25. 


```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ColorGUI implements ActionListener {
    public static JButton btn1;
    public static JLabel txt1;
    public static JButton btn2;
    public static JLabel txt2;
    public static JButton btn3;
    public static JLabel txt3;
    public static JButton btn4;
    public static JLabel txt4;
    public static JButton btn5;
    public static JLabel txt5;
    public static JButton btn6;

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setSize(700, 500);

        JPanel panel = new JPanel();
        panel.setLayout(null);

        btn1 = new JButton("All The Best");
        btn1.setBounds(50, 20, 150, 25);
        btn1.setForeground(Color.RED);
```

```
btn1.addActionListener(new ColorGUI());
panel.add(btn1);

txt1 = new JLabel("All The Best");
txt1.setBounds(100, 100, 150, 25);
txt1.setForeground(Color.RED);
panel.add(txt1);

btn2 = new JButton("All The Best");
btn2.setBounds(200, 20, 150, 25);
btn2.setForeground(Color.pink);
btn2.addActionListener(new ColorGUI());

txt2 = new JLabel("All The Best");
txt2.setBounds(200, 150, 150, 25);
txt2.setForeground(Color.pink);
panel.add(txt2);
panel.add(btn2);

btn3 = new JButton("All The Best");
btn3.setBounds(350, 20, 150, 25);
btn3.setForeground(Color.blue);
btn3.addActionListener(new ColorGUI());

txt3 = new JLabel("All The Best");
txt3.setBounds(300, 200, 150, 25);
txt3.setForeground(Color.blue);
panel.add(txt3);

panel.add(btn3);

btn4 = new JButton("All The Best");
btn4.setBounds(500, 20, 150, 25);
btn4.setForeground(Color.green);
```

```
btn4.addActionListener(new ColorGUI());

txt4 = new JLabel("All The Best");
txt4.setBounds(400, 250, 150, 25);
txt4.setForeground(Color.green);
panel.add(txt4);

panel.add(btn4);

btn5 = new JButton("All The Best");
btn5.setBounds(275, 50, 150, 25);
btn5.setForeground(Color.cyan);
btn5.addActionListener(new ColorGUI());

txt5 = new JLabel("All The Best");
txt5.setBounds(500, 300, 150, 25);
txt5.setForeground(Color.cyan);
panel.add(btn5);
panel.add(txt5);

btn6 = new JButton("Clear");
btn6.setBounds(600, 400, 100, 25);
btn6.setForeground(Color.black);
btn6.addActionListener(new ColorGUI());
panel.add(btn6);

txt1.setVisible(false);
txt2.setVisible(false);
txt3.setVisible(false);
txt4.setVisible(false);
txt5.setVisible(false);
frame.add(panel);
frame.setResizable(false);
frame.setVisible(true);
```

```
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == btn6){
        txt1.setVisible(false);
        txt2.setVisible(false);
        txt3.setVisible(false);
        txt4.setVisible(false);
        txt5.setVisible(false);

        }
        JButton[] btns = { btn1, btn2, btn3, btn4, btn5 };
        JLabel[] txts = { txt1, txt2, txt3, txt4, txt5 };
        for (int i = 0; i < 5; i++) {
            if (e.getSource() == btns[i]) {
                txts[i].setVisible(true);
            }
        }
    }
}

/*
```

Sure! Here's a line-by-line explanation of the code:

1. The code starts with importing necessary classes from the `javax.swing` and `java.awt` packages.

3. The class `ColorGUI` is declared. It implements the `ActionListener` interface, which allows it to handle action events.

5. The `main` method is defined as the entry point of the program.

7. An instance of `JFrame` is created to represent the main window of the GUI.

8. The size of the frame is set to 700 pixels wide and 500 pixels tall.

10. A new `JPanel` instance called `panel` is created. The layout is set to `null` (absolute positioning).

13. The first button, `btn1`, is created with the label "All The Best" and positioned using the `setBounds` method.

14. The text color of `btn1` is set to red using `setForeground` method.

15. An `ActionListener` is added to `btn1` using `addActionListener`, and the `ColorGUI` instance itself is used as the listener.

16. `btn1` is added to the `panel`.

19. A `JLabel` called `txt1` is created with the text "All The Best" and positioned using `setBounds`.

20. The text color of `txt1` is set to red.

21. `txt1` is added to the `panel`.

23. The same process is repeated for `btn2` and `txt2`, with the only difference being the position and color.

26. The same process is repeated for `btn3` and `txt3`, with the only difference being the position and color.

29. The same process is repeated for `btn4` and `txt4`, with the only difference being the position and color.

32. The same process is repeated for `btn5` and `txt5`, with the only difference being the position and color.

35. `btn6` is created with the label "Clear" and positioned using `setBounds`.

36. The text color of `btn6` is set to black.

37. An `ActionListener` is added to `btn6` using `addActionListener`, and the `ColorGUI` instance itself is used as the listener.

38. `btn6` is added to the `panel`.

41. The visibility of `txt1`, `txt2`, `txt3`, `txt4`, and `txt5` is initially set to `false`.

43. The `panel` is added to the `frame`.

44. The frame is set to not be resizable.

45. The frame is set to be visible.

49. The `actionPerformed` method is overridden from the `ActionListener` interface.

50. It checks if the source of the action event is `btn6`, indicating that the "Clear" button was clicked.

51. If `btn6` was clicked, the visibility of `txt1`, `txt2`, `txt3`, `txt4`, and `txt5` is set to `false`.

54. An array of `JButton` objects, `btns`, is created containing `btn1`, `btn2`, `btn3`, `btn4`, and `btn5`.

55. An array of `JLabel` objects, `txts`, is created containing `txt1`, `txt2`, `txt3`, `txt4`, and `txt5`.

56. A loop is used to iterate over the buttons and labels.

57. If the source of the action event matches the current button, the corresponding label is set to visible.

60. The line-by-line explanation of the code

The ColorGUI class extends the Frame class, which represents a top-level window with a title bar and border. It contains several Label components that display the text "All The Best" in different colors.

In the constructor of the ColorGUI class, the labels (lbl1, lbl2, lbl3, lbl4, lbl5) are initialized and configured with different foreground colors using the setForeground() method from the java.awt.Color class. These labels are then added to the frame using the add() method.

The setVisible(true) method is called to make the frame visible, and the setSize() method is used to set the size of the frame to 400 pixels by 300 pixels. The setLayout() method sets the layout manager for the frame to FlowLayout, which arranges the components in a horizontal flow.

The paint() method is overridden to draw text using the Graphics object. The setColor() method from the java.awt.Graphics class is used to set the color, and the drawString() method is used to display the text "All The Best" at different positions on the frame.

Finally, the main() method creates an instance of the ColorGUI class, which initializes and displays the GUI window.

The getSource() method is a method defined in the ActionEvent class, which is a subclass of EventObject. In the context of the ActionListener interface, the getSource() method is used to determine the source of the event that triggered the action.

In the ColorGUI class, the getSource() method is used in the actionPerformed() method to identify which button triggered the action event. It compares the source of the event (e.getSource()) with each button (btn1, btn2, btn3, btn4, btn5) using a loop.

*/

24. Create a class Student with attributes roll no, name, age and course. Initialize values through parameterized constructor. If age of student is not in between 15 and 21 then generate user-defined exception "AgeNotWithinRangeException". If name contains numbers or special symbols raise exception "NameNotValidException". Define the two exception classes.

```
class AgeNotWithinRangeException extends Exception {
    public AgeNotWithinRangeException(String message) {
        super(message);
    }
}

class NameNotValidException extends Exception {
    public NameNotValidException(String message) {
        super(message);
    }
}

class Student {
    private int rollNo;
    private String name;
    private int age;
    private String course;

    public Student(int rollNo, String name, int age, String course) throws AgeNotWithinRangeException,
NameNotValidException {
        this.rollNo = rollNo;
        if (age < 15 || age > 21) {
            throw new AgeNotWithinRangeException("Age is not within the range of 15 to 21.");
        }
        this.age = age;

        if (!isChar(name)) {
            throw new NameNotValidException("Name is not valid. It should only contain alphabets and
spaces.");
        }
```

```
        this.name = name;

        this.course = course;
    }

    public static boolean isChar(String str) {
        for (int i = 0; i < str.length(); i++) {
            if (!Character.isLetter(str.charAt(i)) || !Character.isAlphabetic(str.charAt(i))) {
                return false;
            }
        }
        return true;
    }

    public int getRollNo() {
        return rollNo;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getCourse() {
        return course;
    }
}

public class StudentAgeNameException {
    public static void main(String[] args) {
        try {
```

```
            Student student1 = new Student(1, "John Doe", 18, "Computer Science");
            System.out.println("Student 1:");
            System.out.println("Roll No: " + student1.getRollNo());
            System.out.println("Name: " + student1.getName());
            System.out.println("Age: " + student1.getAge());
            System.out.println("Course: " + student1.getCourse());
            System.out.println();

            Student student2 = new Student(2, "Jane Smith", 14, "Electrical Engineering");
            System.out.println("Student 2:");
            System.out.println("Roll No: " + student2.getRollNo());
            System.out.println("Name: " + student2.getName());
            System.out.println("Age: " + student2.getAge());
            System.out.println("Course: " + student2.getCourse());
            System.out.println();
        } catch (AgeNotWithinRangeException e) {
            System.out.println("Exception: " + e.getMessage());
        } catch (NameNotValidException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
/*
```

Certainly! Here's a line-by-line explanation of the code:

1. The code defines a custom exception class `AgeNotWithinRangeException` that extends the `Exception` class.

3. The `AgeNotWithinRangeException` class has a constructor that takes a `String` parameter `message` and passes it to the superclass (`Exception`) constructor using `super(message)`.

7. The code defines another custom exception class `NameNotValidException` that extends the `Exception` class.

9. The `NameNotValidException` class has a constructor that takes a `String` parameter `message` and passes it to the superclass (`Exception`) constructor using `super(message)`.

13. The `Student` class is defined. It encapsulates the properties of a student: `rollNo`, `name`, `age`, and `course`.

15. The `Student` class constructor is defined. It takes `rollNo`, `name`, `age`, and `course` as parameters. It also declares that it throws `AgeNotWithinRangeException` and `NameNotValidException`, which means the caller of this constructor must handle these exceptions.

17. The `rollNo` is initialized with the value passed as an argument.

18. The `age` is validated using an `if` statement. If the age is not within the range of 15 to 21, an `AgeNotWithinRangeException` is thrown with an appropriate message.

22. The `age` is assigned the value passed as an argument after it passes the validation.

23. The `name` is validated using a helper method `isChar()`. If the name contains any character that is not a letter or alphabetic, a `NameNotValidException` is thrown with an appropriate message.

