

AP Computer Science A

Swing UI Designer

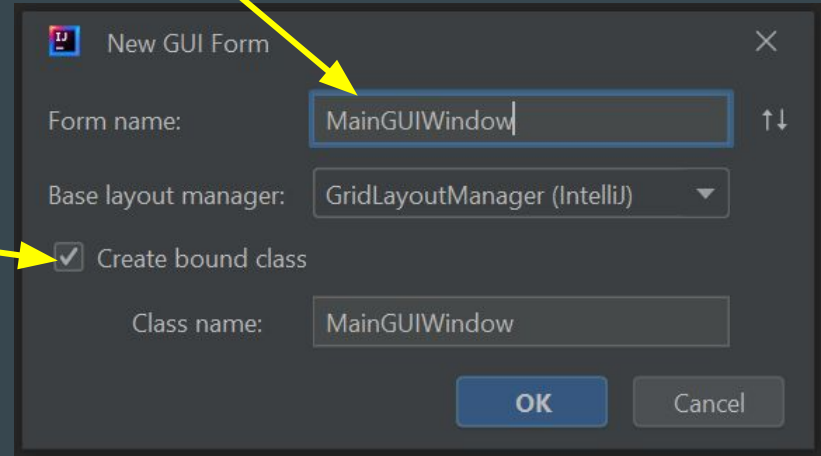
Bonus Topic!
(Not on the AP Exam)

For Building Complex GUIs!

May 1, 2023

Do Now!

1. Open up a new IntelliJ project, LastNameSwingUIDemo
2. Create a `Main.java` file (if not created for you automatically)
3. Go to "File → New → Swing UI Designer → GUI Form"
4. Name the Form something, such as `MainGUIWindow`
5. Make sure the checkbox is checked so a class is automatically created



Coding GUIs

Getting a GUI to look the way you want it to look is difficult, tedious, and annoying!

Whether you're creating a display using graphics, GUIs, HTML, there are tools that can help us create the layout we want.

Today we will learn about the **Swing UI Designer** in IntelliJ

You can use this tool to design your Final Project GUI

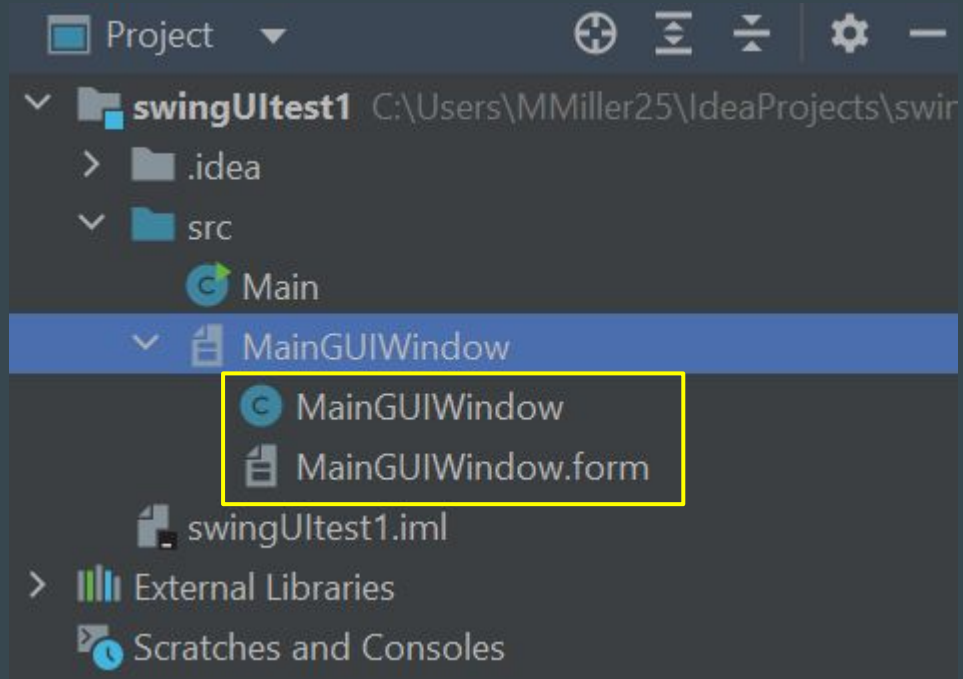
Today will be all about experimenting and creating your own GUIs!

Forms

A "form" is used to design a GUI using the drag-and-drop **Swing UI Builder**, rather than typing everything in code.

