1. Create a program to accept 10 integers from the user, store them in an array, and calculate their sum and average using a for-each loop.

The program uses a for-each loop to iterate through the array to calculate the sum of the integers, which is then used to compute the average.

import java.util.Scanner;

public class SumAndAverage {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int[] numbers = new int[10];

int sum = 0;

// Accept 10 integers from the user

System.out.println("Enter 10 integers:");

for (int i = 0; i < 10; i++) {

numbers[i] = scanner.nextInt();

}

// Calculate the sum using a for-each loop

for (int num : numbers) {

sum += num;

}

// Calculate the average

double average = sum / 10.0;

// Display the sum and average

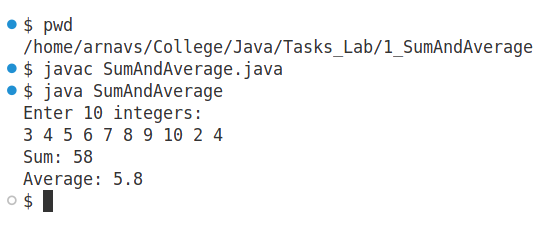
System.out.println("Sum: " + sum);

System.out.println("Average: " + average);

scanner.close();

}

}



1. Design a student class with attributes such as id, name, and marks. Write methods to calculate the grade based on marks and display student details.

The Student class contains attributes and methods to encapsulate student information. The calculateGrade method determines the grade based on the marks, and the displayDetails method prints the student's details.

public class Student {

// Attributes

private int id;

private String name;

private double marks;

// Constructor

public Student(int id, String name, double marks) {

this.id = id;

this.name = name;

this.marks = marks;

}

// Method to calculate grade based on marks

public String calculateGrade() {

if (marks >= 90) {

return "A";

} else if (marks >= 80) {

return "B";

} else if (marks >= 70) {

return "C";

} else if (marks >= 60) {

return "D";

} else {

return "F";

}

}

// Method to display student details

public void displayDetails() {

System.out.println("Student ID: " + id);

System.out.println("Name: " + name);

System.out.println("Marks: " + marks);

System.out.println("Grade: " + calculateGrade());

}

// Main method for testing

public static void main(String[] args) {

// Creating a Student object

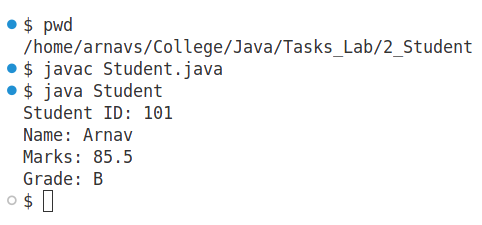
Student student = new Student(101, "Arnav", 85.5);

// Displaying student details

student.displayDetails();

}

}



1. Write a program to demonstrate method overloading by implementing a Calculator class with overloaded methods for add (supporting 2, 3, and 4 integers).

Method overloading allows multiple methods with the same name but different parameter lists. The Calculator class defines three add methods, each taking a different number of parameters.

public class Calculator {

// Method to add two integers

public int add(int a, int b) {

return a + b;

}

// Method to add three integers

public int add(int a, int b, int c) {

return a + b + c;

}

// Method to add four integers

public int add(int a, int b, int c, int d) {

return a + b + c + d;

}

public static void main(String[] args) {

Calculator calculator = new Calculator();

// Demonstrating method overloading

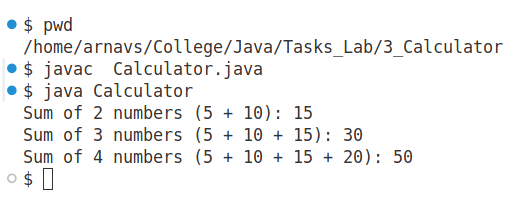
System.out.println("Sum of 2 numbers (5 + 10): " + calculator.add(5, 10));

System.out.println("Sum of 3 numbers (5 + 10 + 15): " + calculator.add(5, 10, 15));

System.out.println("Sum of 4 numbers (5 + 10 + 15 + 20): " + calculator.add(5, 10, 15, 20));

}

}



1. Demonstrate the use of public, private, protected, and package-private access modifiers in Java. Use different classes within the same and different packages.

Access modifiers in Java control the visibility of class members. Public members are accessible from any class, while private members are restricted to the same class. Protected members can be accessed within the same package and by subclasses. Package-private (default) members are accessible only within the same package.

For this, we have used two classes inside a package named Package4 as:

package Package4;

public class ClassA {

// Public attribute

public String publicData = "Public Data";

// Private attribute

private String privateData = "Private Data";

// Method to access private data

public void showPrivateData() {

System.out.println("Private Data: " + privateData);

}

}

package Package4;

public class ClassB extends ClassA {

// Protected attribute

protected String protectedData = "Protected Data";

// Method to access protected data

public void showProtectedData() {

System.out.println("Protected Data: " + protectedData);

}

}

package Package4;

public class ClassC {

public static void main(String[] args) {

// Create objects

ClassA classA = new ClassA();

ClassB classB = new ClassB();

// Accessing public field

System.out.println(classA.publicData);

// Accessing private field via method

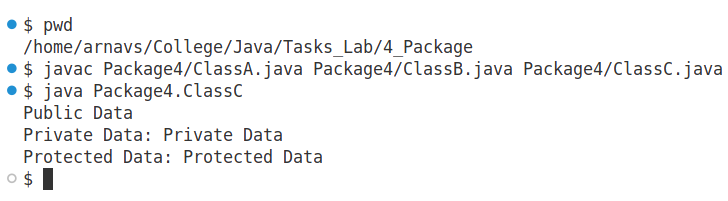
classA.showPrivateData();

// Accessing protected field via subclass

classB.showProtectedData();

}

}



1. Write a program with an interface Shape containing methods area() and perimeter(). Implement it in two classes, Circle and Rectangle. Include an inner class to calculate diagonal for the Rectangle

This program demonstrates the use of interfaces in Java. The Shape interface defines methods for calculating area and perimeter, which are implemented in the Circle and Rectangle classes. Additionally, an inner class is used within Rectangle to calculate the diagonal.

interface Shape {

double area();

double perimeter();

}

class Circle implements Shape {

private double radius;

// Constructor  
public Circle(double radius) {  
 this.radius = radius;  
}  
  
// Implementing area method  
public double area() {  
 return Math.PI \* radius \* radius;  
}  
  
// Implementing perimeter method  
public double perimeter() {  
 return 2 \* Math.PI \* radius;  
}

}

class Rectangle implements Shape {

private double length; private double width;  
public Rectangle(double length, double width) {  
 this.length = length;  
 this.width = width;  
}  
  
public double area() {  
 return length \* width;  
}  
  
public double perimeter() {  
 return 2 \* (length + width);  
}  
  
// Inner class to calculate the diagonal of the rectangle  
class Diagonal {  
 public double calculateDiagonal() {  
 return Math.sqrt(length \* length + width \* width);  
 }  
}

}

public class ShapeDemo {

public static void main(String[] args) {

Circle circle = new Circle(5.0);

Rectangle rectangle = new Rectangle(6.0, 8.0);

// Displaying Circle area and perimeter

System.out.println("Circle Area: " + circle.area());

System.out.println("Circle Perimeter: " + circle.perimeter());

// Displaying Rectangle area and perimeter

System.out.println("Rectangle Area: " + rectangle.area());

System.out.println("Rectangle Perimeter: " + rectangle.perimeter());

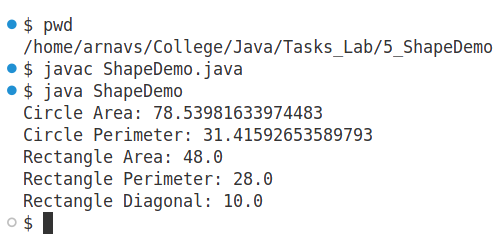
// Creating an inner class object to calculate the diagonal of the Rectangle

Rectangle.Diagonal diagonal = rectangle.new Diagonal();

System.out.println("Rectangle Diagonal: " + diagonal.calculateDiagonal());

}

}



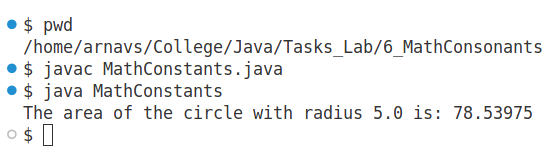
1. Create a class MathConstants with a final variable for PI and a static method to calculate the area of a circle.

