance variable of type **JButton** and initialize this with an object labeled “Left”;
- Add an instance variable of type **JButton** and initialize this with an object labeled “Right”;
- Add an instance variable of type **JButton** and initialize this with an object labeled “Up”;
- Add an instance variable of type **JButton** and initialize this with an object labeled “Down”;
- Add an instance variable of type **JComboBox** and initialize this with an object labeled “coloredList”;
- Add an instance variable of type **DisplayArea**.

Create a constructor that will add all the necessary graphical components to create the layout.

- Change the title with **setTitle** or call super constructor with the title as an argument
- Change the size with **setSize**
- Set the default reaction of Window regarding clicking on Exit button with **setDefaultCloseOperation(EXIT_ON_CLOSE)**. Please explore other options as well.
- Changing the background color, **setBackground(Color.WHITE)**;
- Add the **DisplayArea** object to the center of the **JFrame**;
- Create a **JPanel**;
- Initialize the defined instance variables, for example **bLeft = new JButton(”Left”);**
- Add colors ”red”, ”black”, ”blue”, ”green”, and ”yellow” into defined JComboBox.The constructor of JComboBox accepts an array of String as a parameter so you must create that array and fill it with the names of colors.
- Add the buttons and combobox to the **JPanel**;
- Add the panel to the **SOUTH** region of the **JFrame**;
- Add a main method that simply creates an instance of the class **GUI** and set its visibility to be true through the API **setVisible()**.

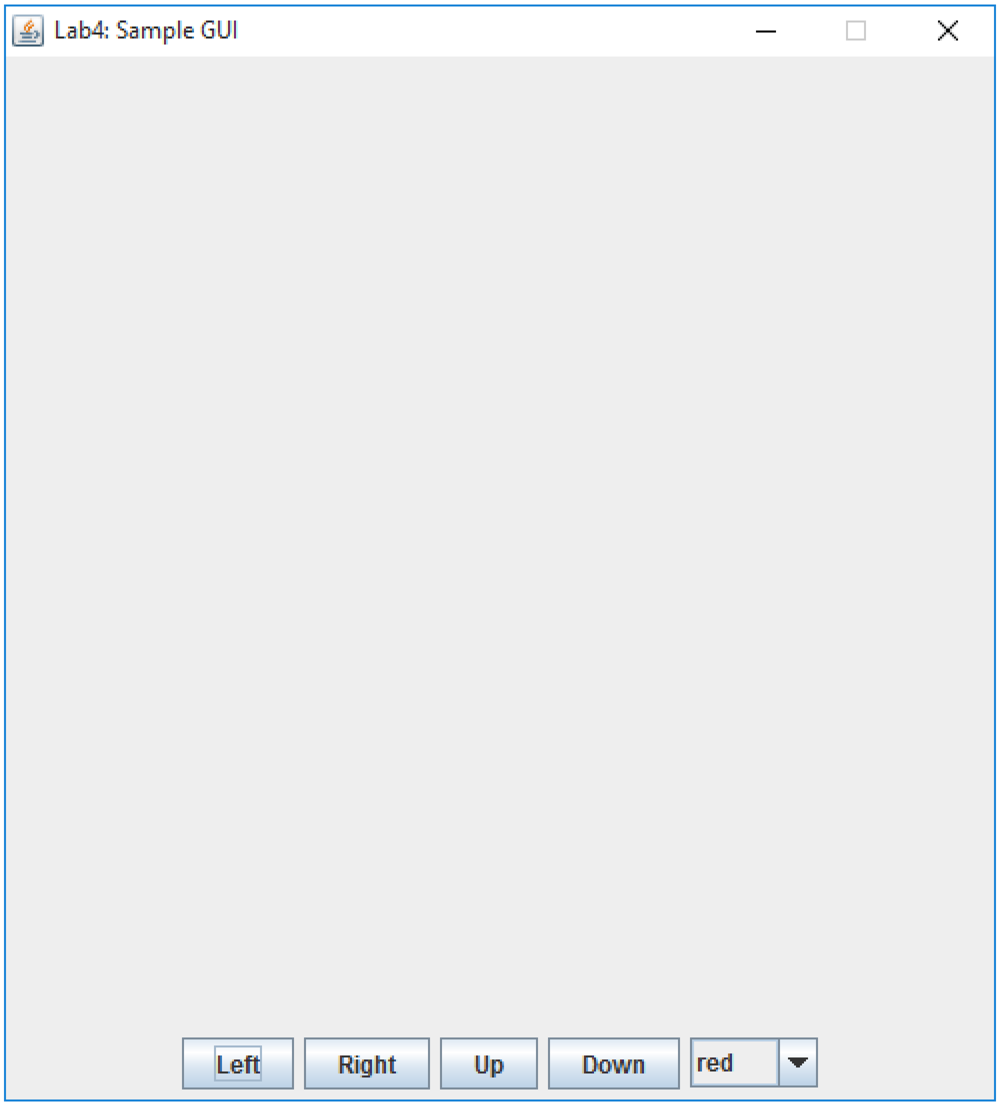
There are two ways to start this application. In a shell (dos/command window) type:

```
> java GUI
```

