Software Design and Architecture

Lab # 10

Abstract class, Interface, GUI using Swing

Instructor: Fariba Laiq

Email: fariba.laiq@nu.edu.pk

Course Code: CL1002

Semester Spring 2023

Department of Computer Science,

National University of Computer and Emerging Sciences FAST Peshawar Campus

Java Abstract Class

The abstract class in Java cannot be instantiated (we cannot create objects of abstract classes).

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

For example

```
// create an abstract class
abstract class Language {
  // fields and methods
}
...

// try to create an object Language
// throws an error
Language obj = new Language();
```

An abstract class can have both the regular methods and abstract methods. For example,

```
abstract class Language {

  // abstract method
  abstract void method1();

  // regular method
  void method2() {
    System.out.println("This is regular method");
  }
}
```

A method that doesn't have its body is known as an abstract method. We use the same abstract keyword to create abstract methods. For example,

```
abstract void display();
```

Here, display() is an abstract method. The body of display() is replaced by ;.

If a class contains an abstract method, then the class should be declared abstract. Otherwise, it will generate an error.

For example

```
// error
// class should be abstract
class Language {

  // abstract method
  abstract void method1();
}
```

Though abstract classes cannot be instantiated, we can create subclasses from it. We can then access members of the abstract class using the object of the subclass. For example,

If the abstract class includes any abstract method, then all the child classes inherited from the abstract superclass must provide the implementation of the abstract method, otherwise that class should also be declared as abstract.

Example Code:

Shape.java

```
public abstract class Shape {
    abstract void draw();
}
```

Rectangle.java

```
public class Rectangle extends Shape{
    @Override
    void draw() {
        System.out.println("Drawing a shape");
    }
}
```

Main.java

```
public class Main {
   public static void main(String[] args) {
     Rectangle r=new Rectangle();
     r.draw();
   }
}
```

Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**.

It cannot be instantiated just like the abstract class.

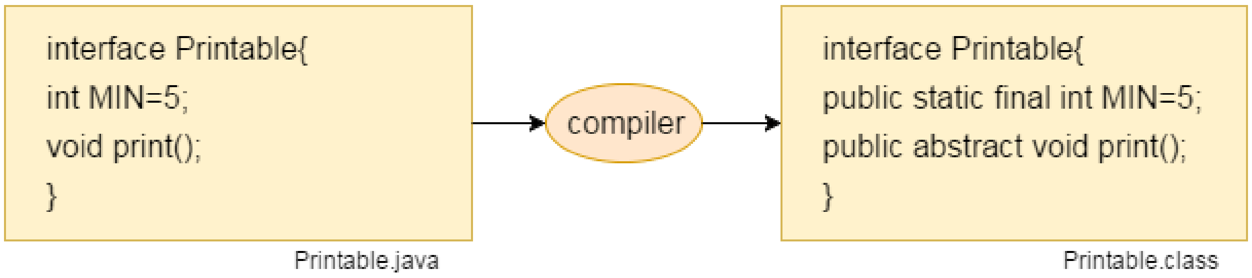
Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

Internal addition by the compiler

The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.

