Short Answer

1. What is Object-Oriented Programming? How is it different from procedural programming?

- **OOP** is a programming paradigm based on the concept of **objects** that contain both **data (fields)** and **methods (functions)**.
- **Procedural programming** (like C) focuses on **functions/procedures** operating on data, while **OOP** (like Java) organizes code into **classes and objects**, emphasizing reusability, abstraction, and modularity.

---

2. Define class and object with an example in Java.

- **Class**: A blueprint or template for creating objects.
- **Object**: An instance of a class.

```
class Car {
    String brand;
    int speed;

    void drive() {
        System.out.println(brand + " is driving at " + speed + " km/h.");
    }
}

public class Main {
    public static void main(String[] args) {
        Car c1 = new Car(); // Object creation
        c1.brand = "Tesla";
        c1.speed = 120;
        c1.drive();
    }
}
```

---

3. What is the difference between method overloading and method overriding?

- **Overloading**: Same method name, different parameter list, within the same class. (Compile-time polymorphism)
- **Overriding**: Subclass provides a new implementation of a method already defined in the parent class. (Runtime polymorphism)

---

4. What is the use of the `this` keyword in Java?

- Refers to the **current object**.
- Used to:
    - Differentiate between instance variables and parameters.
    - Call another constructor in the same class.
    - Pass the current object as an argument.

---

5. What is the difference between constructor and method?

- **Constructor**: Special method used to initialize objects; has the same name as the class; no return type.
- **Method**: Defines behavior; must have a return type (even `void`); can be called multiple times.

---

6. What is inheritance and why is it used?

- **Inheritance** allows a class (child/subclass) to acquire the properties and behaviors of another class (parent/superclass).
- **Use**: Promotes code reusability, method overriding, and hierarchical classification.

📌 Long Answer

1. Explain the four pillars of OOP in detail with Java examples.

1. **Encapsulation**: Wrapping data and methods in a single unit (class).

```
class Account {
    private int balance;
    public void deposit(int amount) { balance += amount; }
    public int getBalance() { return balance; }
}
```

7. **Abstraction**: Hiding implementation details, showing only necessary features.

```
abstract class Vehicle {
    abstract void start();
}
class Car extends Vehicle {
    void start() { System.out.println("Car starts with a key"); }
}
```

14. **Inheritance**: Reusing code from a parent class.

```
class Animal { void eat() { System.out.println("Eating"); } }
class Dog extends Animal { void bark() {
System.out.println("Barking"); } }
```

17. **Polymorphism**: One name, many forms.

- Overloading (compile-time)
- Overriding (runtime)

---

2. Describe the process of object creation in Java with memory diagram.

Steps:

1. **Class loading** → Bytecode loaded by JVM.
2. **Object creation** → `new` allocates memory on heap.
3. **Constructor call** → Initializes object.
4. **Reference assignment** → Reference stored in stack points to heap object.

(Memory diagram would show: **Stack → Reference variable**, **Heap → Object fields**).

---

3. Explain access modifiers in Java and their differences.

- **public** → Accessible everywhere.
- **protected** → Accessible in the same package + subclasses.
- **default** (no modifier) → Accessible only in the same package.
- **private** → Accessible only within the class.

---

4. Discuss Java packages and how to create your own package.

- **Package**: A collection of classes and interfaces grouped together.
- **Creating**:

```
package mypack;  // create package
public class Hello {
    public void msg() { System.out.println("Hello from package"); }
}
```

- **Using**:

```
import mypack.Hello;
```

---

5. Describe interfaces in Java and their importance in multiple inheritance.

- **Interface**: Collection of abstract methods (Java 8+ allows default & static methods).
- Used for **multiple inheritance** since Java doesn't allow extending multiple classes.

```
interface A { void show(); }
interface B { void display(); }
class C implements A, B {
    public void show() { System.out.println("Show"); }
    public void display() { System.out.println("Display"); }
}
```

📌 Coding Questions

1. Student Class Example

```
class Student {
    String name;
    int roll;
    double marks;

    Student(String name, int roll, double marks) {
        this.name = name;
        this.roll = roll;
        this.marks = marks;
    }

    void display() {
        System.out.println("Name: " + name + ", Roll: " + roll + ", Marks: "
+ marks);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("Alice", 101, 85.5);
        Student s2 = new Student("Bob", 102, 90.0);
        s1.display();
        s2.display();
    }
}
```

---

2. Method Overloading

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

public class Main {
    public static void main(String[] args) {
        Calculator c = new Calculator();
        System.out.println(c.add(5, 10));
        System.out.println(c.add(2.5, 3.5));
    }
}
```

---

3. Method Overriding (Inheritance)

```
class Animal {
    void sound() { System.out.println("Animal makes a sound"); }
}
class Dog extends Animal {
    @Override
    void sound() { System.out.println("Dog barks"); }
}

public class Main {
    public static void main(String[] args) {
        Animal a = new Dog();
        a.sound();  // runtime polymorphism
    }
}
```

---

4. Abstract Class Shape

