# 1. Abstract Class

## Task 1: Basic Abstract Class

```
abstract class Shape {
    abstract double area();
}

class Circle extends Shape {
    double radius;

    Circle(double r) {
        this.radius = r;
    }

    double area() {
        return Math.PI * radius * radius;
    }
}

class Rectangle extends Shape {
    double length, breadth;

    Rectangle(double l, double b) {
        this.length = l;
        this.breadth = b;
    }

    double area() {
        return length * breadth;
    }
}

public class TestShape {
    public static void main(String[] args) {
        Shape s1 = new Circle(5);
        Shape s2 = new Rectangle(4, 6);

        System.out.println("Circle Area: " + s1.area());
        System.out.println("Rectangle Area: " + s2.area());
    }
}
```

### □ Output:

```
Circle Area: 78.53981633974483
Rectangle Area: 24.0
```

---

### Task 2: Abstract + Non-abstract

```
abstract class Animal {
    abstract void sound();

    void displayType() {
        System.out.println("This is an animal.");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog says: Bark!");
    }
}

public class TestAnimal {
    public static void main(String[] args) {
        Animal a = new Dog();
        a.sound();
        a.displayType();
    }
}
```

---

## □ 2. Interface – Static and Default Methods

### Task 1: Abstract + Default

```
interface RemoteControl {
    void turnOn();

    default void batteryStatus() {
        System.out.println("Battery is 80%");
    }
}

class TV implements RemoteControl {
    public void turnOn() {
        System.out.println("TV is turned on.");
    }
}

public class TestInterface {
    public static void main(String[] args) {
        TV tv = new TV();
        tv.turnOn();
        tv.batteryStatus();
    }
}
```

---

## Task 2: Static Method in Interface

```
interface Info {
    static void showDetails() {
        System.out.println("Static method in Interface");
    }
}

public class TestStatic {
    public static void main(String[] args) {
        Info.showDetails();
    }
}
```

---

## Task 3: Multiple Interfaces

```
interface Printable {
    void print();
}

interface Scannable {
    void scan();
}

class MultiFunctionPrinter implements Printable, Scannable {
    public void print() {
        System.out.println("Printing document...");
    }

    public void scan() {
        System.out.println("Scanning document...");
    }
}

public class TestMultiInterface {
    public static void main(String[] args) {
        MultiFunctionPrinter mfp = new MultiFunctionPrinter();
        mfp.print();
        mfp.scan();
    }
}
```

---

## □ 3. Final Keyword

### Task 1: Final Variables

```
public class FinalVariable {
    final double PI = 3.14159;

    void printPI() {
        System.out.println("PI: " + PI);
    }

    // PI = 3.15; // Uncommenting this line will cause a compile error.
}
```

---

### Task 2: Final Method

```
class Vehicle {
    final void engineType() {
        System.out.println("Diesel engine");
    }
}

class Car extends Vehicle {
    // Cannot override engineType
    // void engineType() {} → Compile error
}
```

---

### Task 3: Final Class

```
final class Constants {
    static final String APP_NAME = "MyApp";
}

// class SubClass extends Constants {} // Compile error: Cannot inherit from
final class
```

---

# □ 4. Packages and Imports

### Task 1: Creating Package

```
// File: com/school/student/Student.java
package com.school.student;

public class Student {
    public void show() {
        System.out.println("Student class in com.school.student package");
    }
}

// File: Main.java
import com.school.student.Student;

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        s.show();
    }
}
```

---

### Task 2: Static Import

```
import static java.lang.Math.*;

public class StaticImport {
    public static void main(String[] args) {
        System.out.println(sqrt(16));
        System.out.println(PI);
    }
}
```

---

## □ 5. Exception Handling

### Task 1: Try-Catch

```
import java.util.Scanner;

public class DivideExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num;

        System.out.print("Enter a number: ");
        num = sc.nextInt();

        try {
            int result = 100 / num;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
        }
    }
}
```

---

### Task 2: Try-Catch-Finally

```
import java.io.*;

public class FileReadExample {
    public static void main(String[] args) {
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader("data.txt"));
            System.out.println(reader.readLine());
        } catch (IOException e) {
            System.out.println("Error reading file");
        } finally {
            try {
                if (reader != null) reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            System.out.println("File closed.");
        }
    }
}
```

---

## Task 3: Throw and Throws

```
class Voter {
    static void validateAge(int age) throws Exception {
        if (age < 18)
            throw new Exception("You must be 18+ to vote");
        else
            System.out.println("You can vote!");
    }

    public static void main(String[] args) {
        try {
            validateAge(16);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

---

## Task 4: Multiple Catch Blocks

```
public class MultiCatch {
    public static void main(String[] args) {
        try {
            int[] arr = new int[5];
            arr[5] = 100 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Divide by zero error.");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index error.");
        }
    }
}
```

---

## Task 5: Custom Exception

```
class InvalidAccountException extends Exception {
    InvalidAccountException(String msg) {
        super(msg);
    }
}

public class Bank {
    static void checkBalance(int balance) throws InvalidAccountException {
        if (balance < 0)
            throw new InvalidAccountException("Negative balance not
allowed!");
        else
            System.out.println("Valid balance: " + balance);
    }

    public static void main(String[] args) {
        try {
            checkBalance(-100);
        } catch (InvalidAccountException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

---

# □ Mini Library Management

```
// Abstract Class
abstract class Book {
    String title, author;

    Book(String t, String a) {
        title = t;
        author = a;
    }

    abstract void displayInfo();
}

// Interface
interface Borrowable {
    default void returnPolicy() {
        System.out.println("Return within 14 days.");
    }
}

// Final class
final class LibraryConfig {
    static final String LIBRARY_NAME = "Central Library";
    static final String ADDRESS = "Main Road";
}
```

```
class Novel extends Book implements Borrowable {
    Novel(String t, String a) {
        super(t, a);
    }

    void displayInfo() {
        System.out.println("Title: " + title + ", Author: " + author);
    }
}

public class LibraryApp {
    public static void main(String[] args) {
        System.out.println("Welcome to " + LibraryConfig.LIBRARY_NAME);
        Novel n = new Novel("The Alchemist", "Paulo Coelho");
        n.displayInfo();
        n.returnPolicy();
    }
}
```