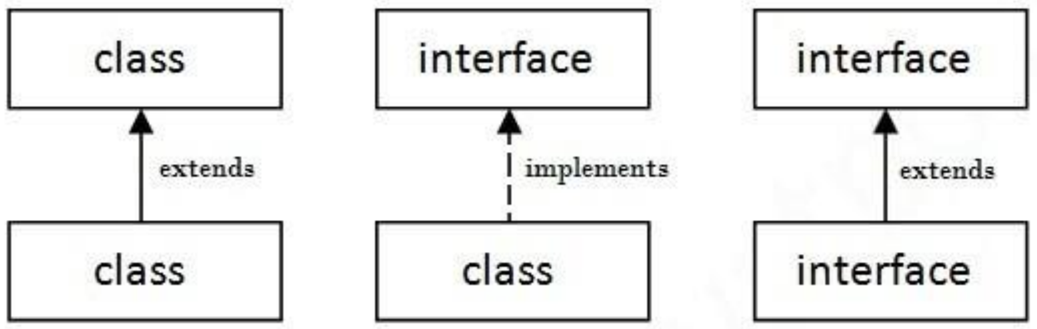


```
interface Printable{
int MIN=5;
void print();
}
```

```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

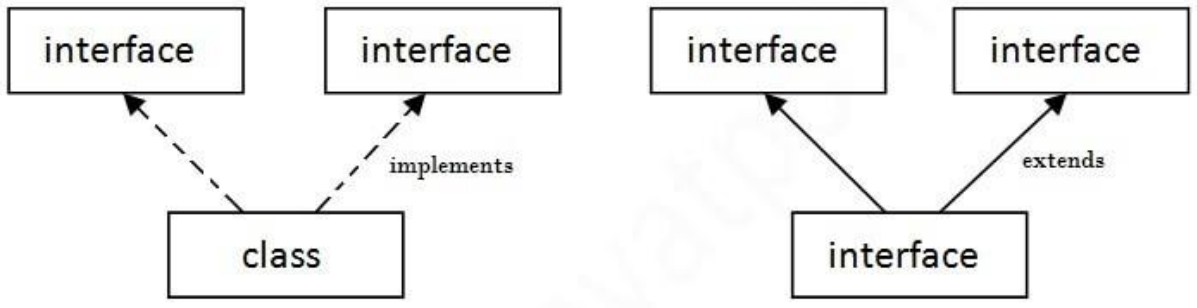
IS-A relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Multiple inheritance in Java by interface

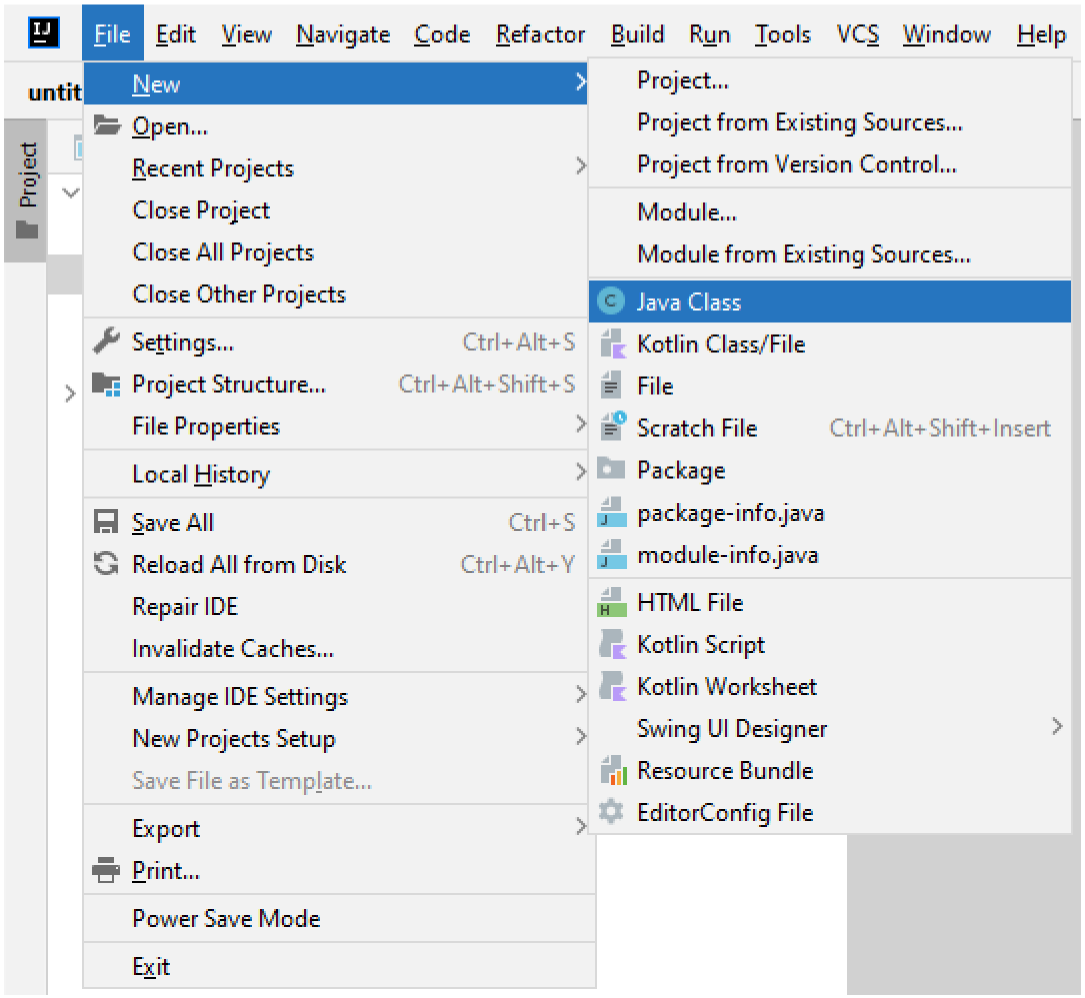
If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

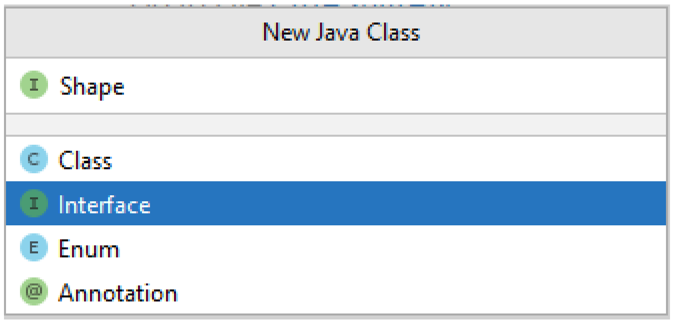


Multiple Inheritance in Java

As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

Create an Interface in Java





Shape.java (Interface)

```
public interface Shape {
    String name = "shape";
    void draw();
}
```

Rectangle.java (class)

```
public class Rectangle implements Shape{
    @Override
    public void draw() {
        // name="Rectangle" error because by default the variables of interface are public static
final
        System.out.println("Drawing rectangle");
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Rectangle r=new Rectangle();
        r.draw();
    }
}
```

Output:

```
Drawing rectangle
```

Multiple Inheritance using Interfaces

interface1.java

```
public interface interface1 {
   void foo1();
}
```

interface2.java

```
public interface interface2 {
    void foo2();
}
```

MyClass.java

```
public class MyClass implements interface1, interface2 {
    @Override
    public void foo1() {
        System.out.println("foo 1");
    }
    @Override
    public void foo2() {
        System.out.println("foo 2");
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
       MyClass c=new MyClass();
       c.foo1();
       c.foo2();
    }
}
```

Interface extending another interface implemented by a class

Interface1.java

```
public interface interface1 {
   void foo1();
}
```

interface2.java

```
public interface interface2 extends interface1{
    void foo2();
}
```

MyClass.java

```
public class MyClass implements interface2 {
    @Override
    public void foo1() {
        System.out.println("foo 1");