This program demonstrates the use of final and static in Java. The MathConstants class defines a constant for PI and includes a static method to calculate the area of a circle using the formula Area = PI \* radius^2.

public class MathConstants {

final double PI = 3.14159;

// Static method to calculate the area of a circle  
public static double calculateArea(double radius) {  
 return PI \* radius \* radius;  
}  
  
public static void main(String[] args) {  
 // Test the static method by passing a radius  
 double radius = 5.0;  
 double area = MathConstants.calculateArea(radius);  
   
 // Display the result  
 System.out.println("The area of the circle with radius " + radius + " is: " + area);  
}  
}



1. Create two packages, com.math.geometry and com.math.algebra, and write classes in each to demonstrate their usage in a program.

This program demonstrates how to create and use Java packages. It defines two packages, com.math.geometry and com.math.algebra, and creates classes in each package to illustrate their usage in a program.

com.math.geometry contains a Rectangle class to compute a rectangle.

com.math.algebra contains an EquationSolver class to solve quadratic equations.

package com.math.geometry;

public class Rectangle {

private double length;

private double width;

// Constructor

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

// Method to calculate area

public double area() {

return length \* width;

}

// Method to calculate perimeter

public double perimeter() {

return 2 \* (length + width);

}

}

package com.math.algebra;

public class EquationSolver {

public static void solveQuadratic(double a, double b, double c)

{  
 double discriminant = b \* b - 4 \* a \* c;  
   
 if (discriminant > 0) {  
 double root1 = (-b + Math.sqrt(discriminant)) / (2 \* a);  
 double root2 = (-b - Math.sqrt(discriminant)) / (2 \* a);  
 System.out.println("Roots are real and different: " + root1 + " and " + root2);  
 } else if (discriminant == 0) {  
 double root = -b / (2 \* a);  
 System.out.println("Root is real and the same: " + root);  
 } else {  
 System.out.println("Roots are complex.");  
 }  
}

}

// Main.java

import com.math.geometry.Rectangle;

import com.math.algebra.EquationSolver;

public class Main {

public static void main(String[] args) {

// Create Rectangle object

Rectangle rectangle = new Rectangle(5.0, 3.0);

// Calculate area and perimeter

System.out.println("Rectangle Area: " + rectangle.area());

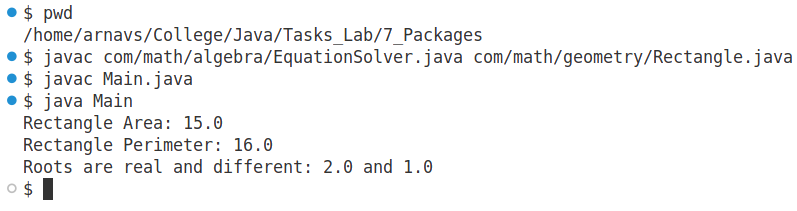
System.out.println("Rectangle Perimeter: " + rectangle.perimeter());

// Solve a quadratic equation

EquationSolver.solveQuadratic(1, -3, 2);

}

}



1. Design a base class Employee with attributes name and salary, and a derived class Manager with additional attributes department. Override a method displayDetails() in both classes.

This program demonstrates inheritance and method overriding in Java. The base class Employee contains common attributes like name and salary, while the derived class Manager adds an additional attribute department and overrides the displayDetails() method to provide specific details for managers.

public class Employee {

private String name;

private double salary;

public Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

// Method to display details (to be overridden)

public void displayDetails() {

System.out.println("Employee Name: " + name);

System.out.println("Employee Salary: " + salary);

}

}

public class Manager extends Employee {

private String department;

public Manager(String name, double salary, String department) {

super(name, salary); // Calling the base class constructor

this.department = department;

}

// Overriding displayDetails() method

@Override

public void displayDetails() {

// Calling the base class method to display common details

super.displayDetails();

System.out.println("Department: " + department);

}

}

public class Main {

public static void main(String[] args) {

Employee emp = new Employee("John Doe", 50000);

Manager mgr = new Manager("Alice Smith", 75000, "HR");

System.out.println("Employee Details:");

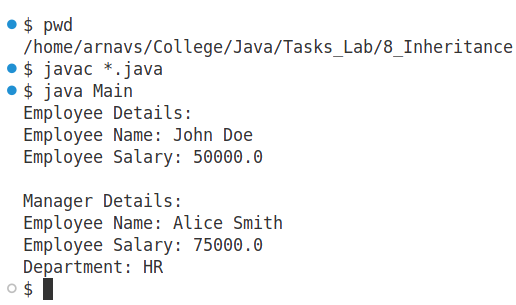
emp.displayDetails();

System.out.println("\nManager Details:");

mgr.displayDetails();

}

}



1. Write a program that prompts the user to enter two integers and performs division. Handle exceptions for invalid inputs (e.g., non-numeric input) and division by zero.

This program demonstrates exception handling in Java. It prompts the user to input two integers, performs division, and handles potential exceptions such as invalid (non-numeric) inputs and division by zero.

import java.util.Scanner;

public class DivisionCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int numerator = 0, denominator = 0;

// Prompt the user for input

try {

System.out.print("Enter the numerator: ");

numerator = Integer.parseInt(scanner.nextLine());

System.out.print("Enter the denominator: ");

denominator = Integer.parseInt(scanner.nextLine());

// Perform division and handle division by zero

if (denominator == 0) {

throw new ArithmeticException("Cannot divide by zero!");

}

int result = numerator / denominator;

System.out.println("Result: " + result);

} catch (NumberFormatException e) {

System.out.println("Error: Invalid input! Please enter valid integers.");

} catch (ArithmeticException e) {

System.out.println("Error: " + e.getMessage());

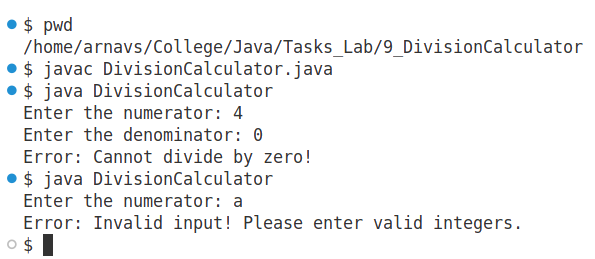
} finally {

scanner.close();

}

}

}



1. Write a multithreaded program to print numbers from 1 to 10 using two threads, where one thread prints odd numbers and the other prints even numbers.

This program demonstrates multithreading in Java. Two threads are used: one prints the odd numbers from 1 to 10, and the other prints the even numbers. The program synchronizes the threads to ensure that the numbers are printed in the correct order.

public class OddEvenThread {

private static final Object lock = new Object();

// Thread to print odd numbers

static class OddNumberThread extends Thread {

public void run() {

for (int i = 1; i <= 10; i += 2) {

synchronized (lock) {

System.out.println(i);

lock.notify(); // Notify the other thread

try {

if (i < 9) {

lock.wait(); // Wait for the other thread to print even numbers

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

}

}

// Thread to print even numbers

static class EvenNumberThread extends Thread {

public void run() {

for (int i = 2; i <= 10; i += 2) {

synchronized (lock) {

System.out.println(i);

lock.notify(); // Notify the other thread

try {

if (i < 10) {

lock.wait(); // Wait for the other thread to print odd numbers

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

}

}

public static void main(String[] args) {

// Create two threads: one for odd numbers and one for even numbers

OddNumberThread oddThread = new OddNumberThread();

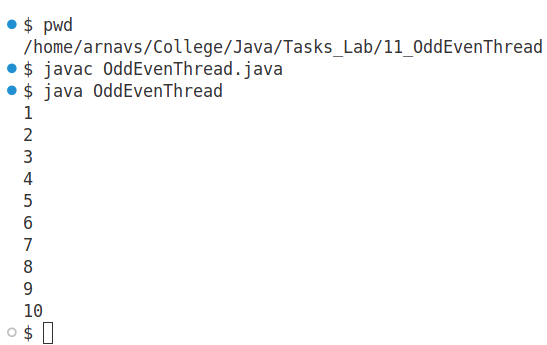
EvenNumberThread evenThread = new EvenNumberThread();

// Start both threads

oddThread.start(); evenThread.start();

}

}



1. Create a program where multiple threads update a shared counter. Use synchronization to ensure thread-safe operations.