28. The `name` is assigned the value passed as an argument after it passes the validation.

30. The `course` is assigned the value passed as an argument.

34. The `isChar()` method is defined. It takes a `String` parameter `str` and checks if each character in the string is a letter or alphabetic. It returns `true` if all characters pass the test; otherwise, it returns `false`.

39. The `getRollNo()` method is defined. It returns the `rollNo` of the student.

43. The `getName()` method is defined. It returns the `name` of the student.

47. The `getAge()` method is defined. It returns the `age` of the student.

51. The `getCourse()` method is defined. It returns the `course` of the student.

55. The `StudentAgeNameException` class is defined. It contains the `main` method, which serves as the entry point of the program.

57. Inside the `main` method, two `Student` objects (`student1` and `student2`) are created using the `Student` constructor.

58-62. The properties of `student1` and `student2` are printed to the console using `System.out.println()` statements.

65-66. The code includes exception handling using `try-catch` blocks.

68-71. If an `AgeNotWithinRangeException` is thrown during the creation of `student1` or `student2`, the corresponding exception message is printed to the console.

73-76. If a `NameNotValidException` is thrown during the creation of `student1` or `student2`, the corresponding exception message is printed to the console.

By using custom exceptions, the `Student` class can validate the age and name of the student objects during their creation and handle any invalid input accordingly in the `main` method.

The main aim of this program is to demonstrate exception handling and validation in Java. The program focuses on creating a `Student` object with specific properties (`rollNo`, `name`, `age`, and `course`), but it enforces certain rules and constraints on the input values.

The program utilizes custom exception classes, `AgeNotWithinRangeException` and `NameNotValidException`, to handle cases where the age is not within the range of 15 to 21 or the name contains characters other than letters and spaces.

By throwing and catching these custom exceptions, the program ensures that the `Student` objects are created with valid data. If any input violates the specified constraints, an appropriate exception is thrown and caught, allowing the program to handle the exceptional condition gracefully and provide meaningful error messages.

The program aims to showcase the usage of custom exceptions, exception propagation, and how to handle exceptions using `try-catch` blocks.

The code defines two custom exception classes: AgeNotWithinRangeException and NameNotValidException, which both extend the Exception class. These exceptions are used to handle specific error conditions that can occur when creating a Student object.

The Student class represents a student with attributes such as rollNo, name, age, and course. The constructor of the Student class throws the AgeNotWithinRangeException and NameNotValidException if the provided age or name does not meet certain criteria.

In the Student class, the isNameValid method is used to validate the name by checking if it contains only alphabets and spaces using a regular expression ([a-zA-Z]+).

The Main class contains the main method, which demonstrates the usage of the Student class. It creates two Student objects, passing the required information to the constructor. If any exception is thrown during the creation of a Student object, it is caught and the corresponding error message is displayed.

*/

23. Write a Java program to find the length of the longest consecutive elements sequence from an unsorted array of integers.
Sample array: [49, 1, 3, 200, 2, 4, 70, 5]
The longest consecutive elements sequence is [1, 2, 3, 4, 5], therefore the program will return its length 5.

```
import java.util.Scanner;


public class ConsecutiveNumbers {
    public static int longestConsecutive(int[] nums) {
        int maxLength = 0;

        for (int num : nums) {
            int currentNum = num;
            int currentLength = 1;

            // Check for consecutive elements by incrementing the current number
            while (contains(nums, currentNum + 1)) {
                currentNum++;
                currentLength++;
            }

            maxLength = Math.max(maxLength, currentLength);
        }

        return maxLength;
    }

    // Helper method to check if the array contains a specific number
    public static boolean contains(int[] nums, int target) {
        for (int num : nums) {
            if (num == target) {
                return true;
            }
        }
        return false;
```

```
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        int[] nums = new int[n];
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }

        int length = longestConsecutive(nums);
        System.out.println("The length of the longest consecutive elements sequence is: " + length);

        scanner.close();
    }
}

/*
```

The longestConsecutive method takes an array of integers (nums) as input and returns an integer representing the length of the longest consecutive sequence. It uses a HashSet called numSet to store the numbers from the input array for efficient lookup.

Here's a step-by-step explanation of how the code works:

Create a HashSet called numSet to store the numbers.

Initialize maxLength to 0, which will keep track of the maximum length of a consecutive sequence.

Iterate over the nums array and add each number to the numSet.

Iterate over the nums array again.

For each number num, check if num - 1 is not present in numSet. If true, it means num is the start of a new consecutive sequence.

Initialize currentNum as num and currentLength as 1.

While numSet contains currentNum + 1, increment currentNum and currentLength.

Compare maxLength with currentLength and update maxLength with the maximum value.

Repeat steps 5-8 for each number in the nums array.

Return the value of maxLength.

In the main method, an example array nums is defined. The longestConsecutive method is called with this array, and the returned length is printed to the console.

import java.util.Scanner;: Imports the Scanner class from the java.util package to allow user input.

public class ConsecutiveNumbers {: Defines the ConsecutiveNumbers class.

public static int longestConsecutive(int[] nums) {: Defines the longestConsecutive method that takes an array of integers (nums) as a parameter and returns an integer. This method calculates the length of the longest consecutive elements sequence in the array.

int maxLength = 0;: Initializes the maxLength variable to 0, which will store the maximum length of consecutive elements.

for (int num : nums) {: Iterates over each element num in the nums array.

int currentNum = num;: Initializes the currentNum variable to the current element being iterated.

int currentLength = 1;: Initializes the currentLength variable to 1, as the current element itself is considered part of the consecutive sequence.

while (contains(nums, currentNum + 1)) {: Enters a loop that checks if the next consecutive number exists in the array by calling the contains method.

currentNum++;: Increments the currentNum variable to check the next number.

currentLength++;: Increments the currentLength variable to track the length of the consecutive sequence.

maxLength = Math.max(maxLength, currentLength);: Updates the maxLength variable with the maximum length encountered so far.

return maxLength;: Returns the final maximum length of the consecutive sequence.

public static boolean contains(int[] nums, int target) {: Defines the contains method that takes an array of integers (nums) and a target number (target) as parameters and returns a boolean value. This method checks if the target number exists in the array.

for (int num : nums) {: Iterates over each element num in the nums array.

if (num == target) {: Checks if the current element is equal to the target number.

return true;: Returns true if the target number is found in the array.

`return false

*/

22. Write a Java program to create a class known as Person with methods called getFirstName() and getLastName(). Create a subclass called Employee that adds a new method named getEmployeeId() and overrides the getLastName() method to include the employee's job title.

```
import java.util.Scanner;

class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}

class Employee extends Person {
    private int employeeId;
    private String jobTitle;

    public Employee(String firstName, String lastName, int employeeId, String jobTitle) {
        super(firstName, lastName);
        this.employeeId = employeeId;
        this.jobTitle = jobTitle;
    }
```

```
    public int getEmployeeId() {
        return employeeId;
    }

    public String getLastName() {
        return super.getLastName() + ", " + jobTitle;
    }
}

public class EmployeeName {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter person's first name: ");
        String firstName = scanner.nextLine();

        System.out.print("Enter person's last name: ");
        String lastName = scanner.nextLine();

        Person person = new Person(firstName, lastName);
        System.out.println("Person: " + person.getFirstName() + " " + person.getLastName());

        System.out.print("Enter employee's first name: ");
        firstName = scanner.nextLine();

        System.out.print("Enter employee's last name: ");
        lastName = scanner.nextLine();

        System.out.print("Enter employee ID: ");
        int employeeId = scanner.nextInt();
        scanner.nextLine(); // Consume the remaining newline character

        System.out.print("Enter employee's job title: ");
        String jobTitle = scanner.nextLine();
```

```
        Employee employee = new Employee(firstName, lastName, employeeId, jobTitle);
        System.out.println("Employee: " + employee.getFirstName() + " " + employee.getLastName());
        System.out.println("Employee ID: " + employee.getEmployeeId());

        scanner.close();
    }
}

/*
Sure! Here's the line-by-line explanation of the code:

```java
import java.util.Scanner;

class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

- The `Person` class represents a person and has private fields `firstName` and `lastName`.
```

- The constructor `Person(String firstName, String lastName)` initializes the `firstName` and `lastName` fields.

- The `getFirstName()` and `getLastName()` methods provide access to the `firstName` and `lastName` fields, respectively.

```java
class Employee extends Person {
    private int employeeId;
    private String jobTitle;

    public Employee(String firstName, String lastName, int employeeId, String jobTitle) {
        super(firstName, lastName);
        this.employeeId = employeeId;
        this.jobTitle = jobTitle;
    }

    public int getEmployeeId() {
        return employeeId;
    }

    public String getLastName() {
        return super.getLastName() + ", " + jobTitle;
    }
}
```

- The `Employee` class extends the `Person` class and adds additional fields `employeeId` and `jobTitle`.

- The constructor `Employee(String firstName, String lastName, int employeeId, String jobTitle)` initializes the `firstName`, `lastName`, `employeeId`, and `jobTitle` fields.

- The `getEmployeeId()` method returns the employee ID.

- The `getLastName()` method overrides the `getLastName()` method of the `Person` class to include the employee's job title in the last name.

```java
public class EmployeeName {
    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter person's first name: ");
        String firstName = scanner.nextLine();

        System.out.print("Enter person's last name: ");
        String lastName = scanner.nextLine();

        Person person = new Person(firstName, lastName);
        System.out.println("Person: " + person.getFirstName() + " " + person.getLastName());
```

- The `main()` method is the entry point of the program.
- A `Scanner` object `scanner` is created to read user input.
- The user is prompted to enter the person's first name and last name, which are stored in the variables `firstName` and `lastName`.
- An instance of the `Person` class called `person` is created using the provided first name and last name.
- The person's information is printed using `person.getFirstName()` and `person.getLastName()`.

```java
        System.out.print("Enter employee's first name: ");
        firstName = scanner.nextLine();

        System.out.print("Enter employee's last name: ");
        lastName = scanner.nextLine();

        System.out.print("Enter employee ID: ");
        int employeeId = scanner.nextInt();
        scanner.nextLine(); // Consume the remaining newline character

        System.out.print("Enter employee's job title: ");
        String jobTitle = scanner.nextLine();

        Employee employee = new Employee(firstName, lastName, employeeId, jobTitle);
        System.out.println("Employee: " + employee.getFirstName() + " " + employee.getLastName());
```

```
        System.out.println("Employee ID: " + employee.getEmployeeId());

        scanner.close();
    }
}

/*
Sure! Here's the line-by-line explanation of the code:

```java
import java.util.Scanner;

class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

- The `Person` class represents a person and has private fields `firstName` and `lastName`.

- The constructor `Person(String firstName, String lastName)` initializes the `firstName` and `lastName` fields.
```

- The `getFirstName()` and `getLastName()` methods provide access to the `firstName` and `lastName` fields, respectively.

```java
class Employee extends Person {
    private int employeeId;
    private String jobTitle;

    public Employee(String firstName, String lastName, int employeeId, String jobTitle) {
        super(firstName, lastName);
        this.employeeId = employeeId;
        this.jobTitle = jobTitle;
    }

    public int getEmployeeId() {
        return employeeId;
    }

    public String getLastName() {
        return super.getLastName() + ", " + jobTitle;
    }
}
```

- The `Employee` class extends the `Person` class and adds additional fields `employeeId` and `jobTitle`.