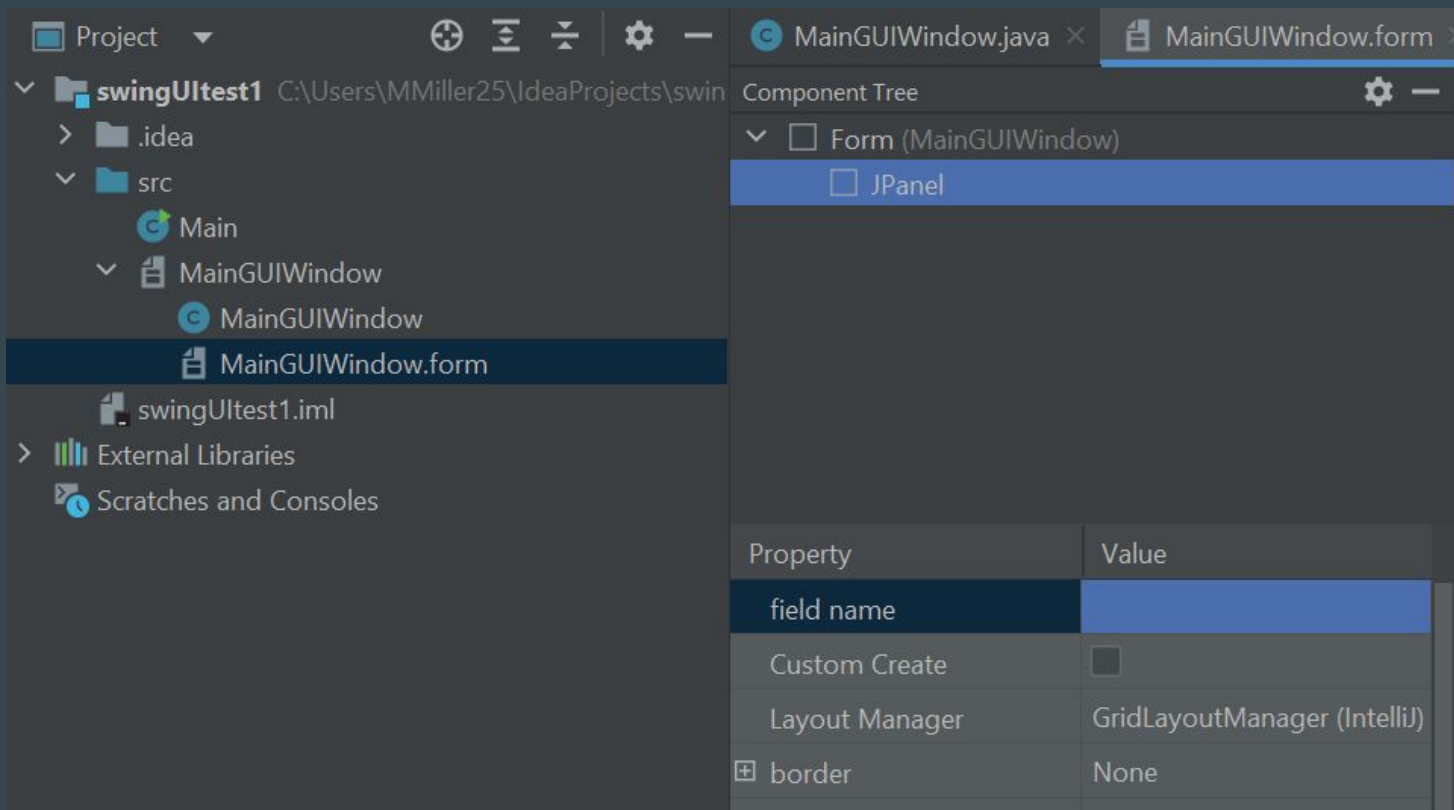
A form gets "connected" to a class; any component you add using the designer will automatically get added to your class file.

The "MainGUIWindow" class is going to extend JFrame (as we will see in a second)



Step 1

Open the form and you will see there is a JPanel already added

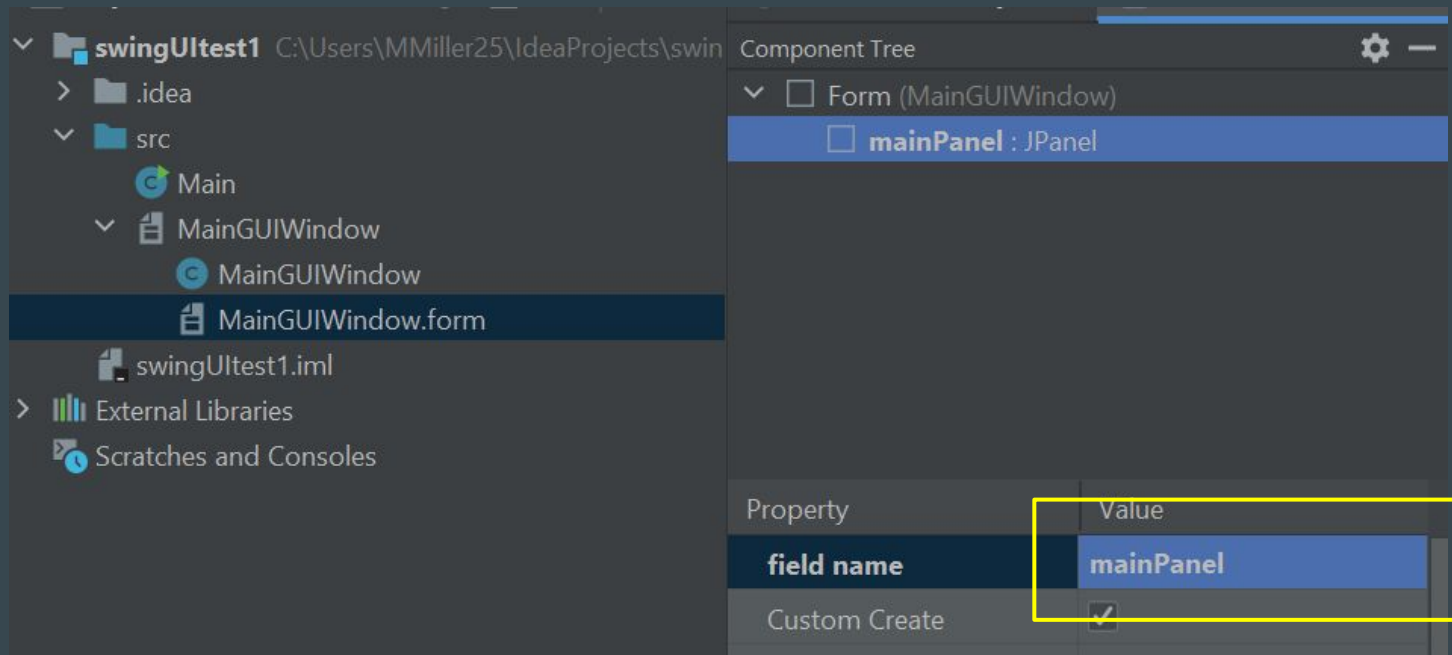


The screenshot shows the IntelliJ IDEA interface. On the left, the 'Project' view displays the file structure of 'swingUtest1'. The 'MainGUIWindow.form' file is selected. On the right, the 'Component Tree' shows a 'Form (MainGUIWindow)' containing a 'JPanel'. Below the component tree is a table with properties and values.