This program demonstrates synchronization in multithreading. Multiple threads update a shared counter, and synchronization is used to ensure that each thread safely updates the counter without causing data inconsistency due to race conditions. The increment() method is synchronized to ensure that only one thread can update the counter at a time. This prevents race conditions. The program creates 10 threads, each incrementing the counter 1000 times.

public class SharedCounter {

private int counter = 0;

// Synchronized method to safely increment the counter

public synchronized void increment() {

counter++;

}

// Method to get the current value of the counter

public int getCounter() {

return counter;

}

public static void main(String[] args) throws InterruptedException {

SharedCounter sharedCounter = new SharedCounter();

// Create 10 threads that will update the counter

Thread[] threads = new Thread[10];

for (int i = 0; i < threads.length; i++) {

threads[i] = new Thread(new Runnable() {

@Override

public void run() {

// Increment the counter 1000 times

for (int j = 0; j < 1000; j++) {

sharedCounter.increment();

}

}

});

threads[i].start();

}

// Wait for all threads to finish

for (Thread t : threads) {

t.join();

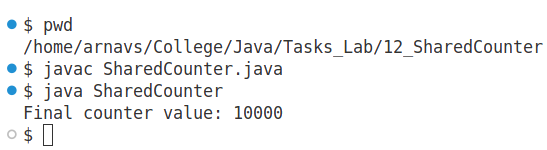
}

// Display the final counter value

System.out.println("Final counter value: " + sharedCounter.getCounter());

}

}



1. Write a program to create three threads with different priorities and observe their execution order.

This program demonstrates thread priorities in Java. Three threads are created with different priority levels (MIN\_PRIORITY, NORM\_PRIORITY, and MAX\_PRIORITY), and their execution order is observed. Thread priorities affect the order in which threads are scheduled by the JVM but don't guarantee a strict order of execution.

public class ThreadPriorityDemo {

static class MyThread extends Thread {

private String name;

public MyThread(String name) {

this.name = name;

}

@Override

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println(name + " - Iteration " + i + " (Priority: " + getPriority() + ")");

try {

Thread.sleep(100); // Simulate work by sleeping for a while

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public static void main(String[] args) throws InterruptedException {

// Create three threads with different priorities

MyThread thread1 = new MyThread("Thread 1");

MyThread thread2 = new MyThread("Thread 2");

MyThread thread3 = new MyThread("Thread 3");

// Set the thread priorities

thread1.setPriority(Thread.MIN\_PRIORITY); // 1

thread2.setPriority(Thread.NORM\_PRIORITY); // 5 (default)

thread3.setPriority(Thread.MAX\_PRIORITY); // 10

// Start the threads

thread1.start(); thread2.start(); thread3.start();

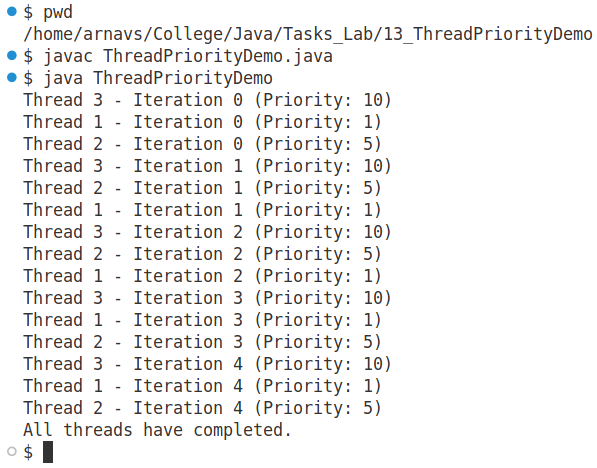
// Wait for all threads to finish

thread1.join(); thread2.join(); thread3.join();

System.out.println("All threads have completed.");

}

}



1. Write a program to copy the contents of one file to another using byte streams

This program demonstrates file copying using byte streams in Java. It reads the contents of one file using FileInputStream and writes it to another file using FileOutputStream. This method handles all types of file contents, including binary data.

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class FileCopy {

public static void main(String[] args) {

// Specify the source and destination file paths

String sourceFile = "source.txt";

String destinationFile = "destination.txt";

// Byte array for file copying

byte[] buffer = new byte[1024]; // Buffer to hold data temporarily

int bytesRead;

try (FileInputStream inputStream = new FileInputStream(sourceFile);

FileOutputStream outputStream = new FileOutputStream(destinationFile)) {

// Read and write file data in chunks

while ((bytesRead = inputStream.read(buffer)) != -1) {

outputStream.write(buffer, 0, bytesRead); // Write bytes to destination

}

System.out.println("File copied successfully!");

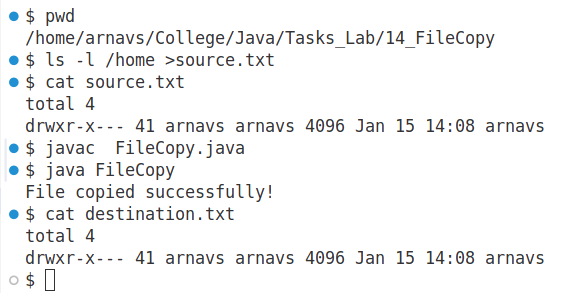
} catch (IOException e) {

System.out.println("Error occurred: " + e.getMessage());

}

}

}



1. Create a program to read from a text file and write its content to another file, line by line.

This program demonstrates reading content from a text file line by line and writing it to another file. It uses BufferedReader to read lines and BufferedWriter to write them. This approach is more efficient for text files compared to byte streams as it handles character encoding automatically.

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class FileCopyLineByLine {

public static void main(String[] args) {

// Specify the source and destination file paths

String sourceFile = "source.txt";

String destinationFile = "destination.txt";

// Try-with-resources to automatically close resources

try (BufferedReader reader = new BufferedReader(new FileReader(sourceFile));

BufferedWriter writer = new BufferedWriter(new FileWriter(destinationFile))) {

String line;

// Read and write each line

while ((line = reader.readLine()) != null) {

writer.write(line);

writer.newLine(); // Add a newline after each line

}

System.out.println("File content copied successfully!");

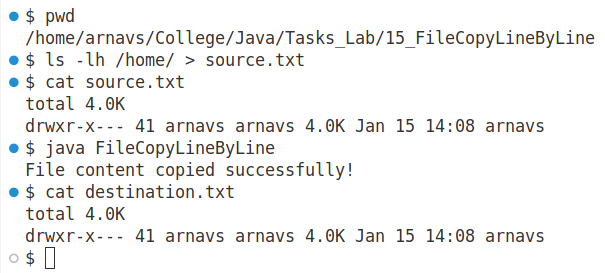
} catch (IOException e) {

System.out.println("Error occurred: " + e.getMessage());

}

}

}



1. Write a program to demonstrate the use of RandomAccessFile by writing data to a file at specific positions and reading it back.

This program demonstrates the use of RandomAccessFile in Java. The RandomAccessFile class allows you to read and write data to a file at specific positions, making it useful for tasks where you need to modify or read parts of a file without sequential access.

import java.io.IOException;

import java.io.RandomAccessFile;

public class RandomAccessFileDemo {

public static void main(String[] args) {

String fileName = "randomAccessFileDemo.txt";

try (RandomAccessFile raf = new RandomAccessFile(fileName, "rw")) {

// Write data to the file at specific positions

raf.writeUTF("Hello, ");

raf.seek(20); // Move the file pointer to position 20

raf.writeUTF("This is a test message!");

raf.seek(10); // Move the file pointer to position 10

raf.writeUTF("World! ");

// Now, read the data back from the file at specific positions

raf.seek(0); // Move the file pointer to the beginning

System.out.println("First read: " + raf.readUTF());

raf.seek(10); // Move the file pointer to position 10

System.out.println("Second read: " + raf.readUTF());

raf.seek(20); // Move the file pointer to position 20

System.out.println("Third read: " + raf.readUTF());

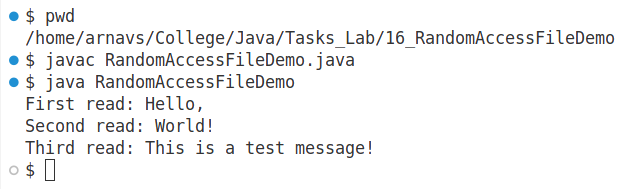
} catch (IOException e) {

System.out.println("Error: " + e.getMessage());

}

}

}



1. Create a program to serialize and deserialize an object of a Book class with attributes title, author, and price.

This program demonstrates the process of serializing and deserializing an object in Java. The Book class, which contains attributes like title, author, and price, is serialized to a file and then deserialized back into an object.

import java.io.Serializable;

public class Book implements Serializable {

private String title;

private String author;

private double price;

// Constructor

public Book(String title, String author, double price) {

this.title = title;

this.author = author;

this.price = price;

}

// Getters and setters

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

// Method to display book information

public void displayBookInfo() {

System.out.println("Title: " + title);

System.out.println("Author: " + author);

System.out.println("Price: " + price);

