HW 03: Creating Graphical User Interfaces (GUI) in Java

Due 8:00 AM Friday, Oct 23, 2020

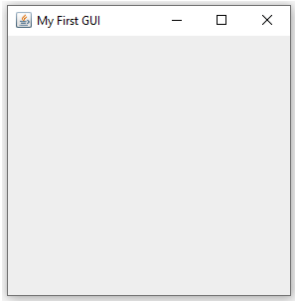
Part I: Hand-coded Java GUI Tutorial

Step 1) Create a new window

Using Eclipse, Create a new Java Project, Add a new class and name it `GuiTest` with a `main` method, then type the following code:

```
01.     import javax.swing.*;
02.     public class GuiTest{
03.             public static void main(String args[]){
04.                     JFrame frame = new JFrame("My First GUI");
05.                     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
06.                     frame.setSize(300,300);
07.                     frame.setVisible(true);
08.             }
09.     }
```

Step 2) Save, Compile and Run the code.



Verify that you get a square window the tile "My First GUI", a minimize button, a maximize button and a close button.

Observe:

- The program you just wrote consists of a simple class with a `main` method which was able to create a graphical window because it created an object of type `JFrame`.
- `JFrame` is a class that was coded to draw a window; nevertheless it is a bona fide java class with method, properties, constructor, etc.

Answer the following questions:

- What does the constructor of `JFrame` look like? (Write the formal method signature of the constructor that was used.

  public Jframe(String title) throw Headless Exception

1- What is the parameter of this specific constructor used for?

   title of header in gui

2- What is the nature of `JFrame.EXIT_ON_CLOSE`? (is it a method, field,…). Write down its formal declaration line

   Method Frame setDefaultCloseOperation(Jframe EXIT_ON_CLOSE)

- Find out which method of `JFrame` changes the cursor displayed when the mouse is over the window. Write down the line you would use to change the cursor to an hourglass.
  __Cursor hourglassCursor= new Cursor(Cursor.WAIT_CURSOR);
  Cursor.setCursor(hourglassCursor)

Step 3) Add a Button to the window:

Add the following two lines after the declaration and initialization of frame:

```
JButton button1 = new JButton("Press");
frame.getContentPane().add(button1);
```

Step 4) Execute the code. You will get a big button

Verify that a button labeled "Press" occupies the whole window.

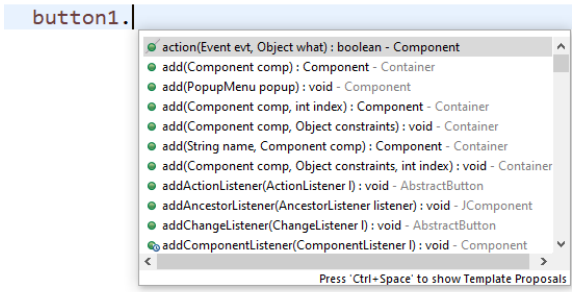
Observe:

Adding a button consists of creating an object of type `JButton`, then adding that object to the content pane.

- We did not specify the size of the button, so it filled the window.
- Pressing the button does nothing so far.

Answer the following questions:

On a new line below the declaration of `button1`, type `button1` then press the dot . key to ask Eclipse to display the public members of the class `JButton`, then choose the method that sets the **text** of the button. Write the line of code required to change the text of the button to "Press Me"



Answer: button1.setText("Click me");

1- Type `button` then press the dot . key and choose the method that sets the **tooltip** of the button. Write the line of code required to change the tooltip of the button to "Press me and watch what happens"

Answer: __button1.setToolTipText("Press me and watch what happens");

Step 5) Add another button to the same window:

Add the following two lines before the `frame.setVisible` line :

```
JButton button2 = new JButton("Button 2");
frame.getContentPane().add(button2);
```

Step 6) Save, Compile and Run the code.

Observe:

- Adding a second button also filled the window and thus concealed the first button. We need a way to display both buttons.

