Lab 11 Requirements

Create a new Eclipse workspace named "**Lab11_1234567890**" on the desktop of your computer (replace **1234567890** with your student ID number). For each question below, create a new project in that workspace. Call each project by its question number: "**Question1**", "**Question2**", etc. If you do not remember how to create a workspace or projects, read the "*Introduction to Eclipse*" document which is on iSpace. Answer all the questions below. At the end of the lab, create a ZIP archive of the whole workspace folder. The resulting ZIP file must be called "**Lab11_1234567890.zip**" (replace **1234567890** with your student ID number). Upload the ZIP file on iSpace.

Question 1

Create a **MyFrame** class that extends **JFrame**. In the constructor of **MyFrame**, add all the necessary code to create a frame of size 400 x 300 that uses a border layout manager. Then, in the top part of the frame, add a **JPanel** that uses a centered flow layout manager and contains two buttons. The buttons do nothing for now.

Create a class **MyPanel** that extends **JPanel** and add a **MyPanel** object in the center part of the frame.

In the constructor of **MyPanel**, add a mouse listener to the panel using an anonymous class that extends **MouseAdapter**. Inside the **mouseClicked** method of the mouse listener, use the **getButton** method of the **MouseEvent** object to test when the left mouse button was clicked: compare the result of **getButton** against the constant **MouseEvent.BUTTON1** (left button) and then print a message indicating the position of the left button click in the panel. The position of the mouse can be obtained using the **getX** and **getY** methods of the **MouseEvent** object.

Then in the **main** method of the **Start** class, create an anonymous class that implements the **Runnable** interface with a **run** method that creates a **MyFrame** object, and use the **javax.swing.SwingUtilities.invokeLater** method to run that code on the event dispatch thread.

Run the program and click with the mouse here and there in the panel with different mouse buttons to check that your program works as expected. Also try clicking with the different mouse buttons on and around the two buttons in the top of the frame to see what happens.

Question 2

Add two private integer instance variables **x** and **y** to the class **MyPanel**, with **-1** and **-1** as initial values. Then modify the **mouseClicked** method of the mouse listener to store in **x** and **y** the coordinates of the mouse every time the left mouse button is clicked.

Override the **paintComponent** method of the **MyPanel** class to draw a red square of size **1** (to simulate a red point) at the position indicated by the values of the **x** and **y** fields. Do not forget to call **super.paintComponent** inside the **paintComponent** method to properly clean the panel before drawing anything new.

Run the program. Your program should print the correct position of the mouse every time you click the left mouse button in the center panel of the frame, but you will notice that no red point is drawn in the panel! This is because the **paintComponent** method of the center panel is only called when Swing decides that the content of the panel needs to be redrawn, and a mouse click is not enough for Swing to decide to do that. To see this, click the left mouse button inside the center panel of the frame, then resize the window. Suddenly a red point will appear at the position where you

clicked the mouse button! This is because resizing the window forces Swing to repaint it, which it turns makes Swing automatically call the **paintComponent** method of the **MyPanel** panel. The **paintComponent** method only then uses the values stored in the **x** and **y** fields to draw a single red point, which appears at the last position where the mouse was clicked (and therefore where the **mouseClicked** method was last called).

To force Swing to repaint the panel after every mouse click, add a call to the **repaint** method of the panel inside the **mouseClicked** method after **x** and **y** have been modified. This **repaint** method is inherited from **JPanel** and forces Swing to repaint everything every time it is called. This will force Swing to repaint the red point at the correct position every time the **mouseClicked** method processes a mouse click of the left button in the panel.

Run the program. Check that the red point is always drawn at the correct position whenever you click the left mouse button in the center panel.

Question 3

We now want to always redraw all the points that have ever been drawn in the panel, not just the last point. To do this, we must save the coordinates of all these points so that we can redraw them all one by one in the **paintComponent** method every time this method is called.

To save the coordinates of the various mouse positions we click, replace the **x** and **y** instance variables of the **MyPanel** class with a single private instance variable called **points** of type **ArrayList<Point>**. The **Point** class is provided to you by Swing. In the constructor of **MyPanel**, initialize the **points** instance variable with a new arraylist object of the same type.

In the **mouseClicked** method of the mouse listener, use the **getPoint** method of the mouse event object to get a **Point** object representing the position of the mouse click (that **Point** object internally stores both the **x** and **y** coordinates of the mouse click event). Then add this **Point** object to the arraylist using the arraylist's **add** method.

Then, in the **paintComponent** method, add a loop to draw in the panel all the points of the arraylist. You can get the number of elements in the arraylist by using the **size** method of the arraylist; you can access a specific element of the arraylist at index **i** by using the **get(i)** method of the arraylist (element indexes start at zero in an arraylist). The **Point** class has **getX** and **getY** methods to get the coordinates of the point (these two methods return values of type **double** so you need to cast the returned values into the **int** type before you can use them to draw a point).

Question 4

When running the program of the previous question, you will notice that the result is not very exciting since you get different points drawn without any relation between them. Therefore change the loop in the **paintComponent** method so that it draws lines from one point to the next, instead of just drawing independent points.

Make sure your code still works correctly when only one point is stored in the arraylist, in which case only a single point must be printed.

Question 5

Add a public **clearAllPoints** method to **MyPanel** that uses the **clear** method of the arraylist to remove all the **Point** objects in the arraylist. Then add an anonymous action listener to the first button in the constructor of **MyFrame** so that this action listener calls the **clearAllPoints** method of the center panel in the frame when the first button is clicked. Change the text of the first button to be "**Reset**".

Note: when you run the program and use the "**Reset**" button, the drawing will be erased only when you start drawing something else! Again this is because Swing does not automatically redraw the content of the panel every time the content of the arraylist changes. To fix this, add a call to the **repaint** method at the end of the **clearAllPoints** method of **MyPanel**, to force Swing to repaint the panel as soon as the "**Reset**" button is clicked.

Question 6

Add a public **undoPoint** method to **MyPanel** that uses the **remove** method of the arraylist to remove the last point in the arraylist. The last point in the arraylist is the point that was last added so removing the last point undoes the last point addition. Again, this method should call the **repaint** method after it is done removing the last point. Then add an anonymous action listener to the second button in the constructor of **MyFrame** so that this action listener calls the **undoPoint** method of the center panel in the frame when the second button is clicked. Change the text of the second button to be "**Undo**".

Make sure the undo feature works even when no point is drawn.