Or create a new instance of **GUI** in the interactions window of **DrJava**:

```
> GUI f = new GUI();
```

Your application should look something like this.

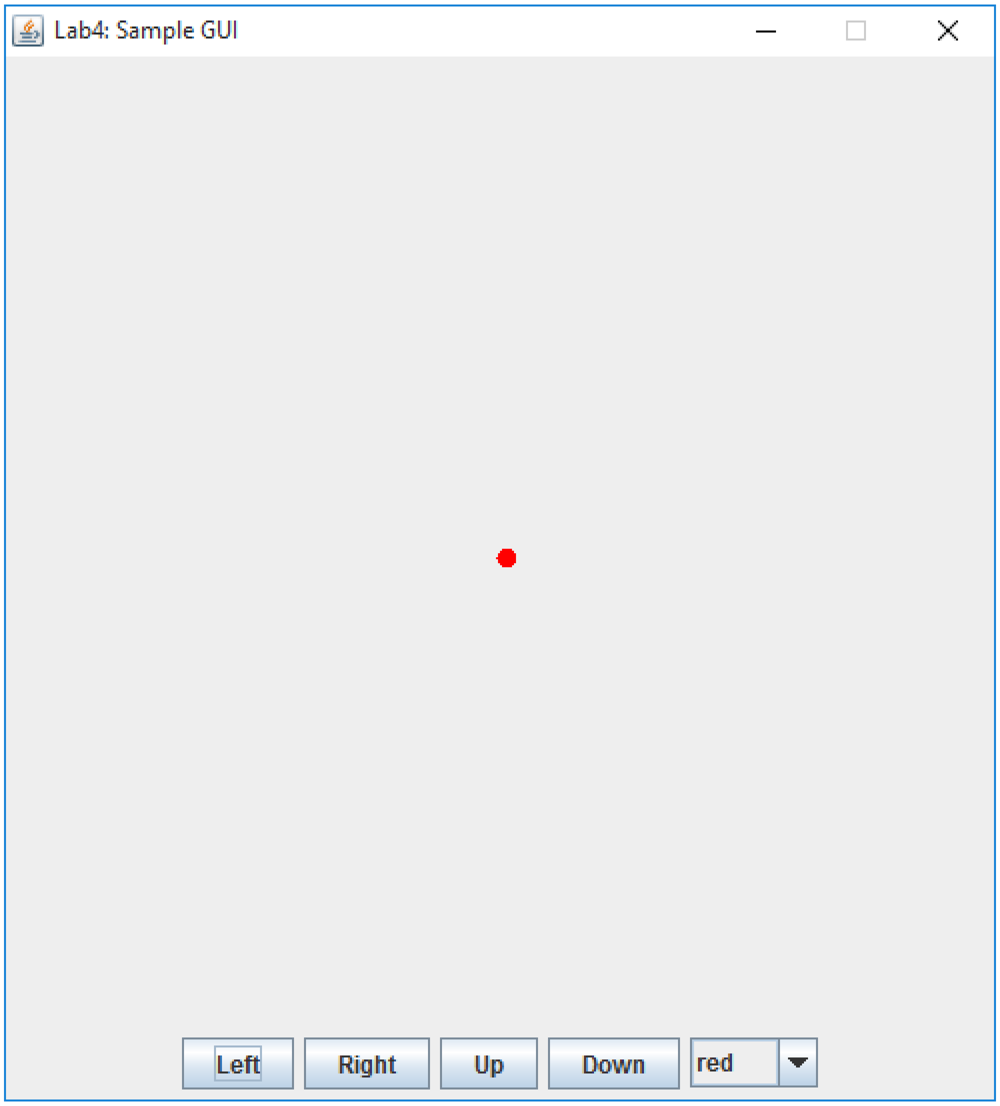


1.2 paint

Each time JPanel needs to paint a graphic it calls its method paint(Graphics g).

- Modify DisplayArea, add an instance variable of type Point called center;
- Modify the constructor and initialize center to “new Point(250, 250)”;
- Override the method paint(Graphics g). You will be painting onto the object designated by g. First set the default color to red, g.setColor(Color.RED), then paint a circle, of diameter 10, centered at center.x, center.y, g.fillOval(center.x - 5, center.y - 5, 10, 10).

You can now compile and test the application. You should see a red circle at the centre of the display area.



1.3 Action listeners

All that is left to do is making the dot move, to the left or to the right, when the user clicks onto the left or right button.

- First we need to implement the interface **ActionListener** by the class **DisplayArea**
- Then, it requires modifying the method actionPerformed of the classDisplayArea.