Step 7) Use a layout manager to display the two buttons.

Replace the two lines that add the buttons to the content pane by these two lines (you will need to add the right import statement):

```
frame.getContentPane().add(BorderLayout.NORTH,button1);

frame.getContentPane().add(BorderLayout.SOUTH,button2);
```

Step 8) Save, Compile and Run the code.

Observe:

- The two buttons are now bound to the top and bottom of the window.

Answer the following questions:

2- Change the position of the two buttons to make them bound to the left and right side of the window (west, east) instead of up and down (north, south). Write the two lines of code required to do so:

frame.getContentPane().add(BorderLayout West,button1)

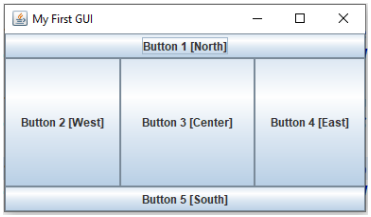
frame.getContentPane().add(BorderLayout East,button2)

A Note On Layout Managers

A layout manager is used to *layout* (or *arrange*) the GUI components inside a container.

There are many layout managers, but the most frequently used are:

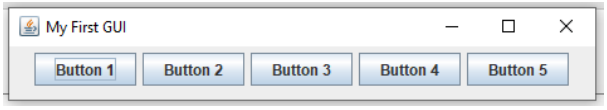
BorderLayout



A `BorderLayout` places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every `JFrame`

FlowLayout

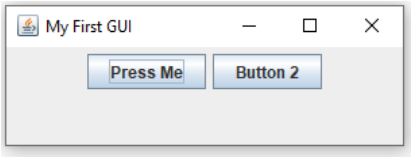
`FlowLayout` is the default layout manager for every `JPanel`. It simply lays out components in sequence one after another, so depending on the size of the window you end up with the following:



GridBagLayout

It is the most sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell.

Step 8) Change the layout of the content pane to a `FlowLayout`



- Set the layout of the content pane right before you add the buttons to it:

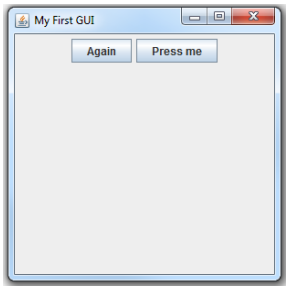
```
frame.getContentPane().setLayout(new FlowLayout());
```

Step 9) Save, Compile and Run the code.

Observe:

- The two buttons are now displayed from left to right starting from the top of the window.
- Change the order of the two lines that add the buttons and notice how the buttons have swapped positions

Step 10) Adding Event Handlers



Two add event handlers you need to sub-class the `JFrame` class:

- Add a new class `MyFrame` to your project
- Make `MyFrame` inherit the `JFrame` (you'll need to import `javax.swing.JFrame`)
- Declare two `JButtons` inside the class `button1`, and `button2`
- Create an overloaded constructor for `MyFrame`
- Since it inherits from `JFrame` which has an overloaded constructor that takes a string and displays it in the caption area of the window, make the overloaded constructor of `MyFrame` match that of `JFrame`

- Inside this constructor pass the string to the constructor of the base class. Hint: constructor chaining, use the `super` keyword.