    }
    @Override
    public void foo2() {
        System.out.println("foo 2");
    }
}
```

Main.java

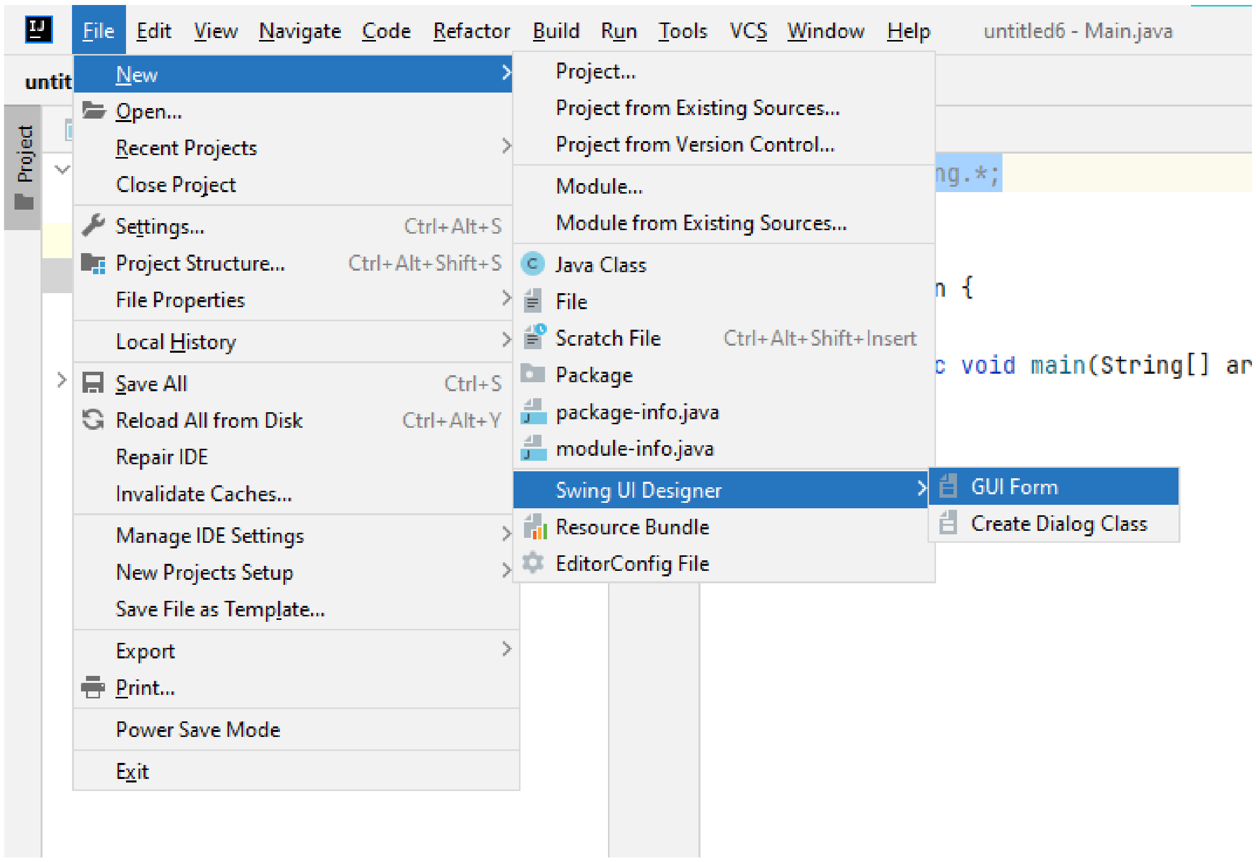
```
public class Main {
   public static void main(String[] args) {
      MyClass c=new MyClass();
      c.foo1();
      c.foo2();
   }
}
```

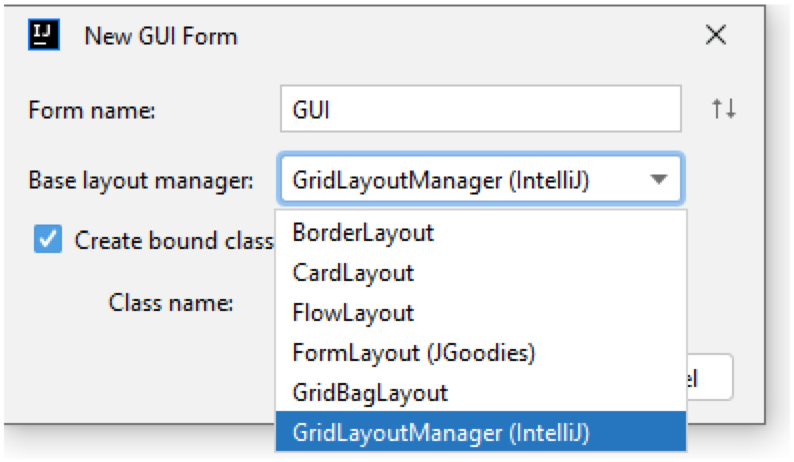
Swing in Java

Swing is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a package in Java that contains all the classes related to GUI. It is located in javax package. To use this package you will need to import it first using the following line.