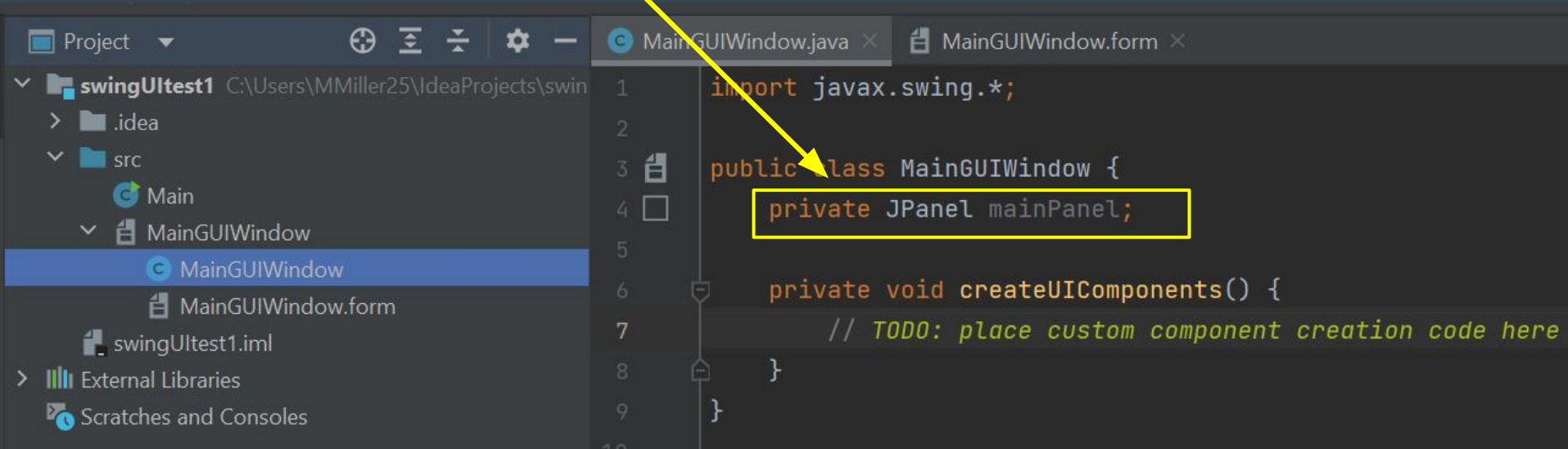
Property	Value
field name	
Custom Create	<input type="checkbox"/>
Layout Manager	GridLayoutManager (IntelliJ)
border	None