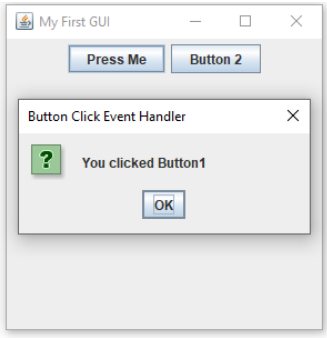
- Cut the code from your `main` method in `GuiTest` and paste it in the overloaded constructor you have just created.
- Here you no longer need to create an instance of `JFrame` because you are inside the constructor of a `JFrame`. Therefore, remove the line that declares an initializes a `JFrame` , then replace all references to `frame` with the `this` keyword.
- We need to reference `button1` and `button2` outside of the constructor that is why we declared them outside the constructor and gave them class-wide scope. Therefore, change the two declaration-and-initialization statements to initialization only statements (remove the `JButton` at the beginning of each line).
- Back in `GUITest.main` create an instance of `MyFrame`, that should be sufficient to see the window.
- Run your program and make sure you are seeing a new window. Note that pressing any of the two buttons does not generate any action yet.
- Now add the event handlers to the buttons:
- Make `MyFrame` implement the interface `ActionListener` (you'll need to import `java.awt.event.ActionListener`)
- In the Eclipse editor, hover your mouse over the `MyFrame` and choose the "Add unimplemented methods" option. This will add the overrides you need to write (`actionPerformed`). The code inside the method will run any time a control generates an event.
- Add the following code to the method:

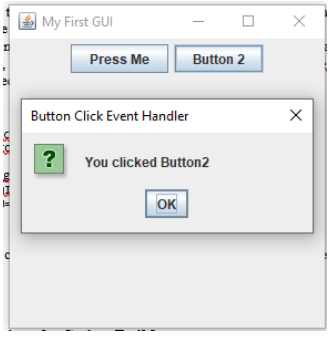
```
@Override
public void actionPerformed(ActionEvent e) {
	JButton b = (JButton) e.getSource();

	if (b==button1)
	{
		JOptionPane.showConfirmDialog(this, "You clicked Button1","Button Click Event Handler", JOptionPane.PLAIN_MESSAGE);
		}else if (b==button2)
		{
			JOptionPane.showConfirmDialog(this, "You clicked Button2","Button Click Event Handler",
JOptionPane.PLAIN_MESSAGE);
		}


	}
```

- Now make the buttons hook to the event handler of the `JFrame`:
- Add the following two lines to the end of the constructor

```
button1.addActionListener(this);

button2.addActionListener(this);
```

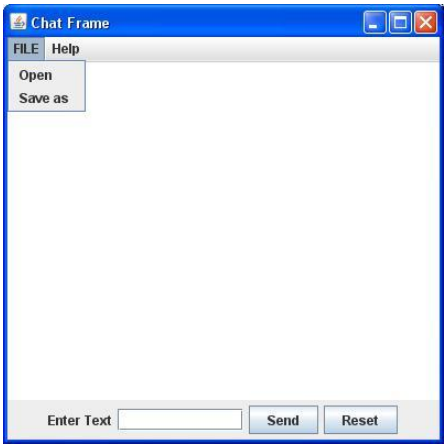
- Run your code and test it.





Part II: Creating your first GUI application

Write code (don't use any GUI builders) to implement the following interface:



- To add a menu, use the `JMenuBar, JMenu,` and `JMenuItem` classes

```
JMenuBar mb = new JMenuBar();
JMenu m1 = new JMenu("FILE");
mb.add(m1);
JMenuItem m11 = new JMenuItem("Open");
m1.add(m11);
this.setJMenuBar(mb);
```

- You need to add the event handlers to each `JMenuItem` (hint: they need to have class-wide scope)
- In the event handler you now need to know the type of object that triggered the event, prior to now we assumed it was always a `JButton`, but now it could be either a `JButton` or a `JMenuItem`. If it is a menu click then display the text of the menu item that was clicked.
- Hints:

```
if (e.getSource() instanceof JButton) {
        JButton b = (JButton) e.getSource();
        ….
} else if (e.getSource() instanceof JMenuItem) {
        JMenuItem m = (JMenuItem) e.getSource();
        if (m==m11)
        …
```

- Run your code and ensure the application is behaving as expected (test it).

Submission Instructions:

1- Create a private GitHub repository using your name
2- Upload your code and this filled-out document to the repository you've just created
3- share the repo with the professor and the TAs