```
import javax.swing.*;
```

Create a new GUI Form using Swing in Java



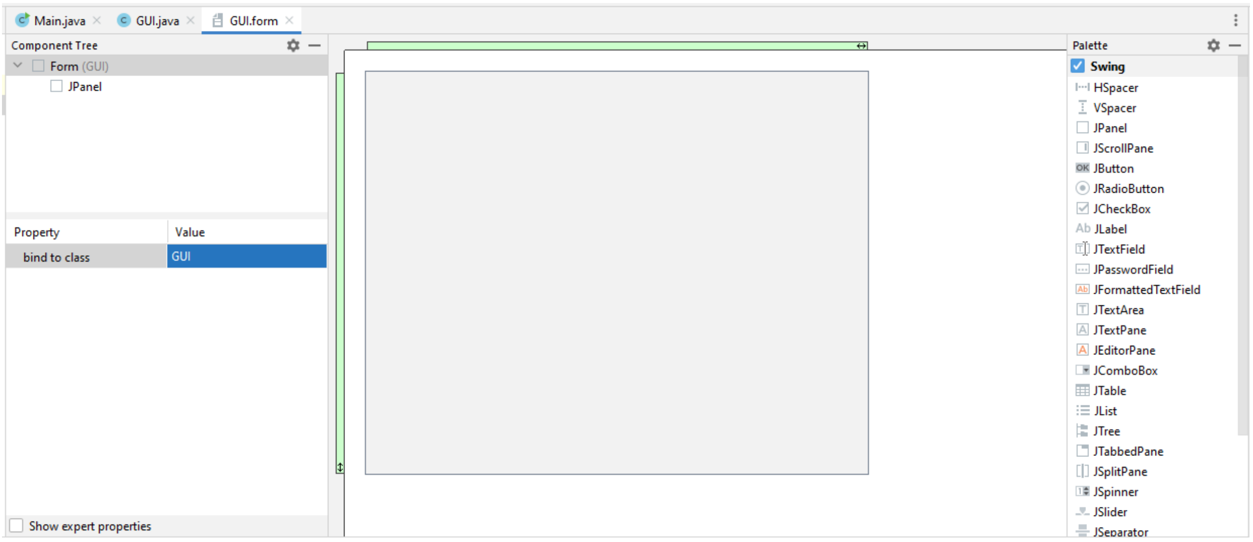


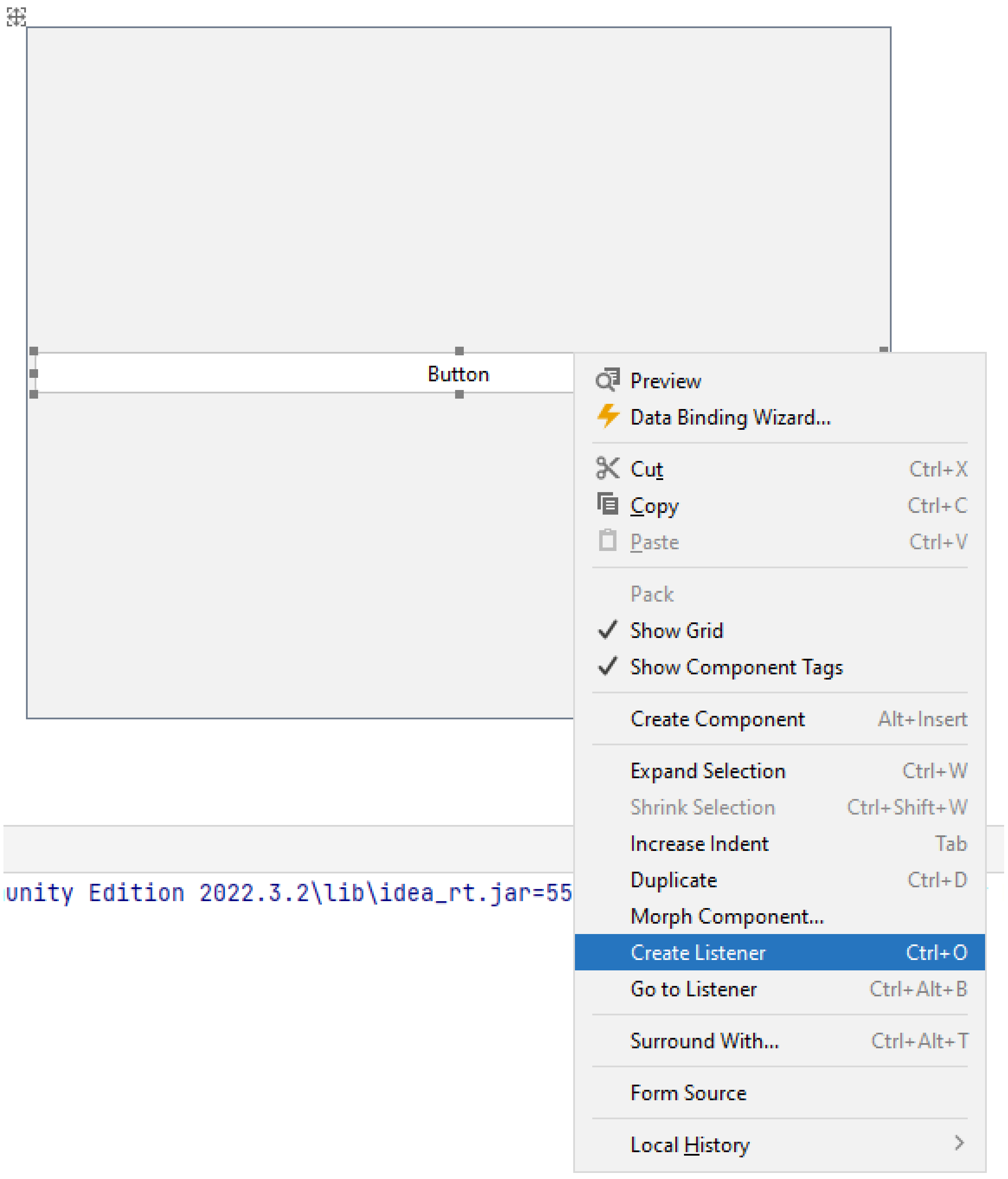
Form name: GUI

Base layout manager: GridLayoutManager (IntelliJ)

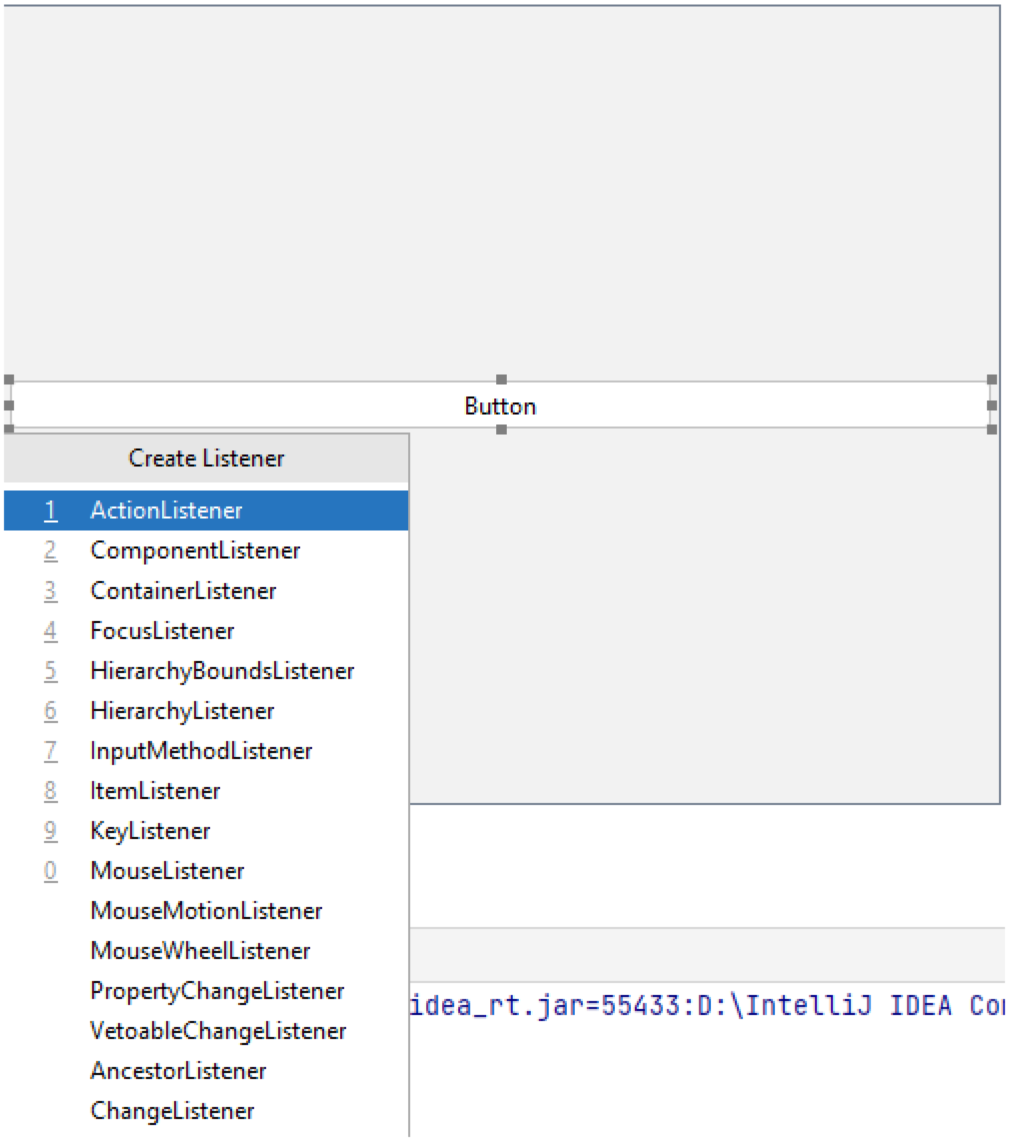
☑ Create bound class

Class name:

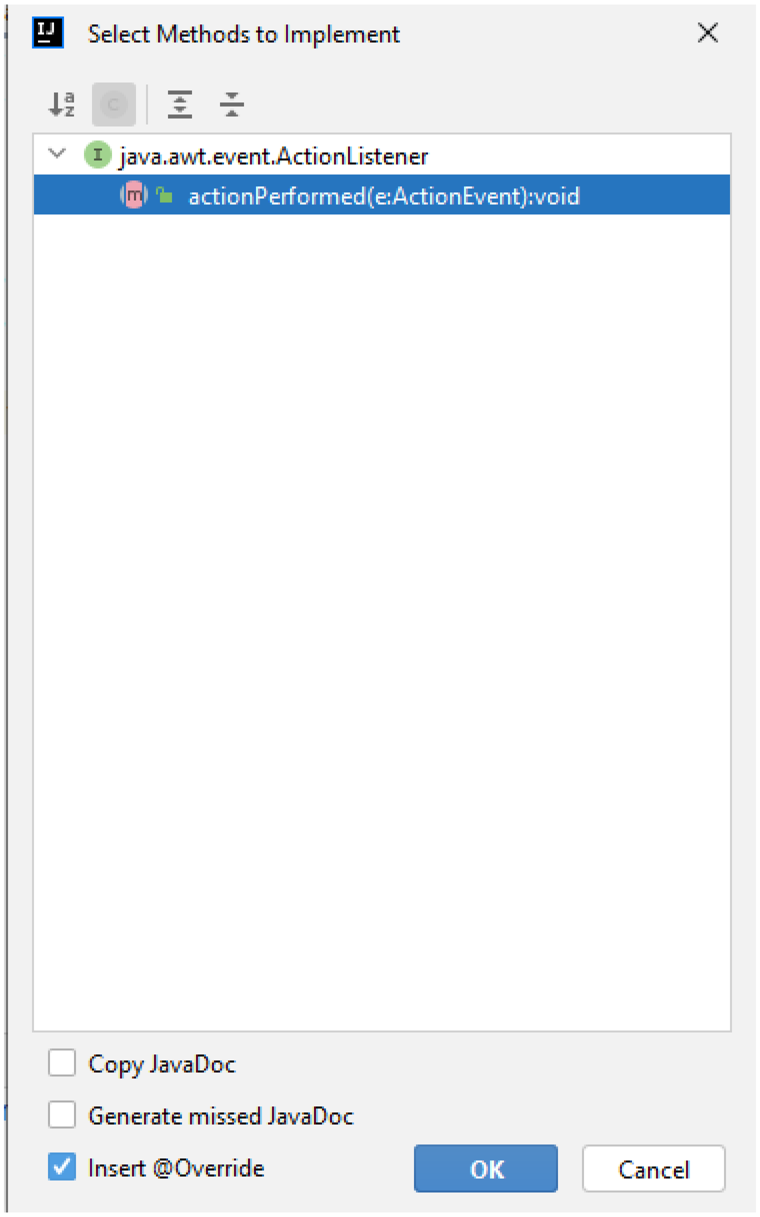

Main.java
GUI.java
GUI.form
Component Tree
Form (GUI)
JPanel
Property
Value
bind to class
GUI
Show expert properties
Palette
Swing
HSpacer
VSpacer
JPanel
JScrollPane
JButton
JRadioButton
JCheckBox
JLabel
JTextField
JPasswordField
JFormattedTextField
JTextArea
JTextPane
JEditorPane
JComboBox
JTable
JList
JTree
JTabbedPane
JSplitPane
JSpinner
JSlider
JSeparator