}

}

import java.io.\*;

public class SerializationDemo {

public static void main(String[] args) {

// Serialize the Book object

Book book1 = new Book("Java Programming", "John Doe", 39.99);

// Serialization process: Write the object to a file

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("book.ser"))) {

out.writeObject(book1);

System.out.println("Book object serialized successfully!");

} catch (IOException e) {

System.out.println("Error during serialization: " + e.getMessage());

}

// Deserialize the Book object

try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("book.ser"))) {

Book deserializedBook = (Book) in.readObject();

System.out.println("Book object deserialized successfully!");

deserializedBook.displayBookInfo(); // Display the details of the deserialized book

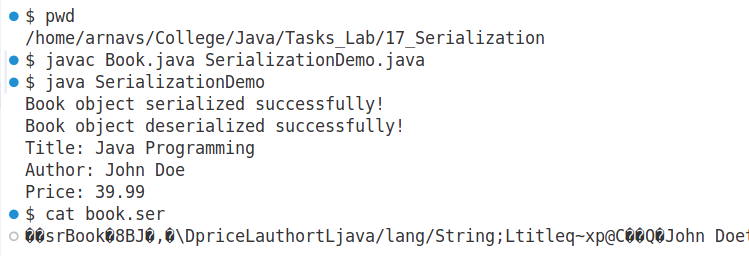
} catch (IOException | ClassNotFoundException e) {

System.out.println("Error during deserialization: " + e.getMessage());

}

}

}



1. Create a simple GUI using AWT and then implement the same using Swing.

AWT is the older GUI framework in Java, and it is platform-dependent. Below is an example that uses AWT to create the GUI.

import java.awt.\*;

import java.awt.event.\*;

public class AWTExample {

public static void main(String[] args) {

// Create a frame

Frame frame = new Frame("AWT Example");

// Create a label

Label label = new Label("Enter text: ");

label.setBounds(50, 50, 100, 30);

// Create a text field

TextField textField = new TextField();

textField.setBounds(150, 50, 200, 30);

// Create a button

Button button = new Button("Submit");

button.setBounds(150, 100, 100, 30);

// Create a result label

Label resultLabel = new Label();

resultLabel.setBounds(150, 150, 200, 30);

// Button click event

button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

resultLabel.setText("You entered: " + textField.getText());

}

});

// Add components to the frame

frame.add(label);

frame.add(textField);

frame.add(button);

frame.add(resultLabel);

// Set layout to null

frame.setLayout(null);

// Set the size and visibility of the frame

frame.setSize(400, 250);

frame.setVisible(true);

// Close the frame when the user closes it

frame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent we) {

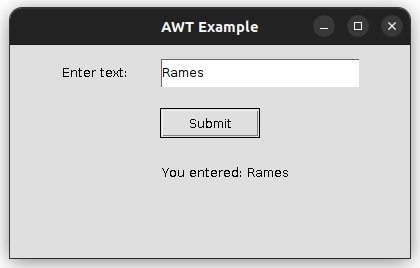
System.exit(0);

}

});

}

}



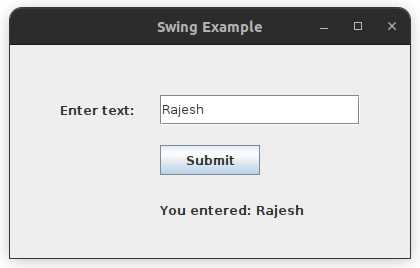
Swing is an improved version of AWT and provides more flexibility, is lightweight, and is platform-independent. Here's the same program implemented using Swing.

import javax.swing.;

import java.awt.event.;

public class SwingExample { public static void main(String[] args) { // Create a frame JFrame frame = new JFrame("Swing Example");

// Create a label  
 JLabel label = new JLabel("Enter text: ");  
 label.setBounds(50, 50, 100, 30);  
  
 // Create a text field  
 JTextField textField = new JTextField();  
 textField.setBounds(150, 50, 200, 30);  
  
 // Create a button  
 JButton button = new JButton("Submit");  
 button.setBounds(150, 100, 100, 30);  
  
 // Create a result label  
 JLabel resultLabel = new JLabel();  
 resultLabel.setBounds(150, 150, 200, 30);  
  
 // Button click event  
 button.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 resultLabel.setText("You entered: " + textField.getText());  
 }  
 });  
  
 // Add components to the frame  
 frame.add(label);  
 frame.add(textField);  
 frame.add(button);  
 frame.add(resultLabel);  
  
 // Set layout to null (absolute positioning)  
 frame.setLayout(null);  
  
 // Set the size and visibility of the frame  
 frame.setSize(400, 250);  
 frame.setVisible(true);  
  
 // Close the frame when the user closes it  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
}  
}



1. Create a Java Applet that displays a "Hello, World!" Message.

This program demonstrates creating a simple Java Applet that displays the message "Hello, World!" in a web browser or applet viewer. Java Applets are a type of Java program that can be embedded in a web page and run in a browser. However, it's important to note that applet support has been deprecated in most modern browsers, and applets are now considered obsolete.

First, we deine an applet class as HelloWorldApplet:

import java.applet.Applet;

import java.awt.Graphics;

public class HelloWorldApplet extends Applet {

// The paint method is used to display content on the applet window

@Override

public void paint(Graphics g) {

// Display "Hello, World!" message on the applet window

g.drawString("Hello, World!", 50, 50);

}

}

And then, we define a html file HelloWorldApplet.html as:

<html>

<body>

<applet code="HelloWorldApplet.class" width="300" height="100">

</applet>

</body>

</html>

1. Create a Swing application with components like JButton, JLabel, and JTextField added to a JPanel, which is then added to a JFrame.

This Swing application demonstrates how to create a GUI with JButton, JLabel, and JTextField components arranged on a JPanel, which is then added to a JFrame. Swing provides a flexible way to build GUI applications in Java. In this example, when the button is clicked, it will display the text entered in the text field onto the label.

import javax.swing.*;*

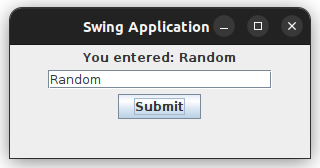
import java.awt.;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class SwingApp {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("Swing Application");  
  
 // Set the layout manager for the JFrame  
 frame.setLayout(new BorderLayout());  
  
 // Create a JPanel to hold components  
 JPanel panel = new JPanel();  
 panel.setLayout(new FlowLayout());  
  
 // Create a JLabel and JTextField for user input and display  
 JLabel label = new JLabel("Enter text:");  
 JTextField textField = new JTextField(20);  
   
 // Create a JButton  
 JButton button = new JButton("Submit");  
  
 // Add an ActionListener to the button to handle button clicks  
 button.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String text = textField.getText(); // Get text from the JTextField  
 label.setText("You entered: " + text); // Update the JLabel with entered text  
 }  
 });  
  
 // Add the components to the panel  
 panel.add(label);  
 panel.add(textField);  
 panel.add(button);  
  
 // Add the panel to the JFrame  
 frame.add(panel, BorderLayout.CENTER);  
  
 // Set the JFrame properties  
 frame.setSize(300, 150);  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 frame.setVisible(true);  
}  
}



1. Create a GUI application to demonstrate FlowLayout by adding buttons in a flow.

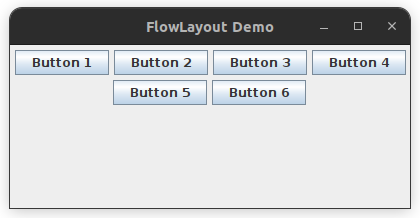
In this Swing application, we'll demonstrate the FlowLayout layout manager by adding multiple JButtons in a flow within a JPanel, which will be placed inside a JFrame. The FlowLayout arranges components sequentially from left to right and wraps them onto the next line if there is not enough space.

import javax.swing.*;*

import java.awt.;

public class FlowLayoutDemo {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("FlowLayout Demo");  
  
 // Set the layout manager for the JFrame  
 frame.setLayout(new FlowLayout());  
  
 // Create several JButtons  
 JButton button1 = new JButton("Button 1");  
 JButton button2 = new JButton("Button 2");  
 JButton button3 = new JButton("Button 3");  
 JButton button4 = new JButton("Button 4");  
 JButton button5 = new JButton("Button 5");  
 JButton button6 = new JButton("Button 6");  
  
 // Add the buttons to the frame  
 frame.add(button1);  
 frame.add(button2);  
 frame.add(button3);  
 frame.add(button4);  
 frame.add(button5);  
 frame.add(button6);  
  
 // Set JFrame properties  
 frame.setSize(400, 200); // Set the size of the window  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
}



1. Design a calculator-like GUI using BorderLayout with buttons at different positions (NORTH, SOUTH, EAST, WEST, and CENTER).