```
abstract class Shape {
    abstract double area();
}
class Circle extends Shape {
    double radius;
    Circle(double r) { radius = r; }
    double area() { return Math.PI * radius * radius; }
}
class Rectangle extends Shape {
    double length, width;
    Rectangle(double l, double w) { length = l; width = w; }
    double area() { return length * width; }
}
public class Main {
    public static void main(String[] args) {
        Shape c = new Circle(5);
        Shape r = new Rectangle(4, 6);
        System.out.println("Circle Area: " + c.area());
        System.out.println("Rectangle Area: " + r.area());
    }
}
```

---

5. Multiple Inheritance using Interfaces

```
interface A { void methodA(); }
interface B { void methodB(); }
class C implements A, B {
    public void methodA() { System.out.println("Method A"); }
    public void methodB() { System.out.println("Method B"); }
}
```

---

6. Constructor Overloading

```
class Person {
    String name;
    int age;

    Person() { name = "Unknown"; age = 0; }
    Person(String n) { name = n; age = 0; }
    Person(String n, int a) { name = n; age = a; }

    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```

---

7. Interface Playable

```
interface Playable {
    void play();
}
class Football implements Playable {
    public void play() { System.out.println("Playing Football"); }
}
class Cricket implements Playable {
    public void play() { System.out.println("Playing Cricket"); }
}
```

---

8. Difference between static methods and instance methods

- **Static method**: Belongs to the class, can be called without creating an object.
- **Instance method**: Belongs to an object, requires object creation.

```
class Example {
    static void staticMethod() { System.out.println("Static Method"); }
    void instanceMethod() { System.out.println("Instance Method"); }
}

public class Main {
    public static void main(String[] args) {
        Example.staticMethod();  // No object needed
        Example e = new Example();
        e.instanceMethod();      // Needs object
    }
}
```

1. Given a partial Java code, identify the errors and correct them

2. Given a partial Java code, identify the errors and correct them

```
class Car {
   String brand;
   int speed;

   void drive() {
      System.out.println("Driving " + brand at speed km/h);
   }
}

public class Main {
   public static void main(String[] args) {
      Car c = new Car;
      c.brand = "Toyota";
      c.speed = 100
      c.drive();
   }
}
```

❌ Errors in code:

1. `System.out.println("Driving " + brand at speed km/h);` → invalid string concatenation.
2. `Car c = new Car;` → must use `new Car();`
3. `c.speed = 100` → missing semicolon.

✅ Corrected Code:

```
class Car {
   String brand;
   int speed;

   void drive() {
      System.out.println("Driving " + brand + " at " + speed + " km/h");
   }
}

public class Main {
   public static void main(String[] args) {
      Car c = new Car();
      c.brand = "Toyota";
      c.speed = 100;
      c.drive();
   }
}
```

🔷 2. Java Program Using Inheritance (Vehicles Example)

```
class Vehicle {
    String brand;
    int speed;

    Vehicle(String brand, int speed) {
        this.brand = brand;
        this.speed = speed;
    }

    void display() {
        System.out.println("Brand: " + brand + ", Speed: " + speed + "
km/h");
    }
}

class Car extends Vehicle {
    int doors;

    Car(String brand, int speed, int doors) {
        super(brand, speed);
        this.doors = doors;
    }

    @Override
    void display() {
        super.display();
        System.out.println("Doors: " + doors);
    }
}

class Bike extends Vehicle {
    boolean hasGear;

    Bike(String brand, int speed, boolean hasGear) {
        super(brand, speed);
        this.hasGear = hasGear;
    }

    @Override
    void display() {
        super.display();
        System.out.println("Has Gear: " + hasGear);
    }
}

public class Main {
    public static void main(String[] args) {
        Car c = new Car("Honda", 180, 4);
        Bike b = new Bike("Yamaha", 120, true);

        c.display();
        b.display();
    }
}
```

🔷 3. Compare Abstract Classes and Interfaces with Examples

Abstract Class

- Can have **abstract methods** (no body) and **concrete methods** (with body).
- Can have **instance variables**.
- Supports **single inheritance only**.

```
abstract class Animal {
    abstract void sound();
    void sleep() { System.out.println("Sleeping..."); }
}

class Dog extends Animal {
    void sound() { System.out.println("Dog barks"); }
}
```

Interface

- Contains **abstract methods only** (Java 8+ allows default & static methods).
- Cannot have instance variables (only constants).
- Supports **multiple inheritance** (a class can implement many interfaces).

```
interface Playable {
    void play();
}

class Football implements Playable {
    public void play() { System.out.println("Playing Football"); }
}
```

🔷 4. Bank Account System Program

```
class BankAccount {
    private String accountHolder;
    private double balance;

    BankAccount(String accountHolder, double balance) {
        this.accountHolder = accountHolder;
        this.balance = balance;
    }

    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: " + amount);
        } else {
            System.out.println("Invalid deposit amount!");
        }
    }

    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew: " + amount);
        } else {
            System.out.println("Insufficient balance!");
        }
    }

    void displayBalance() {
        System.out.println("Account Holder: " + accountHolder + ", Balance: "
+ balance);
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount("Alice", 5000);
        acc.displayBalance();

        acc.deposit(2000);
        acc.withdraw(1500);
        acc.displayBalance();

        acc.withdraw(6000);  // shows insufficient balance
    }
}
```