```
unity Edition 2022.3.2\lib\idea_rt.jar=55
```

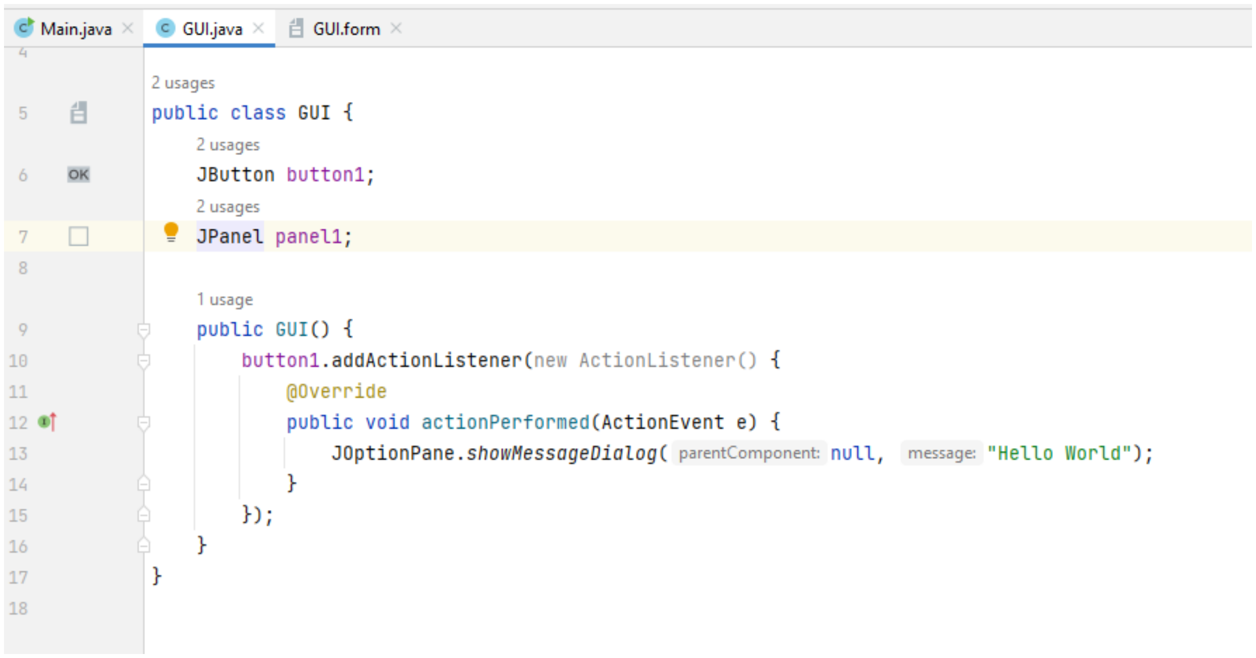


