NAME - ASHWINI BASAVARAJ SIMPAGER
PROJECT - Java Basics & OOPs Assignment Questions

Java Basics

1. **What is Java? Explain its features.**

Java is a high-level, class-based, and object-oriented programming language that is platform-independent and robust.

Features:

- Platform Independent: Java bytecode can run on any machine that has a JVM.
- Object-Oriented: Java supports OOP concepts such as classes and objects.
- Secure: Java provides a secure runtime environment.
- Robust: Java has strong memory management and exception handling.
- Multithreaded: Java supports multiple threads of execution.
- High Performance: Uses JIT compiler for optimized execution

---

2. **Explain the java program execution process**.

1. Write Java code and save it as .java file
2. Compile using `javac` → generates .class (bytecode)
3. Run the program using `java` command → executed via JVM

---

3. **Write a simple Java program to display 'Hello World'.**

```
Java
public class Hello Java {
   public static void main(String[] args) {
      System.out.println("Hello World from Gayatri!");
   }
}
```

---

4. What are data types in Java? List and explain them.

Java has two types:

- Primitive: int, float, char, double, byte, boolean, long, short
- Non-Primitive: String, Array, Class, Interface

Example:

java

int age = 22;

String name = "Gayatri";

---

5. Difference between JDK, JRE, and JVM

Term Description

JVM Runs Java bytecode

JRE JVM + libraries (for running Java apps)

JDK JRE + compiler and tools (for developing Java apps)

---

6. What are variables in Java? Explain with examples.

A **variable** is a container for storing data values.

Example:
```java
int marks = 85;
String student = "Gayatri";
```

---

7. Different types of operators in Java

- **Arithmetic**: +, -, *, /, %
- **Relational**: ==, !=, >, <, >=, <=
- - Logical: &&, ||, !
- **Assignment**: =, +=, -=, etc.
- **Unary**: ++, --
- **Bitwise**: &, |, ^

---

8. Control statements in Java (if, if-else, switch)

Java

```
int num = 10;
if (num > 5) {
   System.out.println("Greater than 5");
} else {
   System.out.println("Less than or equal to 5");
}

switch(num) {
   case 10: System.out.println("Ten"); break;
   default: System.out.println("Other number");
}
```

---

9. Java program to find even or odd number

```
import java.util.Scanner;

public class EvenOdd {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      int number = input.nextInt();
      if (number % 2 == 0)
         System.out.println("Even");
      else
         System.out.println("Odd");
   }
```

}

```java
public class EvenOdd {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number = input.nextInt();
        if (number % 2 == 0)
            System.out.println("Even");
        else
            System.out.println("Odd");
    }
}
```

---

10. Difference between while and do-while loop

While Loop	Do-While Loop
Condition checked first	Condition checked after execution
May never execute	Executes at least once

Object-Oriented Programming (OOPs)

1. Principles of OOPs in Java

- **Encapsulation**: Data hiding using classes
- **Abstraction**: Hiding implementation details
- **Inheritance**: Code reuse through subclasses
- **Polymorphism**: Many forms of methods/objects

---

2. What is a class and object in Java?

```
class Student {
   String name;

   void study() {
      System.out.println(name + " is studying...");
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student(); // object
        s1.name = "Gayatri";
        s1.study();
    }
}
```

```java
class Student {
    String name;

    void study() {
        System.out.println(name + " is studying...");
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student(); // object
        s1.name = "Gayatri";
        s1.study();
    }
}
```

---

4. Program to calculate area of triangle

```
class Triangle {
    double base, height;

    double calculateArea() {
        return 0.5 * base * height;
    }
}

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

```
    Triangle t = new Triangle();
    t.base = 8;
    t.height = 4;
    System.out.println("Area: " + t.calculateArea());
  }
}
```

```java
class Triangle {
    double base, height;

    double calculateArea() {
        return 0.5 * base * height;
    }
}

public class Main {
    public static void main(String[] args) {
        Triangle t = new Triangle();
        t.base = 8;
        t.height = 4;
        System.out.println("Area: " + t.calculateArea());
    }
}
```

---

5. Inheritance with real-life example

```
  class Vehicle {
  void start() {
    System.out.println("Vehicle is starting...");
  }
}

class Car extends Vehicle {
  void drive() {
    System.out.println("Car is driving...");
  }
}

public class Main {
  public static void main(String[] args) {
    Car myCar = new Car();
```

```
    myCar.start();  // Inherited method from Vehicle
    myCar.drive();  // Method from Car class
  }
}
```

```java
class Vehicle {
    void start() {
        System.out.println("Vehicle is starting...");
    }
}