- The constructor `Employee(String firstName, String lastName, int employeeId, String jobTitle)` initializes the `firstName`, `lastName`, `employeeId`, and `jobTitle` fields.

- The `getEmployeeId()` method returns the employee ID.

- The `getLastName()` method overrides the `getLastName()` method of the `Person` class to include the employee's job title in the last name.

```java
public class EmployeeName {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Enter person's first name: ");
        String firstName = scanner.nextLine();

        System.out.print("Enter person's last name: ");
        String lastName = scanner.nextLine();

        Person person = new Person(firstName, lastName);
        System.out.println("Person: " + person.getFirstName() + " " + person.getLastName());
```

- The `main()` method is the entry point of the program.
- A `Scanner` object `scanner` is created to read user input.
- The user is prompted to enter the person's first name and last name, which are stored in the variables `firstName` and `lastName`.
- An instance of the `Person` class called `person` is created using the provided first name and last name.
- The person's information is printed using `person.getFirstName()` and `person.getLastName()`.

```java
        System.out.print("Enter employee's first name: ");
        firstName = scanner.nextLine();

        System.out.print("Enter employee's last name: ");
        lastName = scanner.nextLine();

        System.out.print("Enter employee ID: ");
        int employeeId = scanner.nextInt();
        scanner.nextLine(); // Consume the remaining newline character

        System.out.print("Enter employee's job title: ");
        String jobTitle = scanner.nextLine();

        Employee employee = new Employee(firstName, lastName, employeeId, jobTitle);
        System.out.println("Employee: " + employee.getFirstName() + " " + employee.getLastName());
        System.out.println("Employee ID: " + employee.getEmployeeId());
```

```
        scanner.close();
    }
}
```

- The user is prompted to enter the employee's first name, last name, employee ID, and job title, which are stored in the respective variables.

- An instance of the `Employee` class called `employee` is created using the provided information

*/

21. Write a Java program to create a class called "Student" with a name, grade, and courses attributes, and methods to add and remove courses.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class Student {
    private String name;
    private int grade;
    private List<String> courses;

    public Student(String name, int grade) {
        this.name = name;
        this.grade = grade;
        this.courses = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public int getGrade() {
        return grade;
    }

    public List<String> getCourses() {
        return courses;
    }

    public void addCourse(String course) {
        courses.add(course);
    }
```

```
    public void removeCourse(String course) {
        if (courses.contains(course)) {
            courses.remove(course);
            System.out.println("Course '" + course + "' removed.");
        } else {
            System.out.println("Course '" + course + "' is not present.");
        }
    }
}

class StudentCourseManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter student's name: ");
        String name = scanner.nextLine();

        System.out.print("Enter student's grade: ");
        int grade = scanner.nextInt();
        scanner.nextLine(); // Consume the remaining newline character

        Student student = new Student(name, grade);

        System.out.println("Student details:");
        System.out.println("Name: " + student.getName());
        System.out.println("Grade: " + student.getGrade());

        while (true) {
            System.out.print("Enter 'add' to add a course, 'remove' to remove a course, or 'quit' to exit: ");
            String input = scanner.nextLine();

            if (input.equals("add")) {
                System.out.print("Enter course name to add: ");
                String course = scanner.nextLine();
```

```
            student.addCourse(course);
            System.out.println("Course '" + course + "' added.");
        } else if (input.equals("remove")) {
            System.out.print("Enter course name to remove: ");
            String course = scanner.nextLine();
            student.removeCourse(course);
        } else if (input.equals("quit")) {
            break;
        } else {
            System.out.println("Invalid input. Try again.");
        }
    }

    System.out.println("Final student details:");
    System.out.println("Name: " + student.getName());
    System.out.println("Grade: " + student.getGrade());
    System.out.println("Courses: " + student.getCourses());

    scanner.close();
  }
}

/*
```

The given Java code represents a student course management system.

The `Student` class encapsulates the information and behavior related to a student. It has private attributes such as `name`, `grade`, and `courses`, where `name` stores the student's name, `grade` stores the student's grade level, and `courses` stores a list of courses the student is enrolled in. The constructor initializes the student object with the provided name and grade, and initializes the `courses` list as an empty `ArrayList`.

The class provides getter methods (`getName`, `getGrade`, `getCourses`) to access the student's information. It also provides methods `addCourse` and `removeCourse` to add and remove courses from the student's course list. The `addCourse` method takes a course name as input and adds it to the `courses` list. The `removeCourse` method also takes a course name as input, checks if the course is present in the `courses` list, and removes it if found. If the course is not present, it displays a message indicating that the course is not present in the list.

The `StudentCourseManagement` class contains the `main` method, which serves as the entry point of the program. It creates a `Scanner` object to read user input. It prompts the user to enter the student's name and grade, and creates a `Student` object using the provided information.

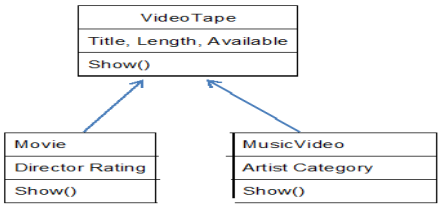
After creating the student object, it displays the student's details including the name and grade. It then enters a loop where it prompts the user to enter options such as 'add' to add a course, 'remove' to remove a course, or 'quit' to exit the program. Based on the user's input, it calls the appropriate methods of the `Student` object to perform the corresponding actions. If the user chooses to add a course, it prompts for the course name and adds it to the student's course list. If the user chooses to remove a course, it prompts for the course name and removes it from the student's course list using the `removeCourse` method.

The loop continues until the user chooses to quit by entering 'quit'. After exiting the loop, it displays the final details of the student, including the name, grade, and the list of courses the student is enrolled in.

Overall, this code provides a basic student course management system that allows the user to add and remove courses for a student and displays the final details after the user is done managing the courses.

*/

20. Write a program, to implement the following hierarchy. Displays information of each class the rectangle represents the classes. The classes Movie and MusicVideo inherits all the members of the class VideoTape.
"




```
class VideoTape {
    protected String title;
    protected int duration;

    public VideoTape(String title, int duration) {
        this.title = title;
        this.duration = duration;
    }

    public void play() {
        System.out.println("Playing " + title + " (Duration: " + duration + " minutes)");
    }
}

class Movie extends VideoTape {
    private String director;

    public Movie(String title, int duration, String director) {
        super(title, duration);
        this.director = director;
    }

    public void displayInfo() {
        System.out.println("Movie Title: " + title);
        System.out.println("Duration: " + duration + " minutes");
        System.out.println("Director: " + director);
    }
```

```
}

class MusicVideo extends VideoTape {
    private String artist;

    public MusicVideo(String title, int duration, String artist) {
        super(title, duration);
        this.artist = artist;
    }

    public void showDetails() {
        System.out.println("Music Video Title: " + title);
        System.out.println("Duration: " + duration + " minutes");
        System.out.println("Artist: " + artist);
    }
}

class VideoHierarchy {
    public static void main(String[] args) {
        Movie movie = new Movie("The Shawshank Redemption", 142, "Frank Darabont");
        movie.displayInfo();
        movie.play();

        System.out.println();

        MusicVideo musicVideo = new MusicVideo("Shape of You", 235, "Ed Sheeran");
        musicVideo.showDetails();
        musicVideo.play();
    }
}



/*
The given Java code defines a hierarchy of classes related to video tapes.
```

The `VideoTape` class serves as the base class and contains the `title` and `duration` attributes, representing the title of the video tape and its duration in minutes, respectively. It also has a `play` method, which simply displays a message indicating that the video tape is being played, including the title and duration.

The `Movie` class is a subclass of `VideoTape` and adds an additional attribute called `director`. It inherits the `title` and `duration` attributes from the base class using the `super` keyword in its constructor. It also introduces a new method called `displayInfo`, which displays detailed information about the movie, including the title, duration, and director.

The `MusicVideo` class is another subclass of `VideoTape` and introduces the `artist` attribute. Similar to the `Movie` class, it inherits the `title` and `duration` attributes from the base class and introduces a method called `showDetails`, which displays information about the music video, including the title, duration, and artist.

The `VideoHierarchy` class contains the `main` method, which serves as the entry point of the program. Inside the `main` method, it creates instances of both `Movie` and `MusicVideo` classes, passing the relevant information such as the title, duration, and director/artist to their respective constructors. It then calls the appropriate methods (`displayInfo`, `showDetails`, and `play`) on these objects to demonstrate their functionalities.

When executed, the program outputs information about the movie "The Shawshank Redemption" with its director and plays it. Then, it displays information about the music video "Shape of You" with its artist and plays it as well.

Overall, this code showcases the concept of inheritance and method overriding in object-oriented programming, where subclasses inherit attributes and behaviors from a base class and can introduce their own unique features or override existing methods.

*/

17. Write a program to read 10 string from console and then print the sorted strings on console (Use String Class).2) combine two string 3) reverse first string nd dispaly it .