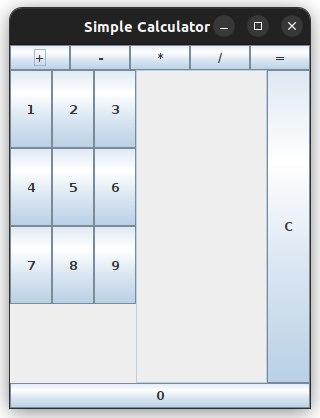
This Swing application demonstrates a simple calculator-like GUI using BorderLayout. The GUI contains buttons for numbers, arithmetic operations, and clear functionality, but without any functionality behind the buttons. The buttons are arranged using BorderLayout in the NORTH, SOUTH, EAST, WEST, and CENTER positions.

import javax.swing.*;*

import java.awt.;

public class SimpleCalculator {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("Simple Calculator");  
  
 // Set the layout manager for the JFrame  
 frame.setLayout(new BorderLayout());  
  
 // Create a JTextField to display the result (currently no functionality)  
 JTextField textField = new JTextField();  
 textField.setHorizontalAlignment(JTextField.RIGHT);  
 textField.setEditable(false);  
 frame.add(textField, BorderLayout.CENTER);  
  
 // Create the buttons for the calculator  
 JButton[] numberButtons = new JButton[10];  
 for (int i = 0; i < 10; i++) {  
 numberButtons[i] = new JButton(String.valueOf(i));  
 }  
   
 JButton addButton = new JButton("+");  
 JButton subButton = new JButton("-");  
 JButton mulButton = new JButton("\*");  
 JButton divButton = new JButton("/");  
 JButton equalsButton = new JButton("=");  
 JButton clearButton = new JButton("C");  
  
 // Create panels for each section of the calculator  
 JPanel westPanel = new JPanel();  
 westPanel.setLayout(new GridLayout(4, 3)); // 4x3 grid for the numbers 1-9  
 for (int i = 1; i <= 9; i++) {  
 westPanel.add(numberButtons[i]);  
 }  
 frame.add(westPanel, BorderLayout.WEST);  
   
 JPanel northPanel = new JPanel();  
 northPanel.setLayout(new GridLayout(1, 5)); // Row with operator buttons  
 northPanel.add(addButton);  
 northPanel.add(subButton);  
 northPanel.add(mulButton);  
 northPanel.add(divButton);  
 northPanel.add(equalsButton);  
  
 // Add the north panel to the frame  
 frame.add(northPanel, BorderLayout.NORTH);  
  
 // Add the '0' button to the south region  
 frame.add(numberButtons[0], BorderLayout.SOUTH);  
  
 // Add the clear button to the east region  
 frame.add(clearButton, BorderLayout.EAST);  
  
 // Set JFrame properties  
 frame.setSize(300, 400); // Set the size of the window  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
 }



1. Create a GUI with a GridLayout that displays a 3x3 grid of buttons labeled 1 to 9.

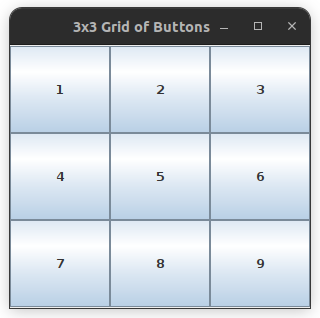
This Swing application creates a 3x3 grid of buttons, labeled from 1 to 9, using the GridLayout layout manager. The buttons are arranged in a 3x3 grid, which is commonly used for number-based layouts like a calculator keypad.

import javax.swing.;

import java.awt.;

public class GridLayoutDemo {

public static void main(String[] args) {  
 JFrame frame = new JFrame("3x3 Grid of Buttons");  
 frame.setLayout(new GridLayout(3, 3));  
 for (int i = 1; i <= 9; i++) {  
 JButton button = new JButton(String.valueOf(i));  
 frame.add(button); // Add button to the frame  
 }  
 frame.setSize(300, 300); // Set the size of the window  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
}



1. Implement a GridBagLayout to arrange components with varying sizes and positions.

This Swing application demonstrates how to use GridBagLayout to arrange components with varying sizes and positions. The GridBagLayout allows for flexible and complex arrangements of components, where each component can have different cell sizes and span across multiple cells.

import java.awt.Button;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import javax.swing.\*;

public class GridBagLayoutDemo extends JFrame{

public static void main(String[] args) {

GridBagLayoutDemo a = new GridBagLayoutDemo();

}

public GridBagLayoutDemo() {

GridBagLayout grid = new GridBagLayout();

GridBagConstraints gbc = new GridBagConstraints();

setLayout(grid);

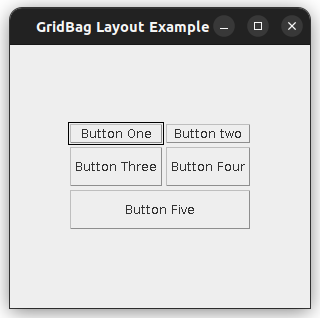
setTitle("GridBag Layout Example");

GridBagLayout layout = new GridBagLayout();

this.setLayout(layout);

gbc.fill = GridBagConstraints.HORIZONTAL;   
 gbc.gridx = 0;   
 gbc.gridy = 0;   
 this.add(new Button("Button One"), gbc);   
  
 gbc.gridx = 1;   
 gbc.gridy = 0;   
 this.add(new Button("Button two"), gbc);   
  
 gbc.fill = GridBagConstraints.HORIZONTAL;   
 gbc.ipady = 20;   
 gbc.gridx = 0;   
 gbc.gridy = 1;   
 this.add(new Button("Button Three"), gbc);   
  
 gbc.gridx = 1;   
 gbc.gridy = 1;   
 this.add(new Button("Button Four"), gbc);   
  
 gbc.gridx = 0;   
 gbc.gridy = 2;   
 gbc.fill = GridBagConstraints.HORIZONTAL;   
 gbc.gridwidth = 2;   
 this.add(new Button("Button Five"), gbc);   
  
 setSize(300, 300);   
 setPreferredSize(getSize());   
 setVisible(true);   
 setDefaultCloseOperation(EXIT\_ON\_CLOSE);   
  
}

}



1. Use GroupLayout to align a JLabel and JTextField in a form-like interface.

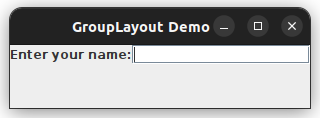
This Swing application uses GroupLayout to align a JLabel and JTextField in a form-like interface. GroupLayout is useful for creating forms and other layouts where components need to be aligned in a structured way, either horizontally or vertically.

import javax.swing.; import java.awt.;

public class GroupLayoutDemo {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("GroupLayout Demo");  
  
 // Set the layout manager for the JFrame to GroupLayout  
 GroupLayout layout = new GroupLayout(frame.getContentPane());  
 frame.setLayout(layout);  
  
 // Create JLabel and JTextField  
 JLabel label = new JLabel("Enter your name:");  
 JTextField textField = new JTextField(15); // 15 columns wide  
  
 // Define the horizontal group (horizontal alignment of components)  
 layout.setHorizontalGroup(  
 layout.createSequentialGroup()  
 .addComponent(label) // Add label  
 .addComponent(textField) // Add text field next to the label  
 );  
  
 // Define the vertical group (vertical alignment of components)  
 layout.setVerticalGroup(  
 layout.createParallelGroup(GroupLayout.Alignment.BASELINE)  
 .addComponent(label) // Align label to the left  
 .addComponent(textField) // Align text field next to the label  
 );  
  
 // Set JFrame properties  
 frame.setSize(300, 100); // Set the window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}

}



1. Create a form with JTextField, JPasswordField, and JTextArea. Add validation to ensure non-empty inputs.

This Swing application creates a simple form with JTextField, JPasswordField, and JTextArea components. It also includes validation to ensure that the inputs are not empty before submitting the form.

import javax.swing.*;*

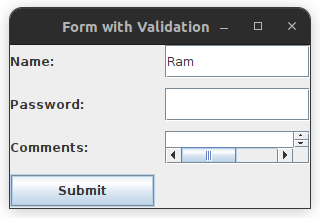
import java.awt.;

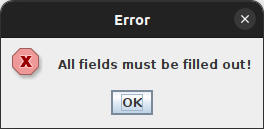
import java.awt.event.ActionEvent;

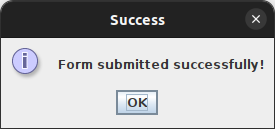
import java.awt.event.ActionListener;

public class FormValidationDemo {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("Form with Validation");  
  