Step 2

Enter a name
for it in the
"field name",
such as
mainPanel

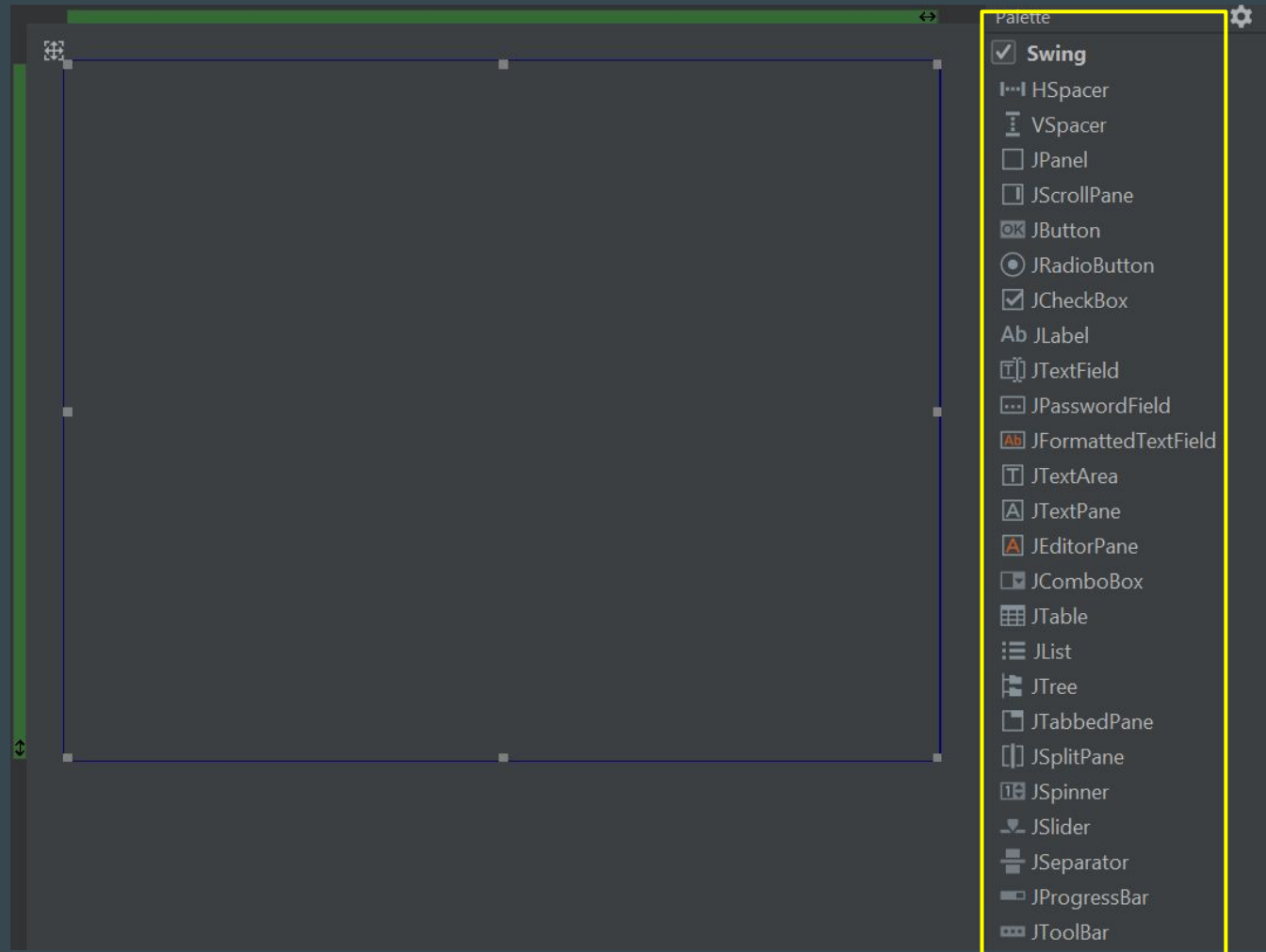


Now, if you go to your `SwingWindow.java` file, you will see the code for the main panel; when you add a component using the UI Designer, it *automatically* gets added as an instance variable in the form's class! The form and the class java file are linked behind the scenes by IntelliJ



Step 3: Using the Palette

The component palette on the right allows you to drag and drop components onto your main panel. This includes other panels, buttons, check boxes, sliders, etc.



Step 4: Make your GUI: drag and drop!

The screenshot shows a Java Swing IDE interface. On the left is a component palette with a tree view containing:

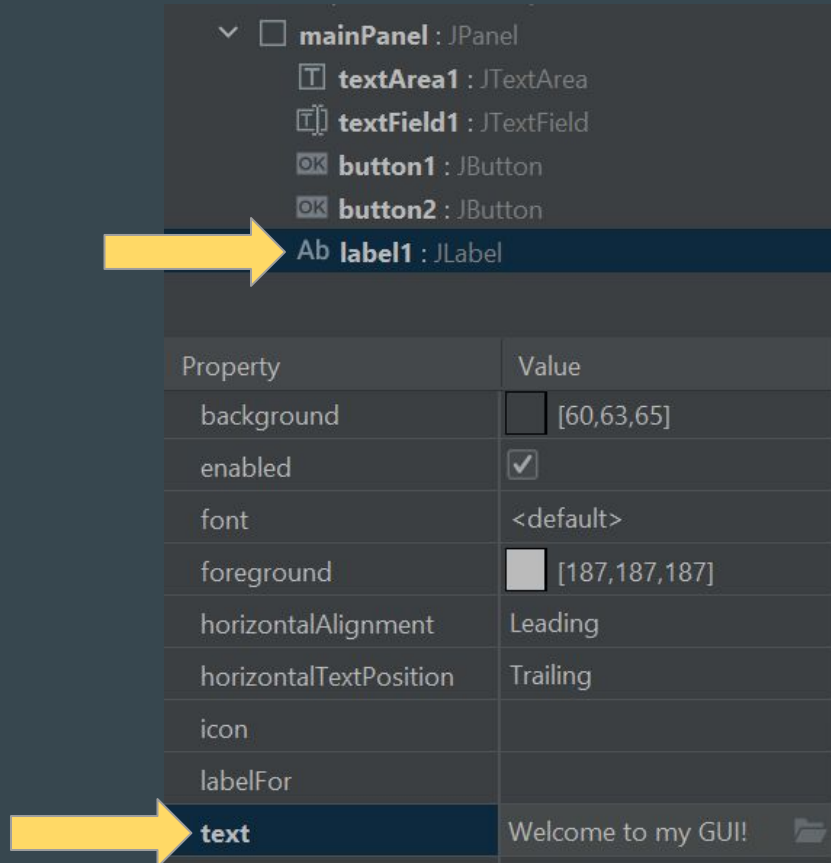
- Form (MainGUIWindow)
 - mainPanel : JPanel
 - myTextArea : JTextArea
 - textField1 : JTextField
 - button1 : JButton
 - label1 : JLabel** (highlighted)
 - button2 : JButton

Below the tree is a table of properties for the selected component:

Property	Value
field name	label1
Custom Create	<input type="checkbox"/>
Horizontal Size Policy	Fixed
Vertical Size Policy	Fixed
Horizontal Align	Center
Vertical Align	Center
Indent	0
Minimum Size	[-1, -1]
Preferred Size	[-1, -1]
width	1

On the right is a visual representation of the GUI. It features a title bar with a lightbulb icon and the text "Welcome to my GUI". Below the title bar is a text input field and a "Send" button. The main area of the window contains a large text area labeled "myTextArea:JTextArea" (highlighted in blue). At the bottom of the window is a button labeled "Click me for a good time!".