class Car extends Vehicle {
    void drive() {
        System.out.println("Car is driving...");
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.start();  // Inherited method from Vehicle
        myCar.drive();  // Method from Car class
    }
}
```

---

5. What is polymorphism?

Compile-time (method overloading):

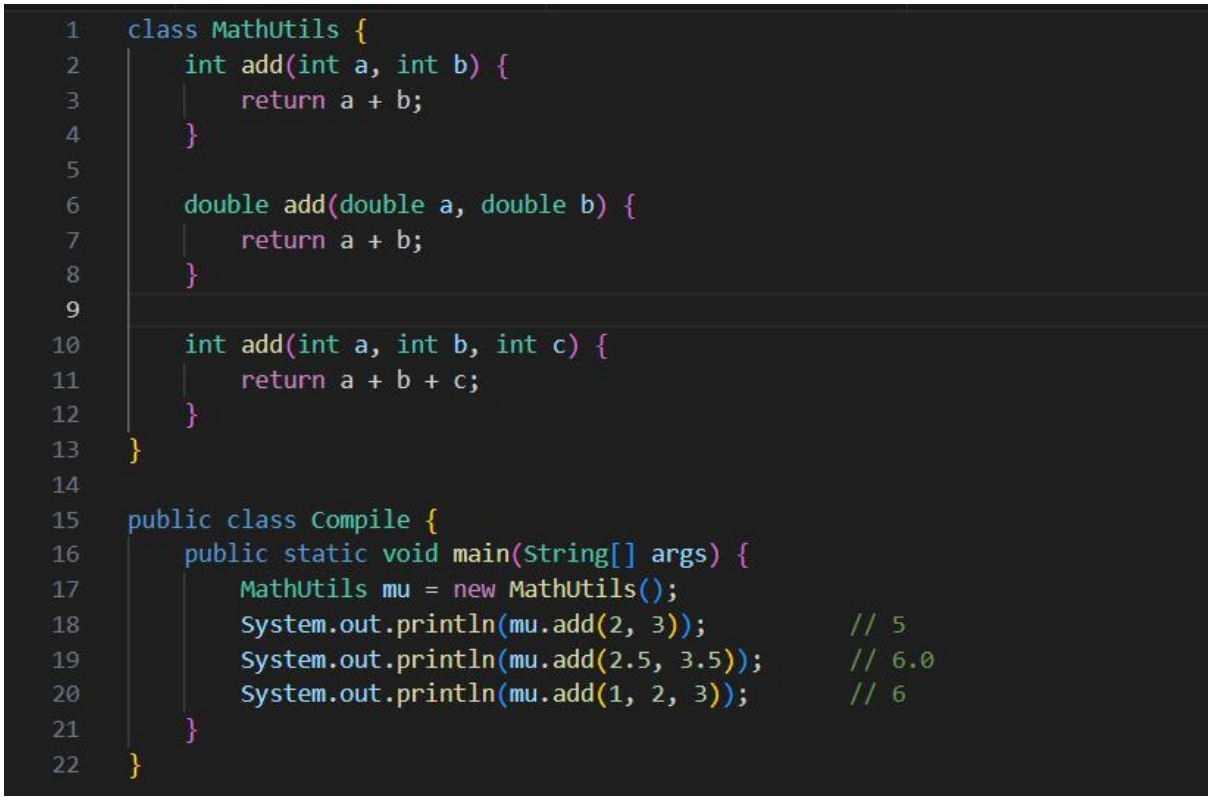
```
class MathUtils {
  int add(int a, int b) {
    return a + b;
  }

  double add(double a, double b) {
    return a + b;
  }

  int add(int a, int b, int c) {
    return a + b + c;
  }
```

```
}

public class Compile {
    public static void main(String[] args) {
        MathUtils mu = new MathUtils();
        System.out.println(mu.add(2, 3));         // 5
        System.out.println(mu.add(2.5, 3.5));     // 6.0
        System.out.println(mu.add(1, 2, 3));      // 6
    }
}
```



```
class MathUtils {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}

public class Compile {
    public static void main(String[] args) {
        MathUtils mu = new MathUtils();
        System.out.println(mu.add(2, 3));         // 5
        System.out.println(mu.add(2.5, 3.5));     // 6.0
        System.out.println(mu.add(1, 2, 3));      // 6
    }
}
```

---

6. Method Overloading vs Overriding

Overloading: Same method name, different parameters (same class)

Overriding: Same method name and parameters in subclass

---

7. Program for encapsulation

```
public class person {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        name = newName;
    }

    public int getAge() {
        return age;
    }
    public void setAge(int newAge) {
        if (newAge > 0) {
            age = newAge;
        } else {
            System.out.println("Age must be positive.");
        }
    }


    public static void main(String[] args) {
        person p1 = new person();

        p1.setName("Riya");
        p1.setAge(18);

        System.out.println("Name: " + p1.getName());
        System.out.println("Age: " + p1.getAge());
```

```
    }
}
```

```java
public class person {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        name = newName;
    }

    public int getAge() {
        return age;
    }
    public void setAge(int newAge) {
        if (newAge > 0) {
            age = newAge;
        } else {
            System.out.println("Age must be positive.");
        }
    }

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

```java
    public static void main(String[] args) {
        person p1 = new person();

        p1.setName("Riya");
        p1.setAge(18);

        System.out.println("Name: " + p1.getName());
        System.out.println("Age: " + p1.getAge());
    }
}

```

---

8. What is abstraction?

Abstraction means hiding details and showing only essential features. Achieved using:

- **Abstract class**
- **Interface**

---

9. Abstract class vs Interface

Abstract Class	Interface
Can have constructors	Cannot have constructors
Can have both abstract and concrete methods	All methods abstract (Java 7)
Supports inheritance	Supports multiple inheritance

---

10. Program using Interface

```
class Shape {
    void draw() {
        System.out.println("Drawing a shape");
    }
}

class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a circle");
    }
}

class Rectangle extends Shape {

      void draw() {
        System.out.println("Drawing a rectangle");
    }
}

public class Runtime {
    public static void main(String[] args) {
        Shape s;
```

```
        s = new Circle();     // object of Circle
        s.draw();             // Output: Drawing a circle

        s = new Rectangle();  // object of Rectangle
        s.draw();             // Output: Drawing a rectangle
    }
}
```

```java
class Shape {
    void draw() {
        System.out.println("Drawing a shape");
    }
}

class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a circle");
    }
}

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

```java
    }
}

public class Runtime {
    public static void main(String[] args) {
        Shape s;

        s = new Circle();     // object of Circle
        s.draw();             // Output: Drawing a circle

        s = new Rectangle();  // object of Rectangle
        s.draw();             // Output: Drawing a rectangle
    }
}
```