 // Create JPanel to hold form components  
 JPanel panel = new JPanel();  
 panel.setLayout(new GridLayout(4, 2, 10, 10));  
  
 // Create components for the form  
 JLabel nameLabel = new JLabel("Name:");  
 JTextField nameField = new JTextField(20);  
 JLabel passwordLabel = new JLabel("Password:");  
 JPasswordField passwordField = new JPasswordField(20);  
 JLabel commentsLabel = new JLabel("Comments:");  
 JTextArea commentsArea = new JTextArea(3, 20);  
 JButton submitButton = new JButton("Submit");  
  
 // Add components to the panel  
 panel.add(nameLabel);  
 panel.add(nameField);  
 panel.add(passwordLabel);  
 panel.add(passwordField);  
 panel.add(commentsLabel);  
 panel.add(new JScrollPane(commentsArea)); // Scroll pane for JTextArea  
 panel.add(submitButton);  
  
 // Add panel to the frame  
 frame.add(panel);  
  
 // Add action listener for submit button  
 submitButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 // Validate the form fields  
 String name = nameField.getText().trim();  
 String password = new String(passwordField.getPassword()).trim();  
 String comments = commentsArea.getText().trim();  
  
 // Check if any field is empty  
 if (name.isEmpty() || password.isEmpty() || comments.isEmpty()) {  
 JOptionPane.showMessageDialog(frame, "All fields must be filled out!", "Error", JOptionPane.ERROR\_MESSAGE);  
 } else {  
 JOptionPane.showMessageDialog(frame, "Form submitted successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);  
 }  
 }  
 });  
 frame.setSize(300, 200); // Set the window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
}







1. Create a GUI with a JCheckBox and JRadioButton to select favorite programming languages. Show the selected options on a JLabel.

This Swing application demonstrates the use of a JCheckBox and JRadioButton to allow users to select their favorite programming languages. The selected options are then displayed on a JLabel.

import javax.swing.;

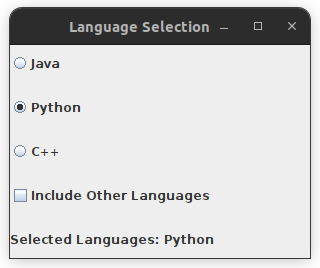
import java.awt.;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class LanguageSelectionDemo {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("Language Selection");  
  
 // Create JPanel to hold components  
 JPanel panel = new JPanel();  
 panel.setLayout(new GridLayout(5, 1, 10, 10));  
  
 // Create JCheckBox for additional languages  
 JCheckBox additionalLangCheckBox = new JCheckBox("Include Other Languages");  
  
 // Create JRadioButtons for programming languages  
 JRadioButton javaRadioButton = new JRadioButton("Java");  
 JRadioButton pythonRadioButton = new JRadioButton("Python");  
 JRadioButton cPlusPlusRadioButton = new JRadioButton("C++");  
  
 // Group the radio buttons so only one can be selected at a time  
 ButtonGroup group = new ButtonGroup();  
 group.add(javaRadioButton);  
 group.add(pythonRadioButton);  
 group.add(cPlusPlusRadioButton);  
  
 // Create JLabel to show the selected options  
 JLabel resultLabel = new JLabel("Selected Languages: None");  
  
 // Add components to the panel  
 panel.add(javaRadioButton);  
 panel.add(pythonRadioButton);  
 panel.add(cPlusPlusRadioButton);  
 panel.add(additionalLangCheckBox);  
 panel.add(resultLabel);  
  
 // Add panel to the frame  
 frame.add(panel);  
  
 // Add action listener for when a selection is made  
 ActionListener languageSelectionListener = new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String selectedLanguages = "Selected Languages: ";  
  
 // Check which radio button is selected  
 if (javaRadioButton.isSelected()) {  
 selectedLanguages += "Java ";  
 }   
 if (pythonRadioButton.isSelected()) {  
 selectedLanguages += "Python ";  
 }   
 if (cPlusPlusRadioButton.isSelected()) {  
 selectedLanguages += "C++ ";  
 }  
  
 if (additionalLangCheckBox.isSelected()) {  
 selectedLanguages += "(Other Languages)";  
 }  
  
 resultLabel.setText(selectedLanguages);  
 }  
 };  
  
 // Add the action listener to each radio button and checkbox  
 javaRadioButton.addActionListener(languageSelectionListener);  
 pythonRadioButton.addActionListener(languageSelectionListener);  
 cPlusPlusRadioButton.addActionListener(languageSelectionListener);  
 additionalLangCheckBox.addActionListener(languageSelectionListener);  
  
 // Set JFrame properties  
 frame.setSize(300, 250); // Set the window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
}



1. Create a GUI with a JComboBox to select a country and a JSlider to set age, and display the selections on a JLabel.

This Swing application demonstrates the use of a JComboBox to select a country and a JSlider to set an age. The selected country and age are then displayed on a JLabel.

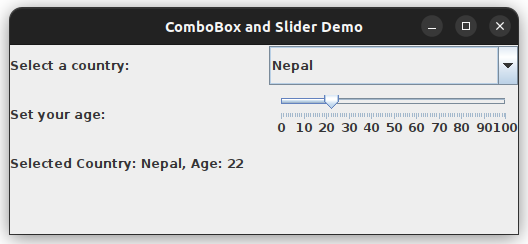
import javax.swing.;

import java.awt.;

import java.awt.event.\*;

public class ComboBoxSliderDemo {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("ComboBox and Slider Demo");  
  
 // Create a JPanel to hold components  
 JPanel panel = new JPanel();  
 panel.setLayout(new GridLayout(4, 1, 10, 10));  
  
 // Create a JComboBox for selecting a country  
 String[] countries = {"USA", "Canada", "Nepal", "UK", "Australia"};  
 JComboBox<String> countryComboBox = new JComboBox<>(countries);  
  
 // Create a JSlider for selecting an age (from 0 to 100)  
 JSlider ageSlider = new JSlider(0, 100, 25);  
 ageSlider.setMajorTickSpacing(10);  
 ageSlider.setMinorTickSpacing(1);  
 ageSlider.setPaintTicks(true);  
 ageSlider.setPaintLabels(true);  
  
 // Create a JLabel to display the selected country and age  
 JLabel resultLabel = new JLabel("Selected Country: None, Age: 25");  
  
 // Add components to the panel  
 panel.add(new JLabel("Select a country:"));  
 panel.add(countryComboBox);  
 panel.add(new JLabel("Set your age:"));  
 panel.add(ageSlider);  
 panel.add(resultLabel);  
  
 // Add panel to the frame  
 frame.add(panel);  
  
 // Add action listeners for the JComboBox and JSlider  
 countryComboBox.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 updateResultLabel(countryComboBox, ageSlider, resultLabel);  
 }  
 });  
  
 ageSlider.addChangeListener(e -> updateResultLabel(countryComboBox, ageSlider, resultLabel));  
  
 // Set JFrame properties  
 frame.setSize(300, 200); // Set the window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
  
// Method to update the result label based on the selected country and age  
private static void updateResultLabel(JComboBox<String> countryComboBox, JSlider ageSlider, JLabel resultLabel) {  
 String selectedCountry = (String) countryComboBox.getSelectedItem();  
 int selectedAge = ageSlider.getValue();  
 resultLabel.setText("Selected Country: " + selectedCountry + ", Age: " + selectedAge);  
}  
}



1. Create a menu bar with menus for "File" and "Edit." Add menu items such as "Open," "Save," and "Exit." Enable and disable them programmatically.

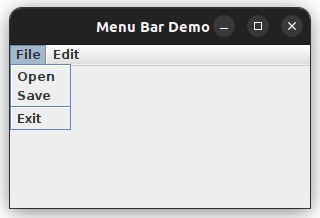
This Swing application creates a menu bar with two menus: File and Edit. It includes menu items such as Open, Save, and Exit. The menu items are enabled and disabled programmatically based on certain conditions.