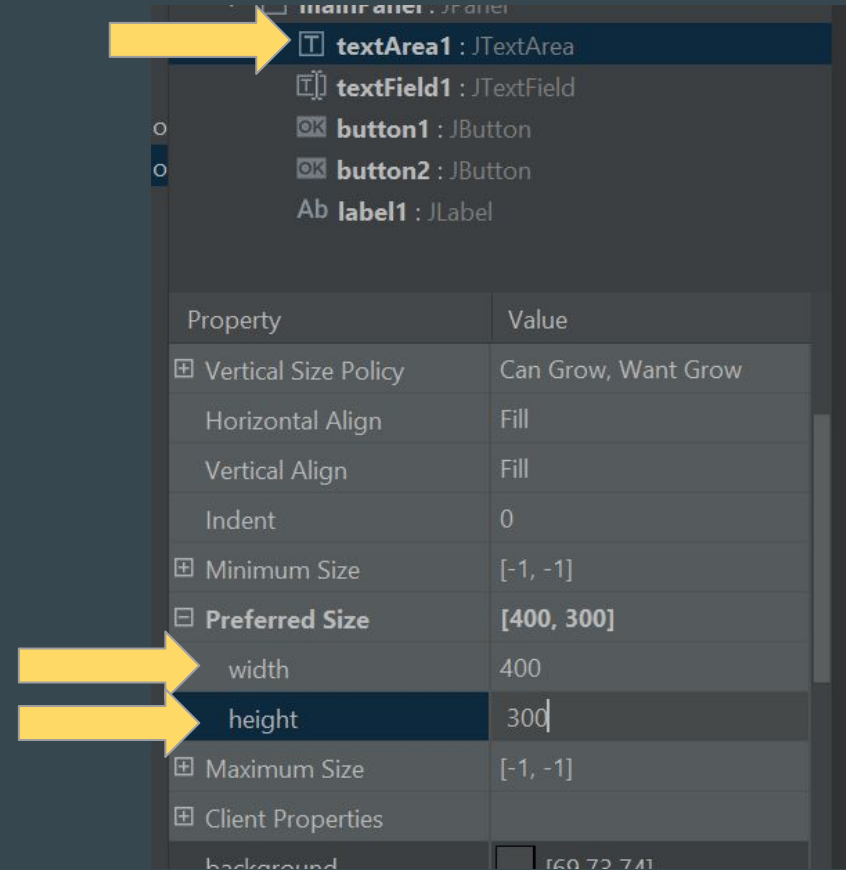
Adjust default settings as you need



mainPanel : JPanel

- textArea1 : JTextArea
- textField1 : JTextField
- button1 : JButton
- button2 : JButton
- label1 : JLabel**

Property	Value
background	[60,63,65]
enabled	<input checked="" type="checkbox"/>
font	<default>
foreground	[187,187,187]
horizontalAlignment	Leading
horizontalTextPosition	Trailing
icon	
labelFor	
text	Welcome to my GUI!



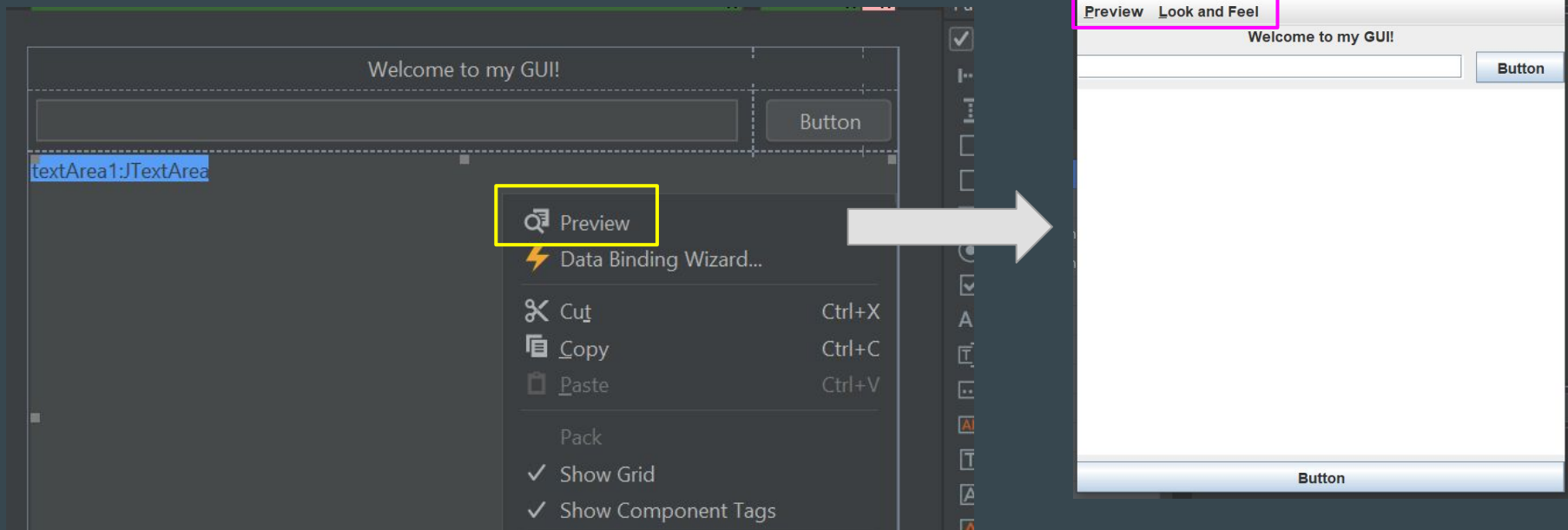
mainPanel : JPanel

- textArea1 : JTextArea**
- textField1 : JTextField
- button1 : JButton
- button2 : JButton
- label1 : JLabel

Property	Value
Vertical Size Policy	Can Grow, Want Grow
Horizontal Align	Fill
Vertical Align	Fill
Indent	0
Minimum Size	[-1, -1]
Preferred Size	[400, 300]
width	400
height	300
Maximum Size	[-1, -1]
Client Properties	
background	[60,63,65]