  First we need to determine which component was pressed. This can be done by calling the method **getSource()** from the class **ActionEvent**. You need to check the type by using **instanceof**

  Then , we should specify which button or color was selected. Methods **getActionCommand()** of the **ActionEvent** and **getSelectedItem** from **JCombobox** return the String used for this purpose respectively.
- If getActionCommand returns “Left”, then we will move the centre 10 pixels to the left;
- If getActionCommand returns “Right”, then we will move the centre 10 pixels to the right;
- If getActionCommand returns “Down”, then we will move the centre 10 pixels to the bottom;

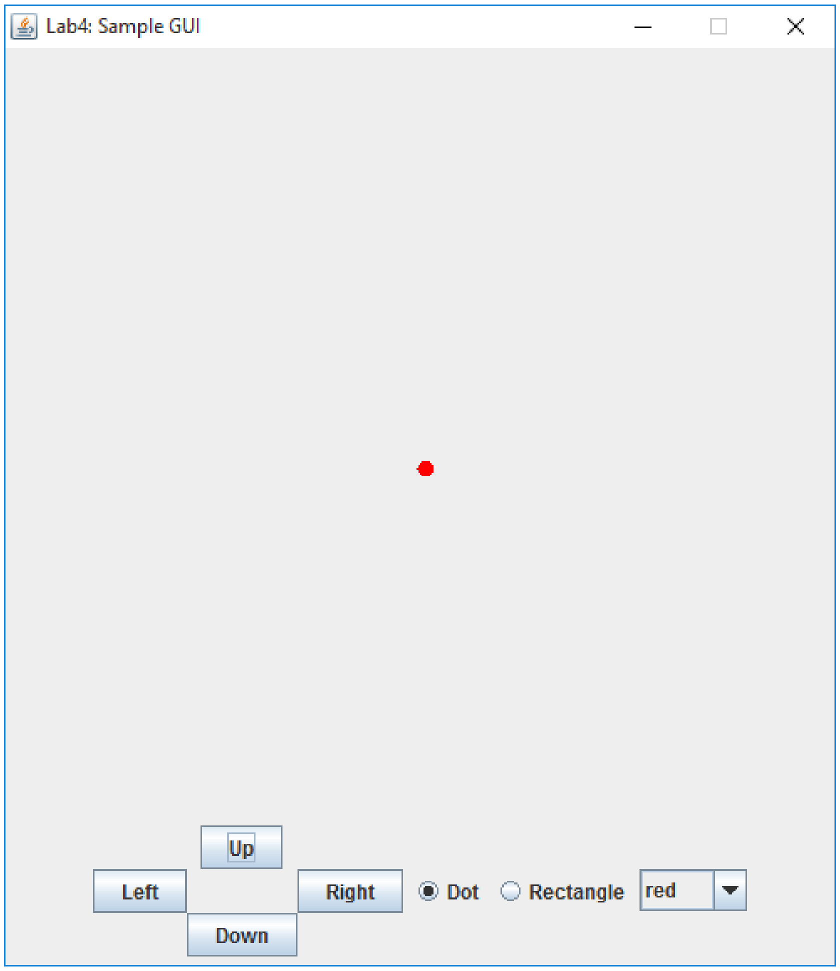
- If getActionCommand returns "Up", then we will move the centre 10 pixels to the top;
- If getSelectedItem returns the name of color, then we will change the color of Point
- We now want to force a call to our method paint(Graphics g), this can be done by calling the method repaint().
- Finally, you must add the class **DesktopArea** as an ActionListener to JButton and JComboBox components. For example, this is done through the following code for the JButton **Left**:

  bLeft.addActionListener(display);

You can now compile and test the application.

1.4 More features

In this subsection, we are going to have more organized User Interface and also be able to change the shape of Point. You can see an screenshot of the program in the following picture.



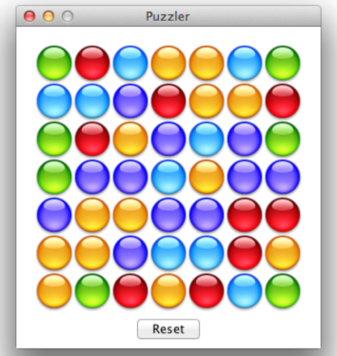
The component used are following:

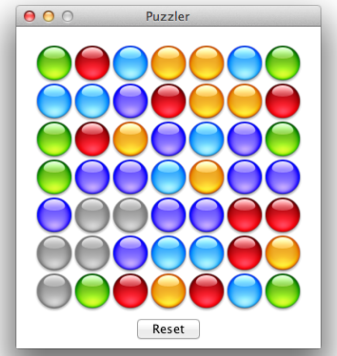
- **JRadioButton**: For selecting **Dot** and **Rectangle** in the program
- **GridBagConstraints**: For organizing JButtons

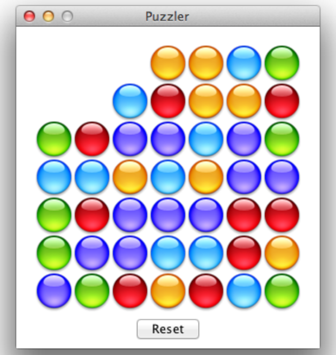
You should first get information about these components and learn how they react. This is part of lab because we would like you lean how to get details of components. Then, start writing your code to make the UI exactly like the screenshot above.

2 Puzzle game

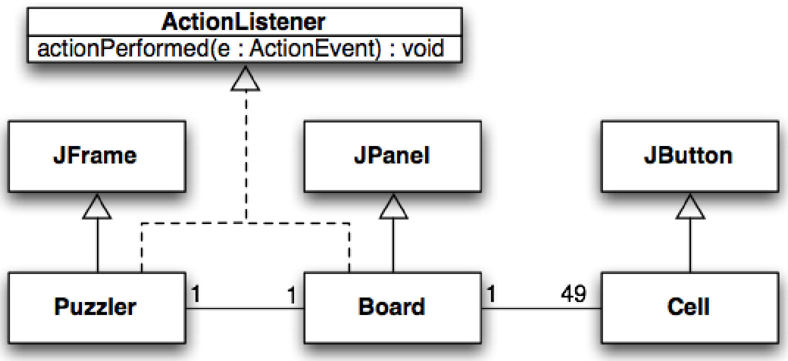
The game **Puzzler** consists of a 7×7 board. Initially, all the cells are filled with marbles from one the five available colors. When the user clicks on a cell for the first time, the cell and all the adjacent cells of the same color are selected. The marbles are gray to indicate this state of the game. When the user clicks a second time on a selected cell, all the selected cells vanish. Marbles fall from top to bottom, and left to right, in order to fill the empty spaces. To win the game, the user must make all the marbles vanish.







The implementation consists of three classes: Puzzler, Board, and Cell.

