Abstraction

1. Create an abstract class Shape
2. The Shape class has two abstract methods calculateArea() and calculatePerimeter. Both the methods have a return type of void
3. Create a class Quadrilateral which extends the abstract class Shape.
4. Implement all the abstract method of the parent class

```java

abstract class Shape {
    abstract void calculateArea();
    abstract void calculatePerimeter();
}

class Quadrilateral extends Shape{

    @Override
    void calculateArea() {
        // TODO Auto-generated method stub
        System.out.println("Area");
    }

    @Override
    void calculatePerimeter() {
        // TODO Auto-generated method stub
        System.out.println("Perimeter");
    }

}
```

```java
public class ShapeTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Shape s= new Quadrilateral();
        s.calculateArea();
        s.calculatePerimeter();
    }

}
```

Console × Problems Debug Shell

```
<terminated> ShapeTest [Java Application] C:\Program Files\Java\jdk-19\bin\jav
Area
Perimeter
```

5. Create an abstract class named Vehicle which consist of two methods: wheel and door. Both the methods have void return type and no parameters. The method wheel has no implementation.

6. Create a class name Bus and extend the Vehicle class.

```java
abstract class Vehicle {
    void wheel() {
        System.out.println("This is wheel");
    }

    abstract void door();
}

class Bus extends Vehicle{

    @Override
    void door() {
        // TODO Auto-generated method stub
        System.out.println("This is door.");
    }

}
```

```java

public class VehicleTest {
    public static void main(String[] args) {
        Vehicle v= new Bus();
        v.door();
        v.wheel();
    }
}

```

Console × Problems Debug Shell

```
<terminated> VehicleTest [Java Application] C:\Program Files\Java\jdk-19\bin\ja
This is door.
This is wheel
```

Interface

7. Create an interface Animal. The Animal interface has two methods eat() and walk()
8. Create another interface Printable. The Printable interface has a method called display();

9. Create a class Cow that implements the Animal and Printable interfaces

```java

interface Animal {
    void eat();
    void walk();
}

interface printable{
    void display();
}

class Cow implements Animal,printable{

    @Override
    public void display() {
        // TODO Auto-generated method stub
        System.out.println("This is display section.");
    }

    @Override
    public void eat() {
        // TODO Auto-generated method stub
        System.out.println("Cow can eat.");
    }

    @Override
    public void walk() {
        // TODO Auto-generated method stub
        System.out.println("Cow can walk.");
    }

}


```

```java

public class AnimalTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Animal a=new Cow();
        printable p= new Cow();
        a.eat();
        a.walk();
        p.display();
    }

}

```

Console × Problems Debug Shell

```
<terminated> AnimalTest [Java Application] C:\Program Files\Java\jdk-19\bin\jav
Cow can eat.
Cow can walk.
This is display section.
```

10. Create an interface LivingBeing
11. Create an method void specialFeature()

Classes

12. Create 2 classes Fish and Bird that implements LivingBeing
13. The specialFeature should display special features of the respective class animal.

```java

interface LivingBeing {
    void specialFeature();
}

class Fish implements LivingBeing{

    @Override
    public void specialFeature() {
        // TODO Auto-generated method stub
        System.out.println("FIsh swims");
    }

}

class Bird implements LivingBeing{

    @Override
    public void specialFeature() {
        // TODO Auto-generated method stub
        System.out.println("Bird Flies");
    }
}
```

```java

public class LivingBeingTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        LivingBeing l=new Fish();
        LivingBeing b= new Bird();
        l.specialFeature();
        b.specialFeature();
    }

}
```

Console × Problems Debug Shell

```
<terminated> LivingBeingTest [Java Application] C:\Program Files\Java\jdk-19\
FIsh swims
Bird Flies
```

Exception

14. In the following program, which exception will be generated

```
public class Demo{
        public static void main(String[] args) {
  System.out.println(10/0);
        }
}
```

Handle the exception above by using try-catch.

```
public class NewExcep {
    public static void main(String[] args) {
        try {
            System.out.println(10/0);
        }
        catch(ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed."
        }

    }
}
```

```
Console × Problems Debug Shell
<terminated> NewExcep [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 28, 2023, 1:06:30 PM
Error: Division by zero is not allowed.
```

15. In the following program, which exception will be generated

```
public class Demo{
        public static void main(String[] args) {
  int[] age = {10,20,25,24,28,27,30,31,32};
        System.out.println(age[9]);
        }
```

}

Handle the exception by using throws keyword.

```
public class Demo{
    public static void main(String[] args) {
      int[] age = {10,20,25,24,28,27,30,31,32};
      try {
          System.out.println(age[9]);
      }
      catch(ArrayIndexOutOfBoundsException e) {
          System.out.println("Error: Index 9 out of bounds for length 9");
      }

        }
}
```

```
Console × Problems Debug Shell
<terminated> Demo [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 28, 2023, 1:09:07 PM – 1:09:07 PM)
Error: Index 9 out of bounds for length 9
```

Regular Expressions

16. Write a Java program to check whether a string contains only a certain set of characters (in this case a-z, A-Z and 0-9).

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegularExp {
    public static void main(String[] args) {
    Pattern p= Pattern.compile("[a-zA-Z0-9]+");
    Matcher m= p.matcher("Merina10");
    boolean b= m.matches();
    System.out.println(b);
    }
}
```

```
Console × Problems Debug Shell
<terminated> RegularExp [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.ex
true
```

17. Write a Java program to find the sequence of one upper case letter followed by lower case letters. Z

```
public class RegularExp{
    public static void main(String[] args) {
        Pattern p = Pattern.compile("^[A-Z][a-z]+$");
        Matcher m = p.matcher("Merina");
        boolean b = m.matches();
        System.out.println(b);
    }
}
```

Console × Problems Debug Shell

```
<terminated> RegularExp [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 28
true
```

18. Develop a Java program to check if a given string represents a file with a ".txt" extension.

```
public class RegularExp{
    public static void main(String[] args) {
        String input = "Jawdaw.txt";
        String allowed = "^[A-Z]+[a-z]+.txt$";
        if (Pattern.matches(allowed, input)) {
            System.out.println("Contains");
        } else {
            System.out.println("Does not Contains");
        }
    }
}
```

Console × Problems Debug Shell

```
<terminated> RegularExp [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec
Contains
```

19. Write a Java program that validates usernames based on the following criteria**:**

- Should start with a letter.
- Can include letters, numbers, and underscores.
- Should be between 3 and 16 characters in length.

```java

public class RegularExp{
public static void main(String[] args) {
    String input = "Jawdaw22312";
    String allowed = "^[A-Za-z][A-Za-z0-9_]+{3,16}$";
    if (Pattern.matches(allowed, input)) {
        System.out.println("Contains");
    } else {
        System.out.println("Does not Contains");
    }
    }
}
```

Console × Problems Debug Shell

```
<terminated> RegularExp [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 28
Contains
```