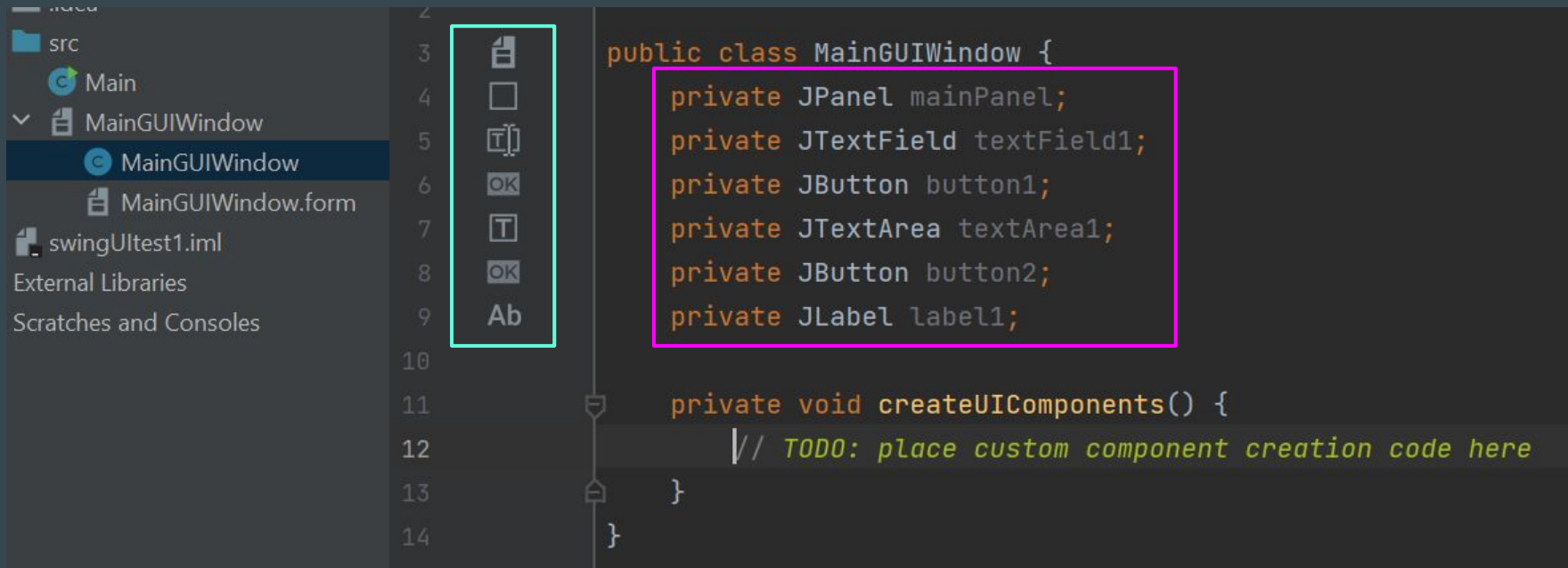
Step 5: Previewing

Right click on the form and choose Preview; there are a couple of **preview options menus** that IntelliJ adds in Preview Mode only, but these won't be in your final GUI

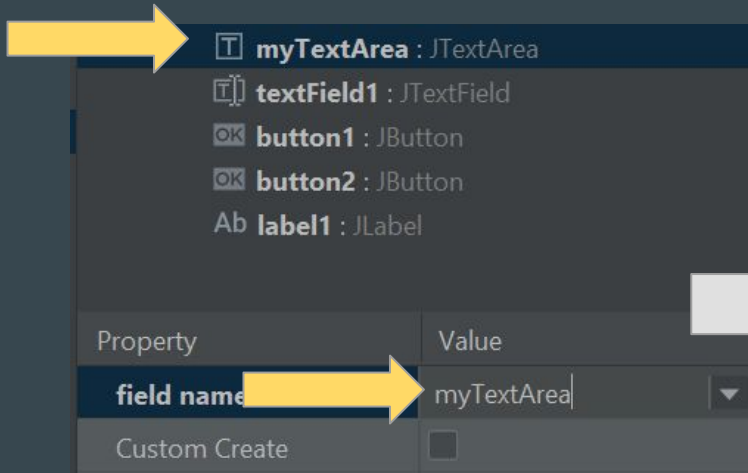


All components are added as instance variables automatically!

The icons on the left indicate the linkage between the code and the form.



Change the default component names if you want to



```
public class MainGUIWindow extends JFrame {  
    private JPanel mainPanel;  
    private JTextField textField1;  
    private JButton button1;  
    private JTextArea myTextArea;  
    private JButton button2;  
    private JLabel label1;  
}
```