```
import java.util.Arrays;
import java.util.Scanner;

public class StringOperations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] strings = new String[10];

        // Read 10 strings from console
        System.out.println("Enter 10 strings:");
        for (int i = 0; i < 10; i++) {
            strings[i] = scanner.nextLine();
        }

        // Sort the strings
        Arrays.sort(strings);

        System.out.println("Sorted strings:");
        for (String str : strings) {
            System.out.println(str);
        }

        // Combine two strings
        System.out.println("Combine two strings:");
        String combined = strings[0] + strings[1];
        System.out.println(combined);

        // Reverse the first string
        System.out.println("Reverse first string:");
        String reversed = reverseString(strings[0]);
        System.out.println(reversed);
```

```
        scanner.close();
    }

    public static String reverseString(String str) {
        StringBuilder reversed = new StringBuilder(str);
        return reversed.reverse().toString();
    }
}
```

/*

The given Java code represents a program that performs various operations on strings.

The `StringOperations` class contains the `main` method, which serves as the entry point of the program. It creates a `Scanner` object to read user input and initializes a string array `strings` with a size of 10.

The program then prompts the user to enter 10 strings one by one. It uses a loop to read each string from the console and stores them in the `strings` array.

After reading the strings, the program sorts the strings using the `Arrays.sort` method. This arranges the strings in lexicographical order.

Next, it prints the sorted strings by iterating over the `strings` array and displaying each string on a separate line.

The program then combines the first two strings from the sorted array using the `+` operator and stores the result in the `combined` variable. It prints the combined string.

Next, the program calls the `reverseString` method, passing the first string from the sorted array as an argument. The `reverseString` method takes a string and uses a `StringBuilder` to reverse the characters in the string. The reversed string is then returned and stored in the `reversed` variable.

Finally, the program prints the reversed string.

In summary, this program reads 10 strings from the user, sorts them, combines the first two strings, and reverses the characters in the first string. It then displays the sorted strings, the combined string, and the reversed string.*/

16. Write a Java Program to count the number of words in a string using HashMap.Output:
Input: Enter String: "This this is is done by Saket Saket";
{Saket=2, by=1, this=1, This=1, is=2, done=1}

```
import java.util.*;
import java.util.HashMap;

public class FinalCountWords {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String str = scanner.nextLine();
        String[] split = str.split(" ");
        HashMap<String, Integer> map = new HashMap<String, Integer>();

        for (int i = 0; i < split.length; i++) {
            if (map.containsKey(split[i])) {
                int count = map.get(split[i]);
                map.put(split[i], count + 1);
            } else {
                map.put(split[i], 1);
            }
        }

        System.out.println(map);
        scanner.close();
    }
}

/*
```

The provided Java code implements a program that counts the occurrences of words in a given string. Here's a paragraph explanation of the code:

The program starts by creating a `Scanner` object to read input from the console. It prompts the user to enter a string and stores it in the `str` variable. The input string is then split into individual words using the `split` method, which splits the string based on spaces.

Next, a `HashMap` named `map` is created. This map will store the words as keys and their corresponding counts as values. The key-value pairs will represent the word and the number of times it appears in the string.

The program enters a loop that iterates through each word in the `split` array. For each word, it checks if the `map` already contains the word as a key using the `containsKey` method. If the word is present, it retrieves the current count value associated with that word using `get`, increments it by 1, and updates the entry in the map using `put`. If the word is not present in the map, it adds a new entry with the word as the key and an initial count of 1.

Once the loop finishes processing all the words, the program prints the contents of the `map`, which displays each unique word along with its corresponding count.

Finally, the `Scanner` is closed to release system resources.

In summary, this program takes a string input, counts the occurrences of each word in the string using a `HashMap`, and prints the word count.

*/

15. Write a Java Program to iterate ArrayList using for-loop, iterator, and advance for-loop. Insert 3 Array List.Input 20 30 40Output:

```
iterator Loop:
20
30
40
Advanced For Loop:
20
30
40
For Loop:
20
30
40
```

```
import java.util.ArrayList;

import java.util.Iterator;

import java.util.Scanner;


public class ArrayListIterationExample {

    public static void main(String[] args) {

        // Create an ArrayList

        ArrayList<Integer> numbers = new ArrayList<>();


        // Get user input

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");

        int count = scanner.nextInt();

        System.out.println("Enter the elements:");

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

            int element = scanner.nextInt();

            numbers.add(element);

        }


        // Iterate using iterator loop

        System.out.println("Iterator Loop:");

        Iterator<Integer> iterator = numbers.iterator();

        while (iterator.hasNext()) {

            System.out.println(iterator.next());

        }


        // Iterate using advanced for loop
```

```
        System.out.println("Advanced For Loop:");
        for (int number : numbers) {
            System.out.println(number);
        }

        // Iterate using regular for loop
        System.out.println("For Loop:");
        for (int i = 0; i < numbers.size(); i++) {
            System.out.println(numbers.get(i));
        }

        scanner.close();
    }
}

/*
```

The provided Java code demonstrates different ways to iterate over an `ArrayList` of integers. Here's a paragraph explanation of the code, along with an explanation of the differences between the iterator loop, advanced for loop, and regular for loop:

The program begins by creating an `ArrayList` called `numbers` to store integers. It then prompts the user to enter the number of elements they want to add to the list. After receiving the count, the program proceeds to read the elements from the user and adds them to the `numbers` list.

Next, the program demonstrates three different methods of iterating over the `ArrayList`.

1. Iterator Loop:

   This loop uses an `Iterator` object obtained from the `numbers` list using the `iterator()` method. The loop condition checks if there is a next element available using the `hasNext()` method. Inside the loop, the `next()` method is called to retrieve and print the next element from the `Iterator`. This loop allows iterating over the elements in a forward-only manner.

2. Advanced For Loop (Enhanced For Loop):

   The advanced for loop is a simplified syntax introduced in Java 5 for iterating over collections and arrays. In this loop, the loop variable (`number` in this case) represents each element of the `numbers` list in successive iterations. The loop automatically iterates through all the elements of the collection without the need for an explicit index or iterator. It provides a concise and readable way to iterate over the elements.

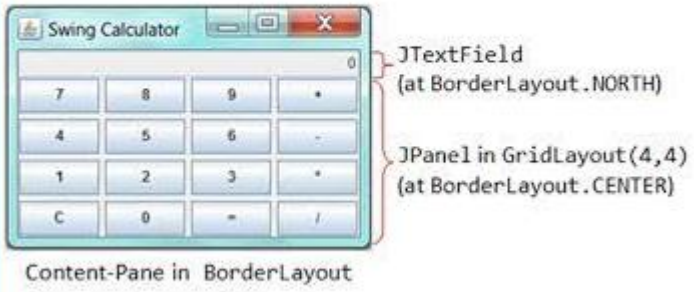
3. Regular For Loop:

   The regular for loop is a traditional loop structure that uses an index variable to access elements. In this case, the loop variable `i` serves as the index. The loop iterates from 0 to `numbers.size() - 1` (the index range of the list) and uses `numbers.get(i)` to retrieve each element at the corresponding index. This loop allows more control over the iteration process, such as accessing elements by index or iterating in reverse order.

After demonstrating the three iteration methods, the program closes the `Scanner` to release system resources.

In summary, the iterator loop, advanced for loop, and regular for loop are different ways to iterate over an `ArrayList`. The iterator loop uses an iterator object to traverse the elements, the advanced for loop simplifies the syntax for iterating over collections, and the regular for loop provides more control over the iteration process by using an index variable.

*/

14. 


```
import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class SwingCalculator implements ActionListener {
  private static JTextField inputBox;

  SwingCalculator(){}
  public static void main(String[] args) {
    createWindow();
  }

  private static void createWindow() {
    JFrame frame = new JFrame("Calculator");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    createUI(frame);
    frame.setSize(200, 200);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
```

```
}

private static void createUI(JFrame frame) {
  JPanel panel = new JPanel();
  SwingCalculator calculator = new SwingCalculator();
  GridBagLayout layout = new GridBagLayout();
  GridBagConstraints gbc = new GridBagConstraints();
  panel.setLayout(layout);

  inputBox = new JTextField(10);
  inputBox.setEditable(false);

  JButton button0 = new JButton("0");JButton button1 = new JButton("1");
  JButton button2 = new JButton("2");JButton button3 = new JButton("3");
  JButton button4 = new JButton("4");JButton button5 = new JButton("5");
  JButton button6 = new JButton("6");JButton button7 = new JButton("7");
  JButton button8 = new JButton("8");JButton button9 = new JButton("9");

  JButton buttonPlus = new JButton("+");JButton buttonMinus = new JButton("-");
  JButton buttonDivide = new JButton("/");JButton buttonMultiply = new JButton("x");
  JButton buttonClear = new JButton("C");JButton buttonDot = new JButton(".");
  JButton buttonEquals = new JButton("=");

  button1.addActionListener(calculator);button2.addActionListener(calculator);
  button3.addActionListener(calculator);button4.addActionListener(calculator);
  button5.addActionListener(calculator);button6.addActionListener(calculator);
  button7.addActionListener(calculator);button8.addActionListener(calculator);
  button9.addActionListener(calculator);button0.addActionListener(calculator);

  buttonPlus.addActionListener(calculator);buttonMinus.addActionListener(calculator);
  buttonDivide.addActionListener(calculator);buttonMultiply.addActionListener(calculator);
  buttonClear.addActionListener(calculator);buttonDot.addActionListener(calculator);
  buttonEquals.addActionListener(calculator);

  gbc.fill = GridBagConstraints.HORIZONTAL;
  gbc.gridx = 0; gbc.gridy = 0; panel.add(button1, gbc);
  gbc.gridx = 1; gbc.gridy = 0; panel.add(button2, gbc);
  gbc.gridx = 2; gbc.gridy = 0; panel.add(button3, gbc);
```

```
    gbc.gridx = 3; gbc.gridy = 0; panel.add(buttonPlus, gbc);
    gbc.gridx = 0; gbc.gridy = 1; panel.add(button4, gbc);
    gbc.gridx = 1; gbc.gridy = 1; panel.add(button5, gbc);
    gbc.gridx = 2; gbc.gridy = 1; panel.add(button6, gbc);
    gbc.gridx = 3; gbc.gridy = 1; panel.add(buttonMinus, gbc);
    gbc.gridx = 0; gbc.gridy = 2; panel.add(button7, gbc);
    gbc.gridx = 1; gbc.gridy = 2; panel.add(button8, gbc);
    gbc.gridx = 2; gbc.gridy = 2; panel.add(button9, gbc);
    gbc.gridx = 3; gbc.gridy = 2; panel.add(buttonDivide, gbc);
    gbc.gridx = 0; gbc.gridy = 3; panel.add(buttonDot, gbc);
    gbc.gridx = 1; gbc.gridy = 3; panel.add(button0, gbc);
    gbc.gridx = 2; gbc.gridy = 3; panel.add(buttonClear, gbc);
    gbc.gridx = 3; gbc.gridy = 3; panel.add(buttonMultiply, gbc);
    gbc.gridwidth = 3;

    gbc.gridx = 0; gbc.gridy = 4; panel.add(inputBox, gbc);
    gbc.gridx = 3; gbc.gridy = 4; panel.add(buttonEquals, gbc);
    frame.getContentPane().add(panel, BorderLayout.CENTER);
  }

  public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    if (command.charAt(0) == 'C') {
      inputBox.setText("");
    }else if (command.charAt(0) == '=') {
      inputBox.setText(evaluate(inputBox.getText()));
    }else {
      inputBox.setText(inputBox.getText() + command);
    }
  }

  public static String evaluate(String expression) {
    char[] arr = expression.toCharArray();
    String operand1 = "";String operand2 = "";String operator = "";
    double result = 0;

    for (int i = 0; i < arr.length; i++) {
      if (arr[i] >= '0' && arr[i] <= '9' || arr[i] == '.') {
```