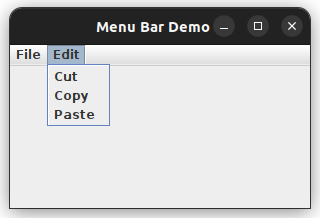
import javax.swing.;

import java.awt.event.;

public class MenuBarDemo {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("Menu Bar Demo");  
  
 // Create a menu bar  
 JMenuBar menuBar = new JMenuBar();  
  
 // Create File and Edit menus  
 JMenu fileMenu = new JMenu("File");  
 JMenu editMenu = new JMenu("Edit");  
  
 // Create menu items for the File menu  
 JMenuItem openMenuItem = new JMenuItem("Open");  
 JMenuItem saveMenuItem = new JMenuItem("Save");  
 JMenuItem exitMenuItem = new JMenuItem("Exit");  
  
 // Create menu items for the Edit menu  
 JMenuItem cutMenuItem = new JMenuItem("Cut");  
 JMenuItem copyMenuItem = new JMenuItem("Copy");  
 JMenuItem pasteMenuItem = new JMenuItem("Paste");  
  
 // Add menu items to the File menu  
 fileMenu.add(openMenuItem);  
 fileMenu.add(saveMenuItem);  
 fileMenu.addSeparator();  
 fileMenu.add(exitMenuItem);  
  
 // Add menu items to the Edit menu  
 editMenu.add(cutMenuItem);  
 editMenu.add(copyMenuItem);  
 editMenu.add(pasteMenuItem);  
  
 // Add the menus to the menu bar  
 menuBar.add(fileMenu);  
 menuBar.add(editMenu);  
  
 // Set the menu bar for the frame  
 frame.setJMenuBar(menuBar);  
  
 // Add action listeners for menu items  
 openMenuItem.addActionListener(e -> System.out.println("Open selected"));  
 saveMenuItem.addActionListener(e -> System.out.println("Save selected"));  
 exitMenuItem.addActionListener(e -> System.exit(0));  
   
 cutMenuItem.addActionListener(e -> System.out.println("Cut selected"));  
 copyMenuItem.addActionListener(e -> System.out.println("Copy selected"));  
 pasteMenuItem.addActionListener(e -> System.out.println("Paste selected"));  
  
 // Programmatically enable and disable menu items  
 saveMenuItem.setEnabled(false); // Disable the "Save" menu item initially  
  
 // Simulate enabling the "Save" menu item after some action (e.g., file is modified)  
 Timer timer = new Timer(5000, e -> {  
 saveMenuItem.setEnabled(true); // Enable the "Save" menu item after 5 seconds  
 });  
 timer.setRepeats(false);  
 timer.start();  
  
 // Set JFrame properties  
 frame.setSize(300, 200); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
}





1. Add icons to menu items and implement keyboard mnemonics and accelerators for quick access

This updated Swing application adds icons to the menu items and implements keyboard mnemonics and accelerators for quick access to the File and Edit menu items. Mnemonics allow the user to access menu items by pressing a single key, and accelerators allow users to use key combinations to trigger actions.

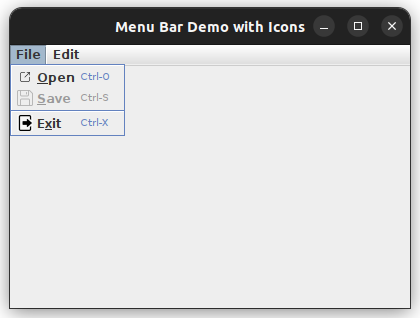
import javax.swing.;

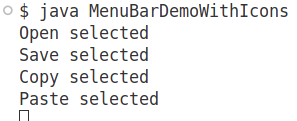
import java.awt.;

import java.awt.event.\*;

public class MenuBarDemoWithIcons {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("Menu Bar Demo with Icons");  
  
 // Create a menu bar  
 JMenuBar menuBar = new JMenuBar();  
  
 // Create File and Edit menus  
 JMenu fileMenu = new JMenu("File");  
 JMenu editMenu = new JMenu("Edit");  
  
 // Create menu items for the File menu  
 JMenuItem openMenuItem = new JMenuItem("Open", createIcon("open\_icon.png"));  
 JMenuItem saveMenuItem = new JMenuItem("Save", createIcon("save\_icon.png"));  
 JMenuItem exitMenuItem = new JMenuItem("Exit", createIcon("exit\_icon.png"));  
  
 // Create menu items for the Edit menu  
 JMenuItem cutMenuItem = new JMenuItem("Cut");  
 JMenuItem copyMenuItem = new JMenuItem("Copy");  
 JMenuItem pasteMenuItem = new JMenuItem("Paste");  
  
 // Set mnemonics for keyboard shortcuts  
 openMenuItem.setMnemonic(KeyEvent.VK\_O); // Alt + O  
 saveMenuItem.setMnemonic(KeyEvent.VK\_S); // Alt + S  
 exitMenuItem.setMnemonic(KeyEvent.VK\_X); // Alt + X  
 cutMenuItem.setMnemonic(KeyEvent.VK\_T); // Alt + T  
 copyMenuItem.setMnemonic(KeyEvent.VK\_C); // Alt + C  
 pasteMenuItem.setMnemonic(KeyEvent.VK\_P); // Alt + P  
  
 // Set accelerators for keyboard shortcuts  
 openMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_O, InputEvent.CTRL\_DOWN\_MASK)); // Ctrl + O  
 saveMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_S, InputEvent.CTRL\_DOWN\_MASK)); // Ctrl + S  
 exitMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_X, InputEvent.CTRL\_DOWN\_MASK)); // Ctrl + X  
 cutMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_T, InputEvent.CTRL\_DOWN\_MASK)); // Ctrl + T  
 copyMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_C, InputEvent.CTRL\_DOWN\_MASK)); // Ctrl + C  
 pasteMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_P, InputEvent.CTRL\_DOWN\_MASK)); // Ctrl + P  
  
 // Add menu items to the File menu  
 fileMenu.add(openMenuItem);  
 fileMenu.add(saveMenuItem);  
 fileMenu.addSeparator();  
 fileMenu.add(exitMenuItem);  
  
 // Add menu items to the Edit menu  
 editMenu.add(cutMenuItem);  
 editMenu.add(copyMenuItem);  
 editMenu.add(pasteMenuItem);  
  
 // Add the menus to the menu bar  
 menuBar.add(fileMenu);  
 menuBar.add(editMenu);  
  
 // Set the menu bar for the frame  
 frame.setJMenuBar(menuBar);  
  
 // Add action listeners for menu items  
 openMenuItem.addActionListener(e -> System.out.println("Open selected"));  
 saveMenuItem.addActionListener(e -> System.out.println("Save selected"));  
 exitMenuItem.addActionListener(e -> System.exit(0));  
   
 cutMenuItem.addActionListener(e -> System.out.println("Cut selected"));  
 copyMenuItem.addActionListener(e -> System.out.println("Copy selected"));  
 pasteMenuItem.addActionListener(e -> System.out.println("Paste selected"));  
  
 // Programmatically enable and disable menu items  
 saveMenuItem.setEnabled(false); // Disable the "Save" menu item initially  
  
 // Simulate enabling the "Save" menu item after some action (e.g., file is modified)  
 Timer timer = new Timer(5000, e -> {  
 saveMenuItem.setEnabled(true); // Enable the "Save" menu item after 5 seconds  
 });  
 timer.setRepeats(false);  
 timer.start();  
  
 // Set JFrame properties  
 frame.setSize(400, 300); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
// Helper method to create an ImageIcon with resized images  
private static ImageIcon createIcon(String path) {  
 ImageIcon icon = new ImageIcon(path);  
 Image img = icon.getImage(); // get the Image from ImageIcon  
 Image scaledImg = img.getScaledInstance(16, 16, Image.SCALE\_SMOOTH); // resize image to fit in menu  
 return new ImageIcon(scaledImg); // return a new ImageIcon with the resized image  
}  
}





1. Create a toolbar with buttons for common actions like "New," "Open," and "Save." Add tooltips for each button.

This simple Swing application demonstrates how to create a toolbar with buttons for common actions like New, Open, and Save. Each button is assigned a tooltip for better usability.