Step 6: Add code to display the GUI

Extend JFrame

Add a constructor, and have it call the helper method

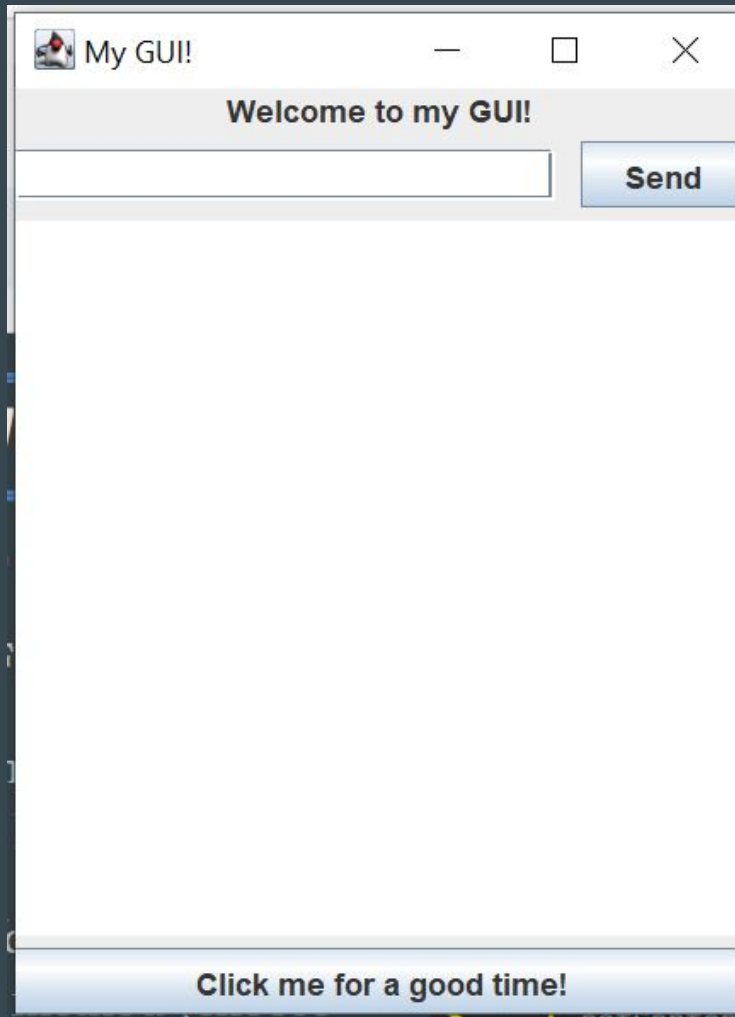
Add the following code in the setup method (choose values for size and location); note the setContentPane method should pass the main panel (whatever you named it)

```
public class MainGUIWindow extends JFrame {  
    private JPanel mainPanel;  
    private JTextField textField1;  
    private JButton button1;  
    private JTextArea myTextArea;  
    private JButton button2;  
    private JLabel label1;
```

```
    public MainGUIWindow() {  
        createUIComponents();  
    }
```

```
    private void createUIComponents() {  
        setContentPane(mainPanel);  
        setTitle("My GUI!");  
        setSize(300, 400);  
        setLocation(450, 100);  
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
        setVisible(true);  
    }
```

Step 7: Run and see it!



BUT WAIT. Don't we need to initialize all of these component in our code?! Where do we create each object?!

```
public class MainGUIWindow extends JFrame {  
    private JPanel mainPanel;  
    private JTextField textField1;  
    private JButton button1;  
    private JTextArea myTextArea;  
    private JButton button2;  
    private JLabel label1;  
  
    public MainGUIWindow() {  
        createUIComponents();  
    }  
  
    private void createUIComponents() {  
        setContentPane(mainPanel);  
        setTitle("My GUI!");  
        setSize(300, 400);  
        setLocation(450, 100);  
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
        setVisible(true);  
    }  
}
```


If we *don't* use the UI Swing Designer*, we would have written our code something like this

* what we have done up to now

We have to *manually* create each panel and component, add components to the panel(s), then add panel(s) to the frame (ugh)

We have to *manually* figure out a layout that works and get things to look nice on the screen (ugh)

```
private JTextField textField1;
private JButton button1;
private JTextArea myTextArea;
private JButton button2;
private JLabel label1;

public MainGUIWindow2() {
    createUIComponents();
}

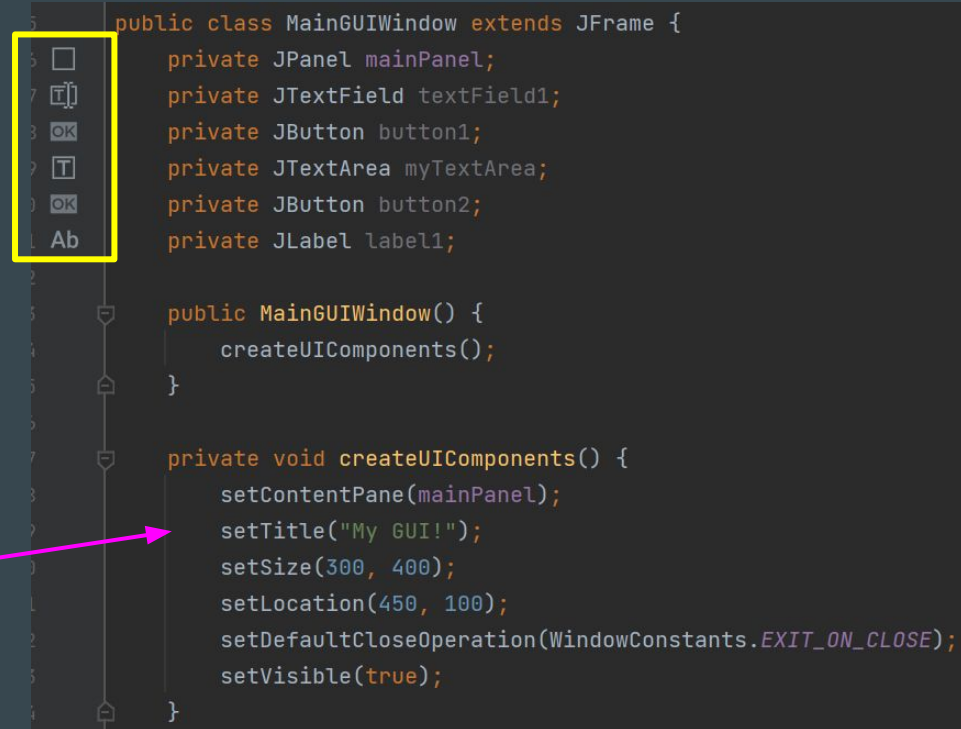
private void createUIComponents() {
    setTitle("My GUI!");
    setSize(300, 400);
    setLocation(450, 100);
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

    mainPanel = new JPanel();
    label1 = new JLabel("Welcome to my GUI!");
    mainPanel.add(label1);
    textField1 = new JTextField();
    mainPanel.add(textField1);
    myTextArea = new JTextArea();
    mainPanel.add(myTextArea);
    button1 = new JButton("Send");
    mainPanel.add(button1);
    button2 = new JButton("Click me for a good time!");
    mainPanel.add(button2);
    mainPanel.setLayout(new GridLayout(3, 1));
    add(mainPanel);
    setVisible(true);
}
```

If we *do* use the UI Swing Designer, the components *automatically* get initialized when you make a MainGUIWindow using the default settings that you set in the UI designer!