```
idea_rt.jar=55433:D:\IntelliJ IDEA Co
```

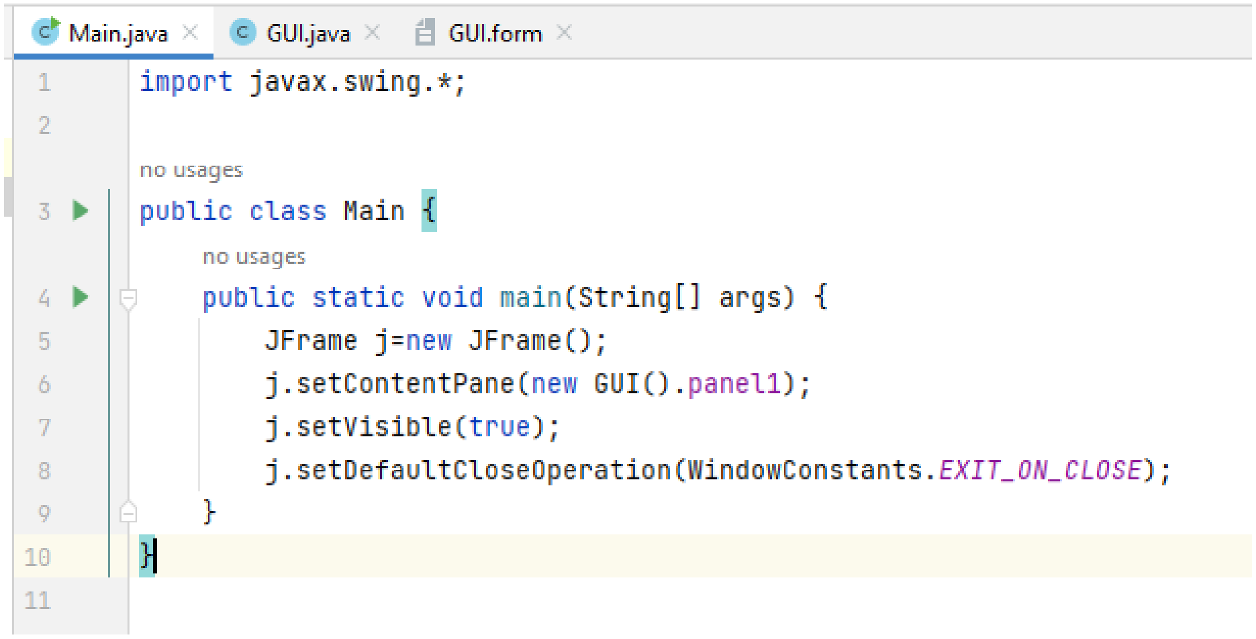

Select Methods to Implement

java.awt.event.ActionListener

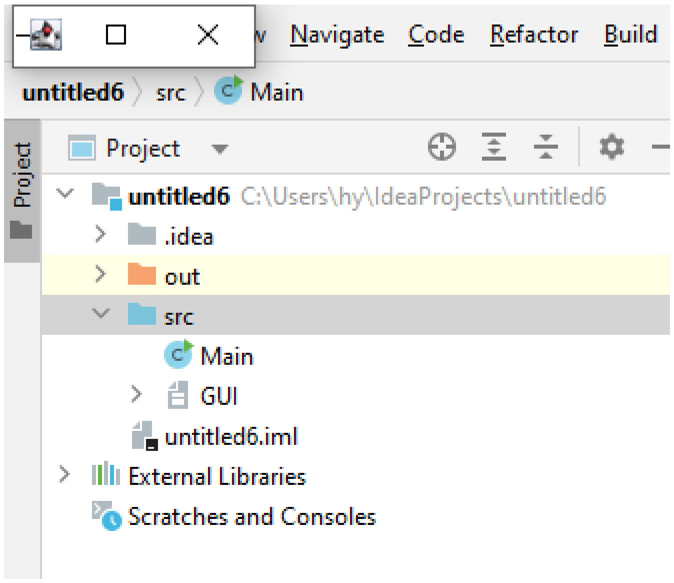
actionPerformed(e:ActionEvent):void

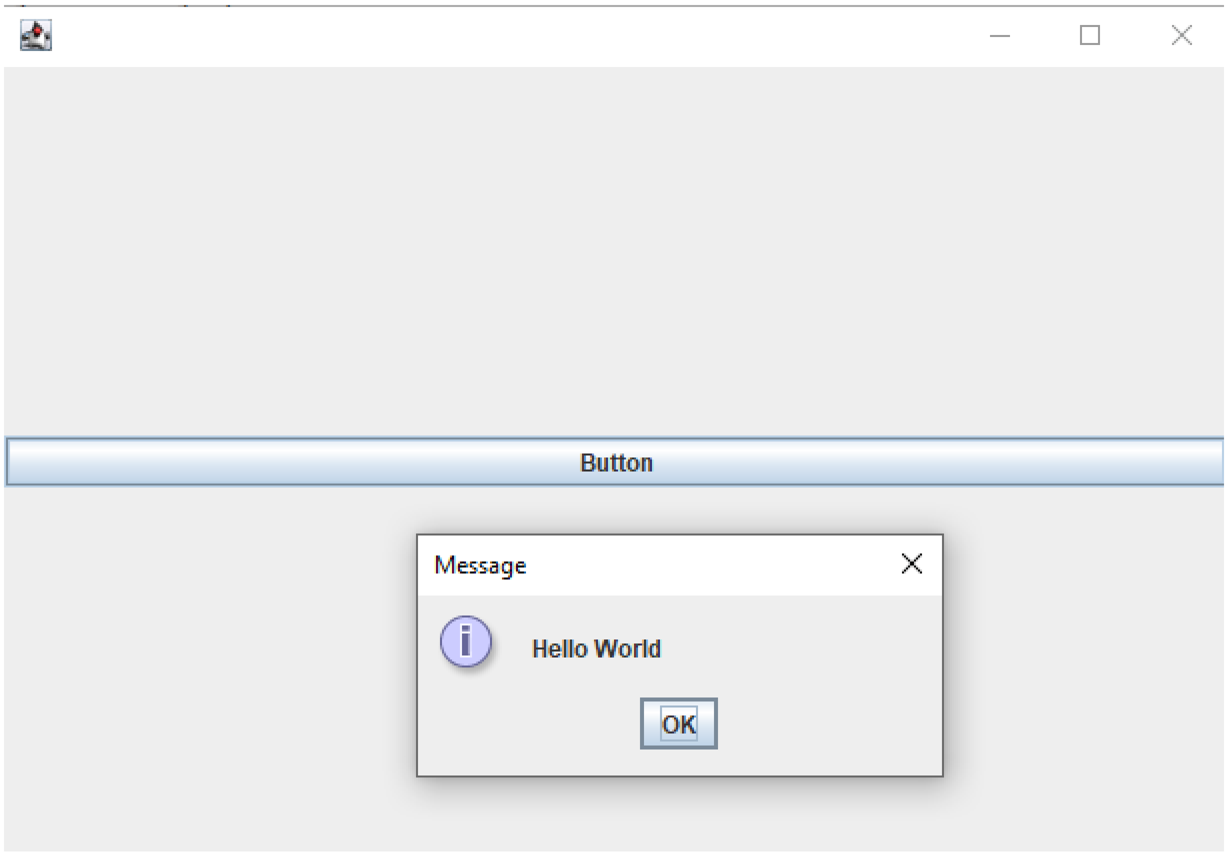

Main.java GUI.java GUI.form