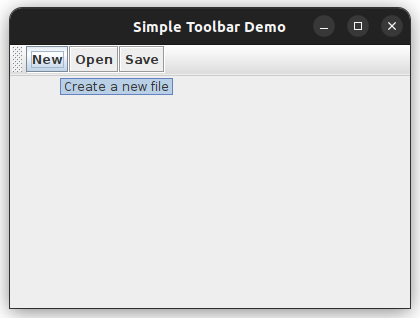
import javax.swing.;

import java.awt.;

import java.awt.event.\*;

public class SimpleToolbarDemo {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("Simple Toolbar Demo");  
  
 // Create a toolbar (JToolBar)  
 JToolBar toolbar = new JToolBar();  
  
 // Create buttons for common actions  
 JButton newButton = new JButton("New");  
 JButton openButton = new JButton("Open");  
 JButton saveButton = new JButton("Save");  
  
 // Add tooltips to the buttons  
 newButton.setToolTipText("Create a new file");  
 openButton.setToolTipText("Open an existing file");  
 saveButton.setToolTipText("Save the current file");  
  
 // Add buttons to the toolbar  
 toolbar.add(newButton);  
 toolbar.add(openButton);  
 toolbar.add(saveButton);  
  
 // Add action listeners for buttons  
 newButton.addActionListener(e -> System.out.println("New button clicked"));  
 openButton.addActionListener(e -> System.out.println("Open button clicked"));  
 saveButton.addActionListener(e -> System.out.println("Save button clicked"));  
  
 // Add the toolbar to the top of the frame  
 frame.add(toolbar, BorderLayout.NORTH);  
  
 // Set JFrame properties  
 frame.setSize(400, 300); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
}



1. Create an application that opens a file dialog to select a file and displays its contents in a JTextArea.

This Swing application demonstrates how to open a file dialog to allow users to select a file, and then display the contents of the selected file in a JTextArea.

import javax.swing.;

import java.awt.;

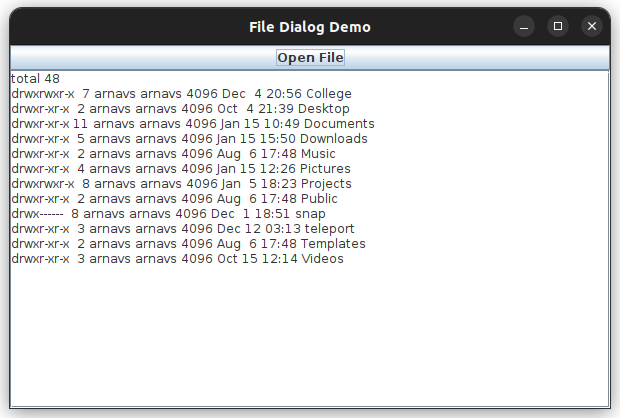
import java.awt.event.;

import java.io.;

import java.nio.file.\*;

public class FileDialogDemo {

public static void main(String[] args) {  
 // Create a JFrame container  
 JFrame frame = new JFrame("File Dialog Demo");  
  
 // Create a JTextArea to display file contents  
 JTextArea textArea = new JTextArea(20, 50);  
 textArea.setEditable(false); // Make it non-editable  
 JScrollPane scrollPane = new JScrollPane(textArea);  
  
 // Create a button to open the file dialog  
 JButton openButton = new JButton("Open File");  
  
 // Action listener for the Open button  
 openButton.addActionListener(e -> openFileDialog(textArea));  
  
 // Set the layout and add components  
 frame.setLayout(new BorderLayout());  
 frame.add(openButton, BorderLayout.NORTH);  
 frame.add(scrollPane, BorderLayout.CENTER);  
  
 // Set JFrame properties  
 frame.setSize(600, 400); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
  
// Method to open the file dialog and read the file contents into JTextArea  
private static void openFileDialog(JTextArea textArea) {  
 // Create a file chooser dialog  
 JFileChooser fileChooser = new JFileChooser();  
 fileChooser.setDialogTitle("Select a File");  
  
 // Show the open dialog  
 int result = fileChooser.showOpenDialog(null);  
  
 // If the user selects a file  
 if (result == JFileChooser.APPROVE\_OPTION) {  
 File selectedFile = fileChooser.getSelectedFile();  
  
 // Read the file contents and display them in JTextArea  
 try {  
 // Read the entire file content into a String  
 String content = Files.readString(selectedFile.toPath());  
 textArea.setText(content); // Set the text of the JTextArea  
 } catch (IOException e) {  
 // Handle file reading errors  
 JOptionPane.showMessageDialog(null, "Error reading file: " + e.getMessage());  
 }  
 }  
}  
}



1. Add a button to open a color chooser dialog. Change the background color of a panel based on the selected color.

This Swing application allows the user to open a color chooser dialog, select a color, and change the background color of a panel based on the selected color.

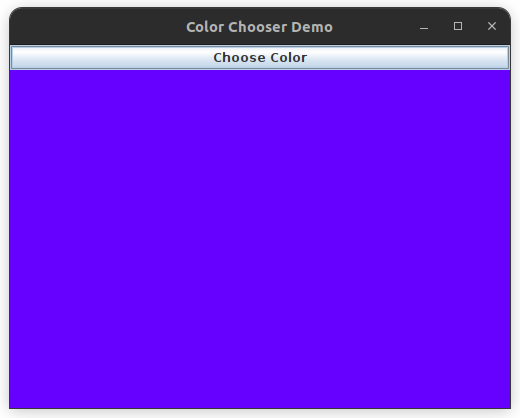
import javax.swing.;

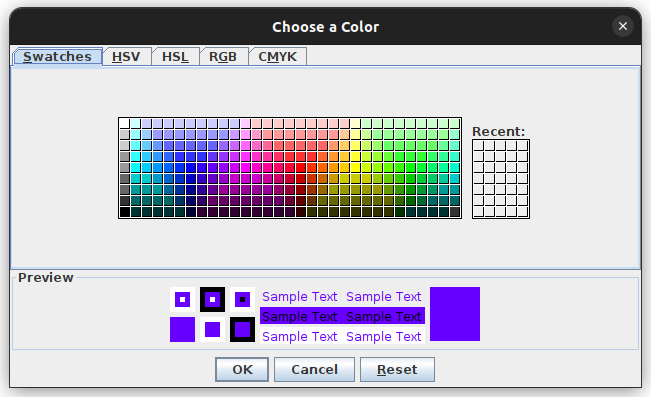
import java.awt.;

import java.awt.event.\*;

public class ColorChooserDemo {

public static void main(String[] args) {  
 JFrame frame = new JFrame("Color Chooser Demo");  
 JPanel panel = new JPanel();  
 panel.setPreferredSize(new Dimension(400, 300));  
 panel.setBackground(Color.WHITE); // Set initial background color to white  
  
 JButton colorButton = new JButton("Choose Color");  
  
 colorButton.addActionListener(e -> openColorChooserDialog(panel));  
  
 frame.setLayout(new BorderLayout());  
 frame.add(colorButton, BorderLayout.NORTH);  
 frame.add(panel, BorderLayout.CENTER);  
  
 frame.setSize(500, 400); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
  
// Method to open the color chooser dialog and change panel background color  
private static void openColorChooserDialog(JPanel panel) {  
 // Open the color chooser dialog and store the selected color  
 Color selectedColor = JColorChooser.showDialog(null, "Choose a Color", panel.getBackground());  
  
 if (selectedColor != null) {  
 panel.setBackground(selectedColor); // Change the background color of the panel  
 }  
}  
}





1. Create a GUI with JInternalFrame and a table (JTable) to display a list of students with their names and grades.

This Swing application demonstrates how to create a GUI with a JInternalFrame containing a JTable that displays a list of students with their names and grades. The JInternalFrame allows for a nested window within the main frame.

import javax.swing.;

import javax.swing.table.DefaultTableModel;

import java.awt.;

public class InternalFrameTableDemo {

public static void main(String[] args) {  
 // Create the main JFrame container  
 JFrame frame = new JFrame("Internal Frame with Table");  
  
 // Set the layout and size for the main frame  
 frame.setLayout(new BorderLayout());  
 frame.setSize(600, 400);  
  
 // Create a JDesktopPane to hold internal frames  
 JDesktopPane desktopPane = new JDesktopPane();  
  
 // Create an internal frame with a title  
 JInternalFrame internalFrame = new JInternalFrame("Student Grades", true, true, true, true);  
 internalFrame.setSize(400, 300);  
 internalFrame.setClosable(true);  
 internalFrame.setMaximizable(true);  
 internalFrame.setResizable(true);  
  
 // Create the JTable with student data  
 String[] columnNames = {"Name", "Grade"};  
 Object[][] data = {  
 {"John", "A"},  
 {"Emma", "B"},  
 {"Michael", "C"},  
 {"Sophia", "B+"},  
 {"James", "A-"}  
 };  
  
 DefaultTableModel model = new DefaultTableModel(data, columnNames);  
 JTable table = new JTable(model);  
   
 // Add the table to a JScrollPane  
 JScrollPane scrollPane = new JScrollPane(table);  
   
 // Add the JScrollPane containing the JTable to the internal frame  
 internalFrame.add(scrollPane, BorderLayout.CENTER);  
   
 // Make the internal frame visible  
 internalFrame.setVisible(true);  
  
 // Add the internal frame to the desktop pane  
 desktopPane.add(internalFrame);  
  
 // Add the desktop pane to the main frame  
 frame.add(desktopPane, BorderLayout.CENTER);  
  
 // Set the frame properties  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 frame.setVisible(true);  
}  
}