Any GUI component instance variable that is "linked" to the form will be initialized *automatically*!

The only setup code we need is some initial values for the main frame; IntelliJ creates the component objects and generates the layout from the form!



```
1 public class MainGUIWindow extends JFrame {  
2     private JPanel mainPanel;  
3     private JTextField textField1;  
4     private JButton button1;  
5     private JTextArea myTextArea;  
6     private JButton button2;  
7     private JLabel label1;  
8  
9     public MainGUIWindow() {  
10         createUIComponents();  
11     }  
12  
13     private void createUIComponents() {  
14         setContentPane(mainPanel);  
15         setTitle("My GUI!");  
16         setSize(300, 400);  
17         setLocation(450, 100);  
18         setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
19         setVisible(true);  
20     }  
21 }
```

Step 8: Adding event listeners

We can add **event listeners** to our components just as before (this is done in code)

1. implement the listener interface you want

```
public class MainGUIWindow extends JFrame implements ActionListener, KeyListener {
```

Step 8: Adding event listeners


2. implement the required interface method(s)

```
@Override  
public void actionPerformed(ActionEvent e)
```

```
@Override  
public void keyTyped(KeyEvent e)
```

```
@Override  
public void keyPressed(KeyEvent e)
```

```
@Override  
public void keyReleased(KeyEvent e)
```



All 3 of these methods are required by the `KeyListener` interface, even if you leave some of them empty (which is OK to do)

Step 8: Adding event listeners

3. add the listeners to the components that we want to interact with

```
private void createUIComponents() {  
    setContentPane(mainPanel);  
    setTitle("My GUI!");  
    setSize(300, 400);  
    setLocation(450, 100);  
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
    button1.addActionListener(this);  
    button2.addActionListener(this);  
    textField1.addKeyListener(this);  
    setVisible(true);  
}
```

Step 8: Adding event listeners

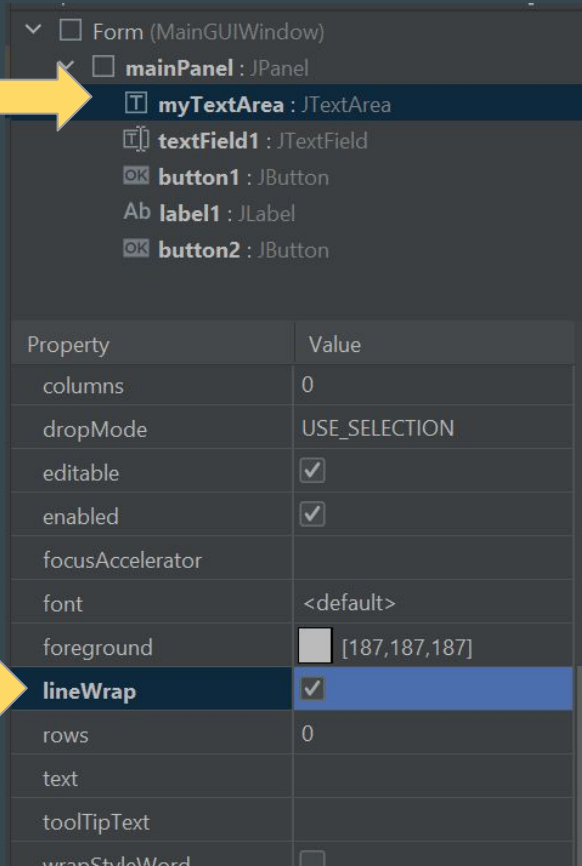
4. make sure your code is written to differentiate between objects of the same type, if necessary

We have two different JButtons that we are listening for, and both trigger this method, so we need to find out which button was pressed

```
@Override
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source instanceof JButton) {
        JButton button = (JButton) source;
        String text = button.getText();

        if (text.equals("Send")) {
            myTextArea.append("button 1 clicked! ");
        } else {
            myTextArea.append("button 2 clicked! ");
        }
    }
}
```

Adjust other settings as you need in the UI Designer:



The screenshot shows the Java Swing UI Designer interface. In the component hierarchy on the left, 'myTextArea : JTextArea' is selected, indicated by a yellow arrow. Below the hierarchy is a table of properties and their values for the selected component.

Property	Value
columns	0
dropMode	USE_SELECTION
editable	<input checked="" type="checkbox"/>
enabled	<input checked="" type="checkbox"/>
focusAccelerator	
font	<default>
foreground	<input type="color" value="#CCCCCC"/> [187,187,187]
lineWrap	<input checked="" type="checkbox"/>
rows	0
text	
toolTipText	
wrapStyleWord	<input type="checkbox"/>

A yellow arrow points to the 'lineWrap' property, which is currently checked.

You could *alternatively* do this in code if you want to (but the UI designer is easier!)



```
private void createUIComponents() {  
    setContentPane(mainPanel);  
    setTitle("My GUI!");  
    setSize(300, 400);  
    setLocation(450, 100);  
    setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);  
    myTextArea.setLineWrap(true);  
    button1.addActionListener(this);  
    button2.addActionListener(this);  
    textField1.addKeyListener(this);  
    setVisible(true);  
}
```

A yellow arrow points to the line `myTextArea.setLineWrap(true);` in the code.

Other Layouts to Explore

Oftentimes, the key to making GUIs look the way you want them to is to understand the "LayoutManager" for the Panels.

The default is the **GridLayout**:

Property	Value
field name	mainPanel
Custom Create	<input type="checkbox"/>
Layout Manager	GridLayoutManager (IntelliJ)
+ border	None
+ margins	[0, 0, 0, 0]



But there are more you can choose from!

<https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

Final product!