```
2 usages
public class GUI {
    2 usages
    JButton button1;
    2 usages
    JPanel panel1;

    1 usage
    public GUI() {
        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Hello World");
            }
        });
    }
}
```


Main.java GUI.java GUI.form

```
import javax.swing.*;

no usages
public class Main {
    no usages
    public static void main(String[] args) {
        JFrame j=new JFrame();
        j.setContentPane(new GUI().panel1);
        j.setVisible(true);
        j.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}
```


v Navigate Code Refactor Build
untitled6 src Main
Project
Project
untitled6 C:\Users\hy\IdeaProjects\untitled6
.idea
out
src
Main
GUI
untitled6.iml
External Libraries
Scratches and Consoles


Button
Message
OK

Hello World

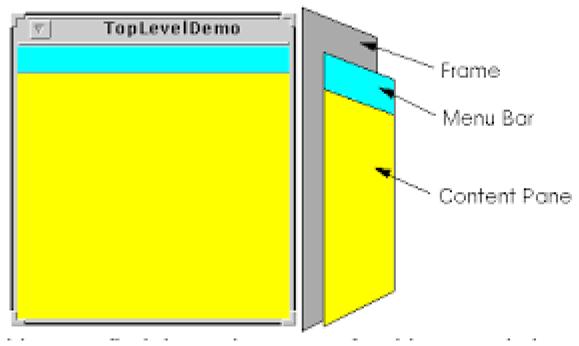
Create a GUI using coding from scratch in Java

JFrame

JFrame is a top-level container that provides a window on the screen. A frame is actually a base window on which other components rely, namely the menu bar, panels, labels, text fields, buttons, etc. Almost every other Swing application starts with the JFrame window.

Content Pane

JFrames have a content pane, which holds the components. These components are sized and positioned by the layout manager



Steps:

Import the required package:

Import javax.swing.*;

Setup the top level container:

JFrame jframe=new JFrame();

Get the component area of the top level container:

Container c=jframe.getContentPane();

c.setLayout(new FlowLayout);

Create and add components:

JButton b=new JButton(“Hello”);

c.add(b1);

Set size of the content pane and make it visible:

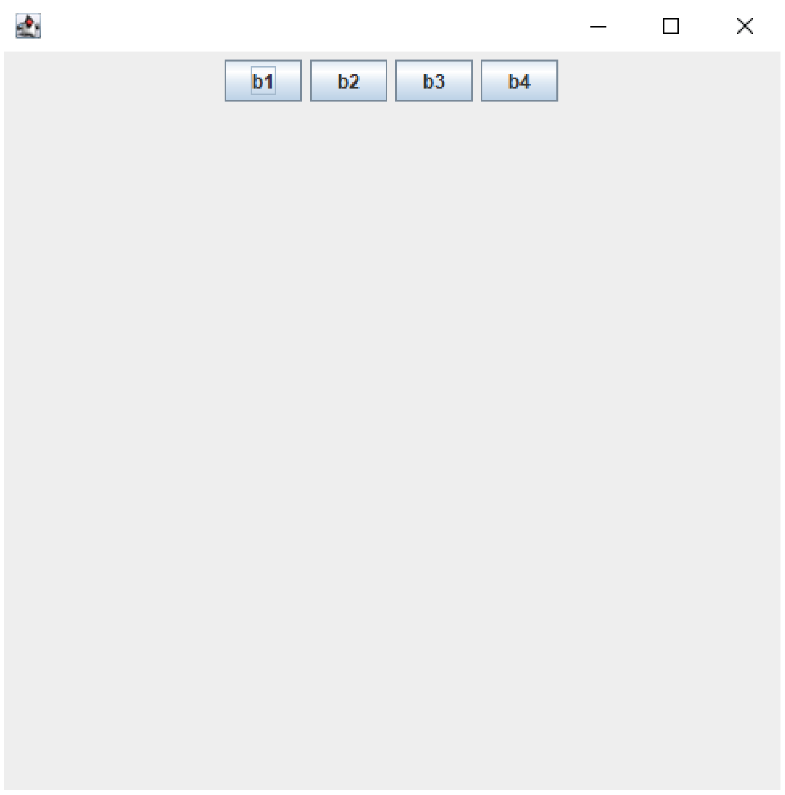
Jframe.setSize(500, 500);

Jframe.setVisibility(true);

Example Code:

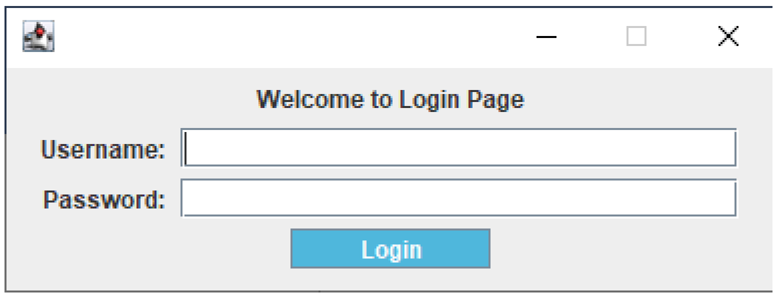
Main.java