```
            if(operator.isEmpty()){
                operand1 += arr[i];
            }else{
                operand2 += arr[i];
            }
        }

        if(arr[i] == '+' || arr[i] == '-' || arr[i] == '/' || arr[i] == '*') {
            operator += arr[i];
        }
    }

    if (operator.equals("+"))
        result = (Double.parseDouble(operand1) + Double.parseDouble(operand2));
    else if (operator.equals("-"))
        result = (Double.parseDouble(operand1) - Double.parseDouble(operand2));
    else if (operator.equals("/"))
        result = (Double.parseDouble(operand1) / Double.parseDouble(operand2));
    else
        result = (Double.parseDouble(operand1) * Double.parseDouble(operand2));
    return operand1 + operator + operand2 + "=" +result;
  }
}

/*
```

The provided Java code implements a basic calculator using Swing, a graphical user interface (GUI) toolkit for Java. Here's a paragraph explanation of the code:

The `SwingCalculator` class represents the main class that contains the `main` method. It also implements the `ActionListener` interface, which allows it to handle events triggered by user actions.

The program starts by creating a window using the `JFrame` class. The `createWindow` method is responsible for setting up the window with the appropriate title and size.

The `createUI` method sets up the user interface components within the window. It creates a `JPanel` to hold the calculator buttons and text field. It also creates an instance of `SwingCalculator` to serve as the event listener for the calculator buttons.

The method then creates all the necessary buttons (digits, operators, clear, dot, and equals) using the `JButton` class. Each button is assigned an action command and added to the panel.

The `GridBagLayout` is used to arrange the buttons in a grid-like manner using the `GridBagConstraints` class. The `gbc` object specifies the grid positions for each button, and the buttons are added to the panel accordingly.

The `inputBox` field, which is an instance of `JTextField`, is used to display the calculator input and output. It is set to non-editable to prevent user input directly into the text field.

The `actionPerformed` method handles button click events. It checks the action command of the clicked button and performs the appropriate action based on the command. If the command is "C", it clears the input text field. If the command is "=", it evaluates the expression entered in the text field using the `evaluate` method. Otherwise, it appends the clicked button's text to the input text field.

The `evaluate` method takes an expression as input and evaluates it. It iterates over the characters of the expression and extracts the operands and operator. It then performs the corresponding mathematical operation and returns the evaluated result as a string.

In summary, the code sets up a basic calculator GUI using Swing. It allows the user to enter numeric expressions and performs basic arithmetic operations.

*/

13.1 Write Java Program to find the transpose of a given matrix.

```
public class MatrixTransposeExample {
    public static void main(String args[]) {
        // Creating a matrix
        int original[][] = {{1, 3, 4}, {2, 4, 3}, {3, 4, 5}};

        // Creating another matrix to store transpose of a matrix
        int transpose[][] = new int[3][3]; // 3 rows and 3 columns

        // Code to transpose a matrix
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                transpose[i][j] = original[j][i];
            }
        }

        System.out.println("Printing Matrix without transpose:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(original[i][j] + " ");
            }
            System.out.println(); // new line
        }

        System.out.println("Printing Matrix After Transpose:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(transpose[i][j] + " ");
            }
            System.out.println(); // new line
        }
    }
}
```

/*

The provided Java code demonstrates how to transpose a matrix. Here's a paragraph explanation of the code:

The `MatrixTransposeExample` class contains the `main` method, which serves as the entry point of the program.

The program begins by creating an original matrix using a 2D integer array. The original matrix is a 3x3 matrix initialized with specific values.

Next, a new matrix called `transpose` is created using another 2D integer array. This matrix will store the transpose of the original matrix.

To calculate the transpose, a nested loop is used. The outer loop iterates over the rows of the original matrix, and the inner loop iterates over the columns. During each iteration, the value at position `(i, j)` in the original matrix is assigned to position `(j, i)` in the transpose matrix. This effectively swaps the rows and columns of the original matrix.

After the transpose operation, the program prints the original matrix and the transpose matrix to the console. The first loop prints the values of the original matrix row by row, while the second loop prints the values of the transpose matrix row by row.

In summary, the code creates a 3x3 matrix, performs the transpose operation by swapping rows and columns, and then displays both the original matrix and the transpose matrix on the console. The transpose of a matrix is obtained by interchanging its rows and columns.

In linear algebra, the transpose of a matrix is an operation that flips the matrix over its diagonal, reflecting its elements along the main diagonal. The transpose of an m x n matrix is an n x m matrix.

To obtain the transpose of a matrix, you swap its rows with columns. The element at row i, column j in the original matrix becomes the element at row j, column i in the transpose matrix. The transpose operation effectively changes the orientation of the matrix.

For example, consider the following matrix:

Original Matrix:

1 2 3

4 5 6

Transposed Matrix:

1 4

2 5

3 6

As you can see, the rows of the original matrix become the columns of the transposed matrix, and the columns of the original matrix become the rows of the transposed matrix.

The transpose operation is useful in various mathematical operations, such as solving systems of linear equations, matrix multiplication, and matrix transformations. It allows for easier manipulation and analysis of matrices in certain applications.

*/

13.2 Write Java Program to find the number of the words in the given text file.

```
import java.io.BufferedReader;
import java.io.FileReader;

public class CountWordFile {
    public static void main(String[] args) throws Exception {
        String line;
        int count = 0;

        // Opens a file in read mode
        FileReader file = new FileReader("C://Users//dell//Java Programs//Question 13//data.txt");
        BufferedReader br = new BufferedReader(file);

        // Gets each line till end of file is reached
        while ((line = br.readLine()) != null) {
            // Splits each line into words
            String words[] = line.split(" ");
            // Counts each word
            count = count + words.length;
        }
```

```
        System.out.println("Number of words present in given file: " + count);
        br.close();
    }
}

/*
```

The code you provided is a Java program that reads a file, counts the number of words in the file, and outputs the count.

Here's a paragraph explanation of the code:

1. The program starts by declaring variables `line` and `count`. `line` is used to store each line of the file, and `count` keeps track of the total word count.

2. A `FileReader` object named `file` is created to read the file. The file path specified is "C://Users//dell//Java Programs//Question 13//data.txt". This path should be modified according to the actual file path on your system.

3. A `BufferedReader` object named `br` is created, which takes the `FileReader` object `file` as an argument. `BufferedReader` is a class that reads text from a character-input stream with efficient buffering of characters, providing better performance compared to reading directly from a `FileReader`.

4. Inside the `while` loop, the program reads each line from the file using the `readLine()` method of `BufferedReader`. If there is no more line to read (reached the end of the file), the loop terminates.

5. For each line read, it is split into words using the `split()` method with a space (" ") as the delimiter. The resulting array of words is stored in the `words[]` array.

6. The length of the `words[]` array is added to the `count` variable to keep track of the total word count.

7. After the loop finishes, the program outputs the total number of words in the given file using `System.out.println()`.

8. Finally, the `BufferedReader` is closed using the `close()` method to release system resources.

Explanation of `BufferedReader` and `FileReader`:

- `FileReader` is a class in Java that is used to read characters from a file. It reads data from a file in a character-by-character manner.

- `BufferedReader` is a class that wraps around a `Reader` (in this case, a `FileReader`) to provide buffering capabilities, which improves the efficiency of reading characters from a stream. It reads characters from the input stream and stores them in an internal buffer, so that subsequent reads from the buffer are faster.

- In this code, `FileReader` is used to open the file, and `BufferedReader` is used to read the contents of the file line by line. The `readLine()` method of `BufferedReader` reads a line of text from the input stream and returns it as a `String`. This allows the program to process the file content line by line and perform word counting operations.

*/

12. Write a Java program to display the pattern like a diamond.
Input number of rows (half of the diamond) :7 Expected Output :

```
*
***
*****
*******
*********
***********
*************
***********
*********
*******
*****
***
*
```

```
import java.util.Scanner;


public class DiamondPattern {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);


        System.out.print("Input number of rows (half of the diamond): ");
        int numRows = scanner.nextInt();


        // Print upper half of the diamond
        for (int i = 1; i <= numRows; i++) {
            for (int k = 1; k <= 2 * i - 1; k++) {
                System.out.print("*");
            }
            System.out.println();
        }


        // Print lower half of the diamond
        for (int i = numRows - 1; i >= 1; i--) {
            for (int k = 1; k <= 2 * i - 1; k++) {
                System.out.print("*");
            }
```

```
            System.out.println();
        }

        scanner.close();
    }
}
```

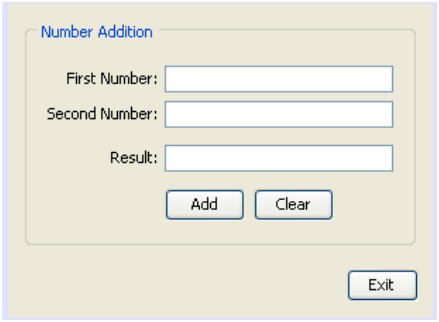
11.1 Write a Java program that takes a number as input and prints its multiplication table up to 10. Test Data:
Input a number: 8
Expected Output:
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
...
8 x 10 = 80

```
import java.util.Scanner;

public class MultiplicationTable {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Input a number: ");
        int number = scanner.nextInt();

        System.out.println("Multiplication table of " + number + ":");
        for (int i = 1; i <= 10; i++) {
            int result = number * i;
            System.out.println(number + " x " + i + " = " + result);
        }

        scanner.close();
    }
}
```

11.2 Write a java program to check that given number is prime or not.

```
import java.util.Scanner;

public class PrimeNumberChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        boolean isPrime = checkPrime(number);

        if (isPrime) {
            System.out.println(number + " is a prime number.");
        } else {
            System.out.println(number + " is not a prime number.");
        }

        scanner.close();
    }

    public static boolean checkPrime(int number) {
        if (number <= 1) {
            return false;
        }

        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) {
                return false;
            }
        }

        return true;
    }
}
```

10. 