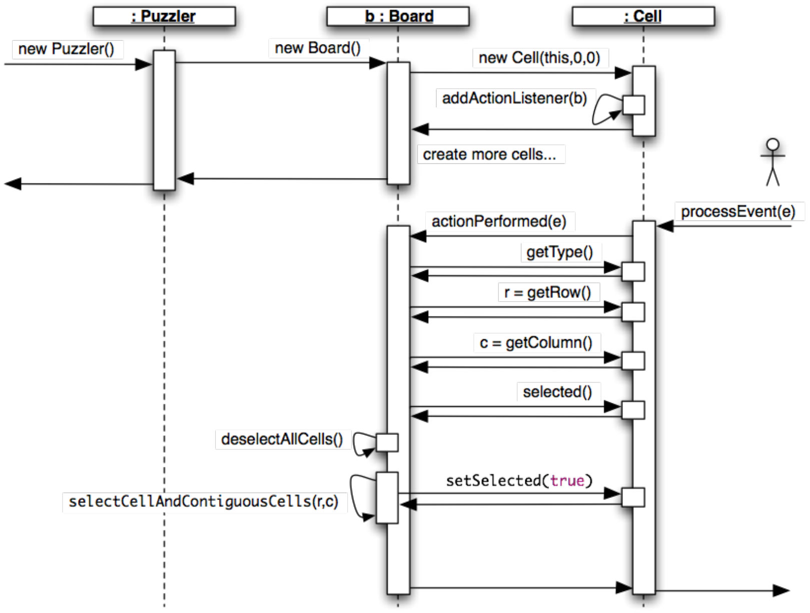


You can clearly see the important role of inheritance and interface for this assignment. Each of the three classes is derived from an existing class. The application needs a main window, which is called here **Puzzler**, and this is a subclass of **JFrame**. We need an object to model the grid onto which all the marbles will be placed. Since the object has access to all the marbles, it will also be responsible for implementing the logic of the game. The class **JPanel** will provide the means to display the marbles, this will be the class **Board**. Finally, the tiles of the **Board** game are implemented using objects of the class **Cell**, a subclass of **JButton**.

The key idea for this implementation is as follows. When a cell vanishes, it simply displays a white square. In order to simulate marbles falling from top to bottom, and left to right, we simply change the type of the source and destination cell. For instance, if a blue marble at position (i, j) moves to position (i', j'). We simply set the type of the cell at position (i, j) to represent the empty cell, whereas the type of the cell at position (i', j') becomes that of a blue marble.

The UML sequence diagram on the page illustrates some of the important sequences of method calls. When the constructor of the class **Puzzler** is called, it will create an object of the class **Board**. The constructor of the class **Board** will itself create **Cell** objects. The diagram shows one such object creation. The constructor of the class **Cell** receives a reference to the **Board** object, which will serve as an action listener for this **Cell** (call to addActionListener(b)).

The second part of the diagram illustrates a possible sequence of method calls resulting from a mouse click. When the user clicks the button, an object is created to represent this action, the method **processEvent** of the button is called. The button will then call the method **actionPerformed** of its action listener (the object that was registered using the method **addActionListener**, as above). The board will determine the source of the event. The method **actionPerformed** interacts with the cell to determine its type, row, and column. If the cell was not selected, all the cells are deselected, and all the adjacent cells are selected, this will include a class to the method **setSelected** of the selected cell, which changes the image to that of a gray marble. Eventually, the control returns to the caller.



The source code can be downloaded from here

3 Quiz (1 mark)

- Draw memory diagrams for all variables and objects created during the execution of the main method below.

```
class Singleton {
    private int value;
    Singleton( int value ) {
        this.value = value;
    }
}

class Pair {
    private Object first;
    private Object second;
    Pair( Object first, Object second ) {
        this.first = first;
        this.second = second;
    }
}

public class Quiz {
    public static void main( String[] args ) {
        Singleton s;
        Pair p1, p2;

        s = new Singleton( 99 );
        p2 = new Pair( s, null );
        p1 = new Pair( s, p2 );
    }
}
```

You can either use a pencil and paper approach or use a drawing/painting program. If you are drawing on a piece of paper, then you have to show your drawing to your TA who will record the information (your mark for the quiz). Alternatively, upload the drawing on the submission page for the laboratory on Balckboard Learn.

- https://uottawa.blackboard.com/

Last Modified: February 5, 2016