```
import javax.swing.*;
import java.awt.*;

public class Main {
    public static void main(String[] args) {
        JFrame j=new JFrame();
        Container c=j.getContentPane();
        c.add(new JButton("b1"));
        c.add(new JButton("b2"));
        c.add(new JButton("b3"));
        c.add(new JButton("b4"));
        c.setLayout(new FlowLayout());
        j.setVisible(true);
        j.setSize(500,500);
        j.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}
```



Example Code 2:

```
import javax.swing.*;
import java.awt.*;

public class Main {
    public static void main(String[] args) {
        JFrame j = new JFrame();
        Container c = j.getContentPane();
        JLabel welcome = new JLabel("Welcome to Login Page");
        welcome.setPreferredSize(new Dimension(350,20));
        welcome.setHorizontalAlignment(SwingConstants.CENTER);
        c.add(welcome);
        JLabel username = new JLabel("Username: ");
        c.add(username);
        JTextField tf_username = new JTextField();
        tf_username.setColumns(25);
        c.add(tf_username);
        JLabel password = new JLabel("Password: ");
        c.add(password);
        JTextField tf_password = new JTextField();
        tf_password.setColumns(25);
        c.add(tf_password);
        JButton loginButton = new JButton("Login");
        loginButton.setBackground(new Color(79, 183, 220));
        loginButton.setPreferredSize(new Dimension(100,20));
        loginButton.setForeground(new Color(255, 255, 255));
        c.add(loginButton);
        c.setLayout(new FlowLayout());
        j.setVisible(true);
        j.setSize(400, 150);
        j.setResizable(false);
        j.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}
```



This code creates a simple login page using a JFrame with a FlowLayout layout manager. Here's a brief explanation of each part of the code:

```
JFrame j = new JFrame();

Container c = j.getContentPane();
```

This creates a new JFrame object and gets its content pane.

```
JLabel welcome = new JLabel("Welcome to Login Page");

welcome.setPreferredSize(new Dimension(350,20));
welcome.setHorizontalAlignment(SwingConstants.CENTER);

c.add(welcome);
```

This creates a JLabel with the text "Welcome to Login Page". The preferred size of the label is set to 350 pixels wide and 20 pixels tall. The setHorizontalAlignment() method is used to center the label within its available space. The label is added to the content pane of the frame.

```
JLabel username = new JLabel("Username: ");

c.add(username);

JTextField tf_username = new JTextField();

tf_username.setColumns(25);

c.add(tf_username);
```

This creates a JLabel with the text "Username:" and a JTextField for the user to enter their username. The text field is set to have 25 columns (i.e., 25 characters wide). Both the label and the text field are added to the content pane of the frame.

```
JLabel password = new JLabel("Password: ");

c.add(password);

JTextField tf_password = new JTextField();

 tf_password.setColumns(25);

c.add(tf_password);
```

This creates another JLabel with the text "Password:" and another JTextField for the user to enter their password. The text field is also set to have 25 columns. Both the label and the text field are added to the content pane of the frame.

```
JButton loginButton = new JButton("Login");

loginButton.setBackground(new Color(79, 183, 220));

loginButton.setPreferredSize(new Dimension(100,20));
```

loginButton.setForeground(new Color(255, 255, 255));

c.add(loginButton);

This creates a JButton with the text "Login". The button's background color is set to a light blue color using a Color object. The button's preferred size is set to be 100 pixels wide and 20 pixels tall. The foreground (text) color of the button is set to white. The button is added to the content pane of the frame.

c.setLayout(new FlowLayout());

j.setVisible(true);

j.setSize(400, 150);

j.setResizable(false);

j.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

This sets the layout manager of the content pane to a FlowLayout. The frame is made visible, with a size of 400 pixels wide and 150 pixels tall. The frame is set to not be resizable. Finally, the close operation for the frame is set to EXIT_ON_CLOSE, which means the application will exit when the user closes the window.