```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class NumberAdditionGUI {
    private JFrame frame;
    private JLabel titleLabel, firstNumLabel, secondNumLabel, resultLabel;
    private JTextField firstNumTextField, secondNumTextField, resultTextField;
    private JButton addButton, clearButton, exitButton;

    public NumberAdditionGUI() {
        createAndShowGUI();
    }

    private void createAndShowGUI() {
        frame = new JFrame("Number Addition");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        // Top panel for the title label
        JPanel topPanel = new JPanel();
        topPanel.setBackground(Color.BLUE);
        titleLabel = new JLabel("Number Addition");
        titleLabel.setForeground(Color.WHITE);
        topPanel.add(titleLabel);
```

```
// Center panel for the input fields
JPanel centerPanel = new JPanel(new GridLayout(3, 2, 10, 10));
firstNumLabel = new JLabel("First Number:");
firstNumTextField = new JTextField();
secondNumLabel = new JLabel("Second Number:");
secondNumTextField = new JTextField();
resultLabel = new JLabel("Result:");
resultTextField = new JTextField();
resultTextField.setEditable(false);
centerPanel.add(firstNumLabel);
centerPanel.add(firstNumTextField);
centerPanel.add(secondNumLabel);
centerPanel.add(secondNumTextField);
centerPanel.add(resultLabel);
centerPanel.add(resultTextField);

// Bottom panel for the buttons
JPanel bottomPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));
addButton = new JButton("Add");
clearButton = new JButton("Clear");
exitButton = new JButton("Exit");
bottomPanel.add(addButton);
bottomPanel.add(clearButton);
bottomPanel.add(exitButton);

// Add action listeners to buttons
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        performAddition();
    }
});

clearButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
```

```
            clearFields();
        }
    });

    exitButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    });

    // Add panels to the frame
    frame.add(topPanel, BorderLayout.NORTH);
    frame.add(centerPanel, BorderLayout.CENTER);
    frame.add(bottomPanel, BorderLayout.SOUTH);

    frame.pack();
    frame.setVisible(true);
}

private void performAddition() {
    try {
        int firstNum = Integer.parseInt(firstNumTextField.getText());
        int secondNum = Integer.parseInt(secondNumTextField.getText());
        int result = firstNum + secondNum;
        resultTextField.setText(String.valueOf(result));
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(frame, "Invalid input. Please enter valid numbers.");
    }
}

private void clearFields() {
    firstNumTextField.setText("");
    secondNumTextField.setText("");
    resultTextField.setText("");
```

```
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new NumberAdditionGUI();
            }
        });
    }
}

/*
```

The code you provided is a Java program that creates a simple GUI application for adding two numbers. It uses Swing components to create the graphical interface.

Here's a paragraph explanation of the code:

1. The `NumberAdditionGUI` class is defined. It has instance variables representing the GUI components such as `JFrame`, `JLabel`, `JTextField`, and `JButton`.

2. The constructor `NumberAdditionGUI()` is called to create and show the GUI.

3. The `createAndShowGUI()` method is defined. It sets up the main frame, layout, panels, labels, text fields, and buttons.

4. Inside the `createAndShowGUI()` method, various Swing components are created and added to panels using different layout managers (`BorderLayout`, `FlowLayout`, `GridLayout`).

5. Action listeners are added to the buttons (`addButton`, `clearButton`, `exitButton`) using anonymous inner classes implementing the `ActionListener` interface. These listeners define the actions to be performed when the buttons are clicked.

6. The `performAddition()` method is called when the "Add" button is clicked. It retrieves the numbers entered in the text fields, performs the addition, and displays the result in the `resultTextField`. If invalid input is detected (e.g., non-numeric values), a message dialog is shown.

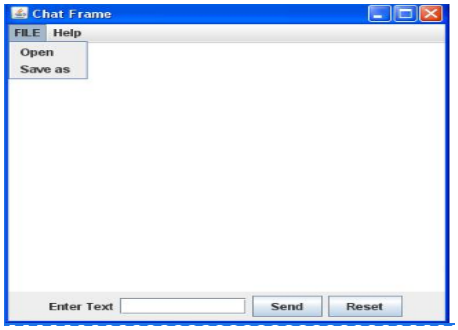
7. The `clearFields()` method is called when the "Clear" button is clicked. It clears the input text fields and the result field.

8. The `main()` method is the entry point of the program. It schedules the creation of the GUI on the event dispatch thread using `SwingUtilities.invokeLater()`.

9. When the program runs, the GUI is displayed with the specified components and their respective functionalities.

The code provides a basic example of creating a GUI application using Swing components and handling user interactions through action listeners.

*/

9. 


```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class GUI {
    public static void main(String[] args) {

        // Creating the Frame
        JFrame frame = new JFrame("Chat Frame");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);

        // Creating the MenuBar and adding components
        JMenuBar mb = new JMenuBar();
        JMenu m1 = new JMenu("FILE");
        JMenu m2 = new JMenu("Help");
        mb.add(m1);
        mb.add(m2);
        JMenuItem m11 = new JMenuItem("Open");
        JMenuItem m22 = new JMenuItem("Save as");
        m1.add(m11);
        m1.add(m22);

        // Creating the panel at bottom and adding components
        JPanel panel = new JPanel(); // the panel is not visible in output
        JLabel label = new JLabel("Enter Text");
```

```
        JTextField tf = new JTextField(10); // accepts up to 10 characters
        JButton send = new JButton("Send");
        JButton reset = new JButton("Reset");
        panel.add(label); // Components Added using Flow Layout
        panel.add(tf);
        panel.add(send);
        panel.add(reset);

        // Text Area at the Center
        JTextArea ta = new JTextArea();

        // Action Listener for Send button
        send.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String text = tf.getText();
                ta.append(text + "\n");
                tf.setText(""); // Clear the text field after appending to the text area
            }
        });

        // Action Listener for Reset button
        reset.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ta.setText(""); // Clear the text area
            }
        });

        // Adding Components to the frame
        frame.getContentPane().add(BorderLayout.SOUTH, panel);
        frame.getContentPane().add(BorderLayout.NORTH, mb);
        frame.getContentPane().add(BorderLayout.CENTER, ta);
        frame.setVisible(true);
    }
}
```

/*

The code you provided creates a basic chat application GUI using Swing components. Here's an explanation of the code:

1. The `GUI` class is defined, which contains the `main()` method. This is the entry point of the program.

2. Inside the `main()` method, a `JFrame` named "Chat Frame" is created. It sets the title, default close operation, and size of the frame.

3. A `JMenuBar` named `mb` is created. It will contain two menus: "FILE" and "Help".

4. Two `JMenu` objects, `m1` and `m2`, are created for the "FILE" and "Help" menus, respectively. These menus are added to the menu bar.

5. `JMenuItem` objects, `m11` and `m22`, are created for the "Open" and "Save as" options, respectively. These menu items are added to the "FILE" menu.

6. A `JPanel` named `panel` is created at the bottom of the frame. It contains a `JLabel` ("Enter Text"), a `JTextField` (`tf`) for user input, and two `JButton` objects (`send` and `reset`). The panel uses the default `FlowLayout` for arranging components.

7. A `JTextArea` named `ta` is created to display the chat messages. It will be placed at the center of the frame.

8. An `ActionListener` is added to the `send` button. When the button is clicked, the listener appends the text from the `tf` text field to the `ta` text area. It also clears the `tf` text field after appending.

9. Another `ActionListener` is added to the `reset` button. When the button is clicked, the listener clears the `ta` text area.

10. Finally, the components (`panel`, `mb`, `ta`) are added to the frame's content pane using the `BorderLayout`. The panel is placed at the south, the menu bar at the north, and the text area at the center. The frame is then set to visible to display the GUI.

In summary, the code creates a simple chat interface with a text area for displaying messages, a text field for entering text, and buttons for sending and resetting the chat. The `JMenuBar` provides a basic menu structure. */

8. You are given a phone book that consists of people's names and their phone number. After that you will be given some person's name as query. For each query, print the phone number of that person. Use HashMap to implement it.The first line will have an integer denoting the number of entries in the phone book. Each entry consists of two lines: a name and the corresponding phone number.
After these, there will be some queries. Each query will contain a person's name. Read the queries until end-of-file.
Constraints:
A person's name consists of only lower-case English letters and it may be in the format 'first-name last-name' or in the format 'first-name'. Each phone number has exactly 8 digits without any leading zeros.For each case, print "Not found" if the person has no entry in the phone book. Otherwise, print the person's name and phone number.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class GUI {
   public static void main(String[] args) {

      // Creating the Frame
      JFrame frame = new JFrame("Chat Frame");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(400, 400);

      // Creating the MenuBar and adding components
      JMenuBar mb = new JMenuBar();
      JMenu m1 = new JMenu("FILE");
      JMenu m2 = new JMenu("Help");
      mb.add(m1);
      mb.add(m2);
      JMenuItem m11 = new JMenuItem("Open");
      JMenuItem m22 = new JMenuItem("Save as");
      m1.add(m11);
      m1.add(m22);

      // Creating the panel at bottom and adding components
```

```
JPanel panel = new JPanel(); // the panel is not visible in output
JLabel label = new JLabel("Enter Text");
JTextField tf = new JTextField(10); // accepts up to 10 characters
JButton send = new JButton("Send");
JButton reset = new JButton("Reset");
panel.add(label); // Components Added using Flow Layout
panel.add(tf);
panel.add(send);
panel.add(reset);

// Text Area at the Center
JTextArea ta = new JTextArea();

// Action Listener for Send button
send.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String text = tf.getText();
        ta.append(text + "\n");
        tf.setText(""); // Clear the text field after appending to the text area
    }
});

// Action Listener for Reset button
reset.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ta.setText(""); // Clear the text area
    }
});

// Adding Components to the frame
frame.getContentPane().add(BorderLayout.SOUTH, panel);
frame.getContentPane().add(BorderLayout.NORTH, mb);
frame.getContentPane().add(BorderLayout.CENTER, ta);
frame.setVisible(true);
```

```
    }
}
```

/*

The code you provided creates a basic chat application GUI using Swing components. Here's an explanation of the code:

1. The `GUI` class is defined, which contains the `main()` method. This is the entry point of the program.

2. Inside the `main()` method, a `JFrame` named "Chat Frame" is created. It sets the title, default close operation, and size of the frame.

3. A `JMenuBar` named `mb` is created. It will contain two menus: "FILE" and "Help".

4. Two `JMenu` objects, `m1` and `m2`, are created for the "FILE" and "Help" menus, respectively. These menus are added to the menu bar.

5. `JMenuItem` objects, `m11` and `m22`, are created for the "Open" and "Save as" options, respectively. These menu items are added to the "FILE" menu.

6. A `JPanel` named `panel` is created at the bottom of the frame. It contains a `JLabel` ("Enter Text"), a `JTextField` (`tf`) for user input, and two `JButton` objects (`send` and `reset`). The panel uses the default `FlowLayout` for arranging components.

7. A `JTextArea` named `ta` is created to display the chat messages. It will be placed at the center of the frame.

8. An `ActionListener` is added to the `send` button. When the button is clicked, the listener appends the text from the `tf` text field to the `ta` text area. It also clears the `tf` text field after appending.

9. Another `ActionListener` is added to the `reset` button. When the button is clicked, the listener clears the `ta` text area.

10. Finally, the components (`panel`, `mb`, `ta`) are added to the frame's content pane using the `BorderLayout`. The panel is placed at the south, the menu bar at the north, and the text area at the center. The frame is then set to visible to display the GUI.

In summary, the code creates a simple chat interface with a text area for displaying messages, a text field for entering text, and buttons for sending and resetting the chat. The `JMenuBar` provides a basic menu structure.

*/

7. You are required to compute the power of a number by implementing a calculator. Create a class My Calculator which consists of a single method long power (int, int). This method takes two integers n and p, as parameters and finds (n)p. If either or is negative, then the method must throw an exception which says " n or p should not be negative". Also, if both and are zero, then the method must throw an exception which says "n or p should not be negative".

```
class MyCalculator {
    public long power(int n, int p) throws Exception {
        if (n < 0 || p < 0) {
            throw new Exception("n or p should not be negative");
        } else if (n == 0 && p == 0) {
            throw new Exception("n and p should not be zero");
        } else {
            return (long) Math.pow(n, p);
        }
    }
}

public class Main01 {
    public static void main(String[] args) {
        MyCalculator calculator = new MyCalculator();

        try {
            long result1 = calculator.power(2, 3);
            System.out.println("Result 1: " + result1);

            long result2 = calculator.power(-2, 3); // This will throw an exception
            System.out.println("Result 2: " + result2);

            long result3 = calculator.power(2, -3); // This will throw an exception
            System.out.println("Result 3: " + result3);

            long result4 = calculator.power(0, 0); // This will throw an exception
            System.out.println("Result 4: " + result4);
        } catch (Exception e) {
```

```
            System.out.println(e.getMessage());
        }
    }
}
```

/*

The code you provided demonstrates the use of a `MyCalculator` class to calculate the power of a number. Here's an explanation of the code:

1. The `MyCalculator` class is defined, which contains a `power` method. This method takes two integer parameters `n` and `p` representing the base and exponent, respectively, and returns a `long` value.

2. Inside the `power` method, there is exception handling logic. If either `n` or `p` is negative, an exception is thrown with the message "n or p should not be negative". If both `n` and `p` are zero, an exception is thrown with the message "n and p should not be zero". Otherwise, the power of `n` raised to `p` is calculated using the `Math.pow` method, and the result is cast to a `long` value before returning.

3. The `Main01` class is defined, which contains the `main` method. This is the entry point of the program.

4. Inside the `main` method, an instance of `MyCalculator` named `calculator` is created.

5. The `power` method of `calculator` is called multiple times, each with different arguments.

6. The first call, `calculator.power(2, 3)`, calculates 2 raised to the power of 3, which is 8. The result is stored in `result1` and printed.

7. The second call, `calculator.power(-2, 3)`, passes a negative value for `n`. This will throw an exception, and the corresponding error message is printed.

8. The third call, `calculator.power(2, -3)`, passes a negative value for `p`. This will also throw an exception, and the error message is printed.

9. The fourth call, `calculator.power(0, 0)`, passes zero values for both `n` and `p`. This violates the condition in the `power` method and throws an exception. The error message is printed.

10. The exceptions thrown in the above cases are caught in a `catch` block, and the error messages are printed using the `getMessage` method of the caught exception.

In summary, the code demonstrates how to handle exceptions when calculating the power of a number. It shows how to catch and handle exceptions thrown by the `power` method and print appropriate error messages.

*/

6. Write the following code in your editor below:
A class named Arithmetic with a method named add that takes 2 integers as parameters and returns an integer denoting their sum.
A class named Adder that inherits from a superclass named Arithmetic. The main method in the Tester class should print the following: SAMPLE O/P:**My superclass is: Arithmetic**
42 13 20

```
class Arithmetic {
   Arithmetic()
    {
        System.out.println("Arithmetic");
    }
    public int add(int a, int b) {
       return a + b;
   }
}


class Adder extends Arithmetic {
   Adder(){
       super();
   }
}


class Main02 {
   public static void main(String[] args) {
       System.out.print("My superclass is: " );
       Adder adder = new Adder();
       System.out.println(adder.add(42, 13));
       System.out.println(adder.add(20, 0));
   }
}


/*
```

The code you provided demonstrates inheritance and method overriding in Java. Here's an explanation of the code:

1. The `Arithmetic` class is defined. It has a default constructor that prints "Arithmetic" when an instance of the class is created. It also has a public method `add` that takes two integers `a` and `b` as parameters and returns their sum.

2. The `Adder` class extends the `Arithmetic` class. It has a default constructor that invokes the constructor of the superclass `Arithmetic` using the `super()` statement.

3. The `Main02` class is defined, which contains the `main` method. This is the entry point of the program.

4. Inside the `main` method, the statement `System.out.print("My superclass is: ");` is used to print the text "My superclass is: " without a newline character.

5. An instance of the `Adder` class named `adder` is created. This invokes the default constructor of the `Adder` class, which, in turn, invokes the constructor of the superclass `Arithmetic` due to the `super()` statement in the `Adder` constructor. As a result, "Arithmetic" is printed.

6. The statement `System.out.println(adder.add(42, 13));` invokes the `add` method of the `Adder` class with arguments 42 and 13. Since the `Adder` class extends the `Arithmetic` class, it inherits the `add` method. The `add` method adds the two numbers and returns the sum, which is then printed.

7. The statement `System.out.println(adder.add(20, 0));` invokes the `add` method of the `Adder` class with arguments 20 and 0. The `add` method of the `Adder` class overrides the `add` method of the `Arithmetic` class. In this case, the overridden method is called, and it performs the addition and returns the sum, which is then printed.

In summary, the code demonstrates inheritance and method overriding. The `Adder` class inherits the `add` method from the `Arithmetic` class and overrides it. The `Main02` class creates an instance of the `Adder` class, invokes the `add` method, and prints the results. Additionally, it shows the use of the `super()` statement to invoke the constructor of the superclass.

*/

5. We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B. Create an abstract class 'Marks' with an abstract method 'getPercentage'. It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students. The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B. Create an object for eac of the two classes and print the percentage of marks for both the students.

```
import java.util.Scanner;

abstract class Marks {
   public abstract float getPercentage();
}

class A extends Marks {
   private float marks1, marks2, marks3;

   public A(float m1, float m2, float m3) {
      marks1 = m1;
      marks2 = m2;
      marks3 = m3;
   }

   public float getPercentage() {
      return ((marks1 + marks2 + marks3) / 300) * 100;
   }
}

class B extends Marks {
   private float marks1, marks2, marks3, marks4;

   public B(float m1, float m2, float m3, float m4) {
      marks1 = m1;
      marks2 = m2;
      marks3 = m3;
      marks4 = m4;
   }
```

```
    public float getPercentage() {
        return ((marks1 + marks2 + marks3 + marks4) / 400) * 100;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter marks for student A (m1, m2, m3):");
        float m1 = scanner.nextFloat();
        float m2 = scanner.nextFloat();
        float m3 = scanner.nextFloat();
        A studentA = new A(m1, m2, m3);

        System.out.println("Enter marks for student B (m1, m2, m3, m4):");
        float m4 = scanner.nextFloat();
        B studentB = new B(m1, m2, m3, m4);

        System.out.println("Percentage of marks for student A: " + studentA.getPercentage() + "%");
        System.out.println("Percentage of marks for student B: " + studentB.getPercentage() + "%");

        scanner.close();
    }
}
```

4. A Company manufactures Vehicles, which could be a Helicopter, a Car, or a Train depending on the customer's demand. Each Vehicle instance has a method called move, which prints on the console the nature of movement of the vehicle. For example, the Helicopter Flies in Air, the Car Drives on Road and the Train Runs on Track. Write a program that accepts input from the user on the kind of vehicle the user wants to order, and the system should print out nature of movement. Implement all Java coding best practices to implement this program.

```
import java.util.Scanner;

abstract class Vehicle {
    public abstract void move();
}

class Helicopter extends Vehicle {
    @Override
    public void move() {
        System.out.println("The Helicopter flies in the air.");
    }
}

class Car extends Vehicle {
    @Override
    public void move() {
        System.out.println("The Car drives on the road.");
    }
}

class Train extends Vehicle {
    @Override
    public void move() {
        System.out.println("The Train runs on the track.");
    }
}

public class VehicleDemo {
    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);

        System.out.println("Welcome to the Vehicle Manufacturer!");
        System.out.println("Please enter the type of vehicle you want to order (Helicopter/Car/Train):");
        String vehicleType = scanner.nextLine();

        Vehicle vehicle;

        if (vehicleType.equalsIgnoreCase("Helicopter")) {
            vehicle = new Helicopter();
        } else if (vehicleType.equalsIgnoreCase("Car")) {
            vehicle = new Car();
        } else if (vehicleType.equalsIgnoreCase("Train")) {
            vehicle = new Train();
        } else {
            System.out.println("Invalid vehicle type. Program exiting.");
            scanner.close();
            return;
        }

        System.out.println("Nature of movement of the vehicle:");
        vehicle.move();

        scanner.close();
    }
}

/*
```

The code you provided demonstrates the use of abstract classes and polymorphism in Java. Here's an explanation of the code:

1. The `Vehicle` class is defined as an abstract class. It has an abstract method `move()` that does not have an implementation. This means that any class extending `Vehicle` must provide its own implementation of the `move()` method.

2. The `Helicopter`, `Car`, and `Train` classes are defined as subclasses of `Vehicle`. Each of these classes overrides the `move()` method inherited from the `Vehicle` class and provides its own implementation of how the specific vehicle moves.

3. The `VehicleDemo` class contains the `main` method, which is the entry point of the program.

4. Inside the `main` method, a `Scanner` object is created to read input from the user.

5. The program prompts the user to enter the type of vehicle they want to order (Helicopter/Car/Train) using the `System.out.println` statement. The user's input is then stored in the `vehicleType` variable using the `nextLine()` method of the `Scanner` object.

6. A `Vehicle` object named `vehicle` is declared but not yet instantiated.

7. Using a series of conditional statements (`if`, `else if`), the program checks the value of the `vehicleType` variable. If it matches "Helicopter", "Car", or "Train" (ignoring case), a corresponding object of the respective class (`Helicopter`, `Car`, or `Train`) is created and assigned to the `vehicle` variable.

8. If the `vehicleType` does not match any of the valid options, an error message is displayed, and the program exits gracefully by closing the `Scanner` object and returning from the `main` method.

9. After determining the type of vehicle and creating the corresponding object, the program displays the message "Nature of movement of the vehicle:" using the `System.out.println` statement.

10. The `move()` method is invoked on the `vehicle` object. Since the actual object type can be `Helicopter`, `Car`, or `Train`, the appropriate implementation of the `move()` method from the corresponding class is called due to polymorphism.

11. The program then closes the `Scanner` object using the `close()` method.

12. The execution of the program ends.

Explanation of `equalsIgnoreCase`:

The `equalsIgnoreCase` method is a Java String method used to compare two strings while ignoring their case (uppercase or lowercase). It returns `true` if the two strings are equal, ignoring case, and returns `false` otherwise. In the provided code, it is used to compare the user's input (`vehicleType`) with the valid vehicle types ("Helicopter", "Car", and "Train") without considering the case. This allows the program to handle user input in a case-insensitive manner, making it more user-friendly.*/

3.1 Java Program to Reverse a String.

```
import java.util.*;

public class StringReverseExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String originalString = scanner.nextLine();
        String reversedString = reverseString(originalString);
        System.out.println("Original String: " + originalString);
        System.out.println("Reversed String: " + reversedString);
        scanner.close();
    }

    public static String reverseString(String str) {
        StringBuilder reversed = new StringBuilder();
        for (int i = str.length() - 1; i >= 0; i--) {
            reversed.append(str.charAt(i));
        }
        return reversed.toString();
    }
}
```

In short, `StringBuilder` is a class in Java that allows you to modify strings efficiently. It provides methods for appending, inserting, deleting, and replacing characters within a string. Unlike the `String` class, which is immutable, `StringBuilder` is mutable, meaning you can change its contents without creating a new object each time. It is useful when you need to perform multiple operations on a string and want to avoid the overhead of creating new strings.

3.2 Write a Java program to check that String is palindrome or not.

```
import java.util.*;

public class PalindromeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String str = scanner.nextLine();
        if (isPalindrome(str)) {
```

```
            System.out.println(str + " is a palindrome.");
        } else {
            System.out.println(str + " is not a palindrome.");
        }
        scanner.close();
    }

    public static boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }

        return true;
    }
}
```

2.1 Java Program to Count Number of Duplicate Words in String

```
import java.util.*;

public class CountDuplicateWords {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the string: ");
        String string = scanner.nextLine();
        int count;

        //Converts the string into lowercase
        string = string.toLowerCase();

        //Split the string into words using built-in function
        String words[] = string.split(" ");

        System.out.println("Duplicate words in a given string : ");
        for(int i = 0; i < words.length; i++) {
            count = 1;
            for(int j = i+1; j < words.length; j++) {
                if(words[i].equals(words[j])) {
                    count++;
                    //Set words[j] to 0 to avoid printing visited word
                    words[j] = "0";
                }
            }

            //Displays the duplicate word if count is greater than 1
            if(count > 1 && words[i] != "0")
                System.out.println(words[i]);

        }
        scanner.close();
    }
}
```

2.2 How to Check if the String Contains 'e' in umbrella

```
public class StringContainsExample {
    public static void main(String[] args) {
        String word = "umbrella";
        if (word.contains("e")) {
            System.out.println("The string contains 'e'.");
        } else {
            System.out.println("The string does not contain 'e'.");
        }
    }
}
```

1.1) Program to remove all repeated elements from an array

```
import java.util.Scanner;

public class RemoveDuplicatesFromArray {
        public static void main(String[] args) {
                Scanner scanner = new Scanner(System.in);
                System.out.print("Enter the number of elements in the array: ");
                int size = scanner.nextInt();
    if(size == 0) {
       System.out.println("0 elements present in the array.");
    } else if(size == 1) {
       int[] array = new int[size];
       System.out.println("Enter the elements of the array:");

       for (int i = 0; i < size; i++) {
          array[i] = scanner.nextInt();
       }
       System.out.println("No duplicates are present in the array.");
    } else if(size < 0) {
       System.out.println("Please enter positive size.");
    } else{
       int[] array = new int[size];
       System.out.println("Enter the elements of the array:");

       for (int i = 0; i < size; i++) {
          array[i] = scanner.nextInt();
       }

       int[] uniqueArray = removeDuplicates(array);

       System.out.println("Array with duplicates removed:");
       for (int num : uniqueArray) {
          System.out.print(num + " ");
```

```
        }
    }
    scanner.close();
  }

  public static int[] removeDuplicates(int[] array) {
    int length = array.length;
    int[] uniqueArray = new int[length];
    int uniqueCount = 0;

    for (int i = 0; i < length; i++) {
      boolean isDuplicate = false;
      for (int j = 0; j < uniqueCount; j++) {
        if (array[i] == uniqueArray[j]) {
          isDuplicate = true;
          break;
        }
      }

      if (!isDuplicate) {
        uniqueArray[uniqueCount] = array[i];
        uniqueCount++;
      }
    }

    int[] resultArray = new int[uniqueCount];
    for (int i = 0; i < uniqueCount; i++) {
      resultArray[i] = uniqueArray[i];
    }

    return resultArray;
  }
}
```

1.2) Write a Java program to find the common elements between two arrays of integers.

```
import java.util.Scanner;

public class CommonElementsBetweenArrays {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the first array: ");
        int size1 = scanner.nextInt();

        System.out.print("Enter the number of elements in the second array: ");
        int size2 = scanner.nextInt();

        int[] array1 = new int[size1];
        int[] array2 = new int[size2];

        System.out.println("Enter the elements of the first array:");
        for (int i = 0; i < size1; i++) {
            array1[i] = scanner.nextInt();
        }

        System.out.println("Enter the elements of the second array:");
        for (int i = 0; i < size2; i++) {
            array2[i] = scanner.nextInt();
        }

        int[] commonElements = findCommonElements(array1, array2);

        System.out.println("Common elements between the two arrays:");
        for (int num : commonElements) {
            System.out.print(num + " ");
        }

        scanner.close();
    }
```

```
    public static int[] findCommonElements(int[] array1, int[] array2) {
        int length1 = array1.length;
        int length2 = array2.length;
        int[] commonElements = new int[Math.min(length1, length2)];
        int count = 0;

        for (int i = 0; i < length1; i++) {
            for (int j = 0; j < length2; j++) {
                if (array1[i] == array2[j]) {
                    commonElements[count] = array1[i];
                    count++;
                    break;
                }
            }
        }

        int[] resultArray = new int[count];
        for (int i = 0; i < count; i++) {
            resultArray[i] = commonElements[i];
        }

        return resultArray;
    }
}
```

18. Write a program to implement following inheritance. Accept data for 5 persons and display the name of employee having salary greater than 5000.

Class Name: Person
Member variables:
Name, age

Class Name: Employee
Member variables:
Designation, salary

```
//Write a program to implement following inheritance. Accept data for 5 persons and display the name of employee having salary greater than 5000.


// Class Name: Person

// Member variables:

// Name, age


// Class Name: Employee

// Member variables:

// Designation, salary




import java.util.Scanner;


class Person {

   private String name;

   private int age;


   public Person(String name, int age) {

      this.name = name;

      this.age = age;

   }


   public String getName() {

      return name;

   }


   public int getAge() {
```

```
        return age;
    }
}

class Employee extends Person {
    private String designation;
    private double salary;

    public Employee(String name, int age, String designation, double salary) {
        super(name, age);
        this.designation = designation;
        this.salary = salary;
    }

    public String getDesignation() {
        return designation;
    }

    public double getSalary() {
        return salary;
    }
}

public class EmployeeSalary {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Employee[] employees = new Employee[5];

        // Accept data for 5 persons
        for (int i = 0; i < 5; i++) {
            System.out.println("Enter details for Employee " + (i + 1));
            System.out.print("Name: ");
            String name = scanner.nextLine();
            System.out.print("Age: ");
```

```
            int age = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Designation: ");
            String designation = scanner.nextLine();
            System.out.print("Salary: ");
            double salary = scanner.nextDouble();
            scanner.nextLine();

            employees[i] = new Employee(name, age, designation, salary);
        }

        // Display the name of employees with salary greater than 5000
        System.out.println("Employees with salary greater than 5000:");
        for (Employee employee : employees) {
            if (employee.getSalary() > 5000) {
                System.out.println(employee.getName());
            }
        }
    }
}
```

19. Implementing “Multiple Inheritance”. Create a two interfaces Account containing methods set() and display() And interface Person containing methods store() and disp(). Derive a class Customer from Person and Account. Accept the name, account number, balance and display all the information related to account along with the interest.

```
interface Account {
    void set(String accountNumber, double balance);

    void display();
}

interface Person {
    void store(String name);

    void disp();
}

class Customer implements Account, Person {
    private String name;
    private String accountNumber;
    private double balance;

    public void set(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    public void display() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Balance: " + balance);
    }

    public void store(String name) {
        this.name = name;
    }

    public void disp() {
        System.out.println("Name: " + name);
    }
```

```
    public void calculateInterest() {
        double interest = balance * 0.05; // Assuming interest rate of 5%
        System.out.println("Interest: " + interest);
    }
}

public class InterestCalci {
    public static void main(String[] args) {
        Customer customer = new Customer();
        customer.store("John Doe");
        customer.set("123456789", 10000.0);
        customer.disp();
        customer.display();
        customer.calculateInterest();
    }
}

/*
```

Sure! Here's a line-by-line explanation of the code:

1. The code defines two interfaces: `Account` and `Person`. These interfaces declare certain methods that classes implementing these interfaces must implement.

4. The `Customer` class implements both the `Account` and `Person` interfaces. It provides implementations for the methods declared in these interfaces.

7-11. The `set` method in the `Customer` class is an implementation of the `set` method from the `Account` interface. It takes an `accountNumber` and `balance` as parameters and sets the corresponding instance variables.

14-18. The `display` method in the `Customer` class is an implementation of the `display` method from the `Account` interface. It prints the `accountNumber` and `balance` to the console.

21-24. The `store` method in the `Customer` class is an implementation of the `store` method from the `Person` interface. It takes a `name` as a parameter and sets the corresponding instance variable.

27-30. The `disp` method in the `Customer` class is an implementation of the `disp` method from the `Person` interface. It prints the `name` to the console.

33-37. The `calculateInterest` method in the `Customer` class is a custom method. It calculates the interest based on the balance (assuming an interest rate of 5%) and prints it to the console.

40-45. The `InterestCalci` class contains the `main` method where the program starts execution.

47. An instance of the `Customer` class is created.

49. The `store` method is called to store the customer's name as "John Doe".

51. The `set` method is called to set the account number as "123456789" and balance as 10000.0.

53. The `disp` method is called to display the customer's name.

55. The `display` method is called to display the account number and balance.

57. The `calculateInterest` method is called to calculate and display the interest.

Each method call in the `main` method demonstrates the functionality of the implemented interfaces and the custom method in the `Customer` class.

*/