Chapter-5: Object Oriented Programming

PART-II: SAMPLE SHORT QUESTIONS

Q1. How Inheritance is implemented in Java?
Inheritance in Java is implemented using the extends keyword. A subclass inherits fields and methods from a parent class, allowing code reuse and method overriding.

Example:

```
class Parent { void show() { System.out.println("Parent"); } }

class Child extends Parent { }
```

Q2. Are static members inherited to subclasses?
Yes, static members are inherited but they **cannot be overridden**. They belong to the class, not the object, and can be accessed using ClassName.member.

Example:

```
class A { static int x = 10; }

class B extends A { }

System.out.println(B.x);  // prints 10
```

Q3. What happens if the parent and child class have a field with same identifier?
If both classes define the same field, the child class hides the parent's field. Accessing the variable depends on the reference type, not the object.

Example:

```
class A { int x = 10; }

class B extends A { int x = 20; }

B obj = new B();

System.out.println(obj.x); // prints 20
```

Q4. Are constructors and initializers also inherited to subclasses?
No, constructors and initializers are **not inherited**. Each subclass must define its own constructor, but it can call the parent's constructor using super().

Example:

```
class A { A() { System.out.println("Parent"); } }

class B extends A { B() { super(); System.out.println("Child"); } }
```

Q5. How do you restrict a member of a class from inheriting by its subclasses?
You can restrict inheritance using the final keyword for methods or classes, or using private access modifier for members.

Example:

```
final class A { }   // cannot be inherited

private int x;      // not accessible in subclass
```

Q6. How do you implement multiple inheritance in Java?
Java does not support multiple inheritance using classes due to ambiguity (diamond problem). Instead, it is implemented through **interfaces**.

Example:

interface A { void show(); }

interface B { void display(); }

class C implements A, B { public void show(){} public void display(){} }

Q7. How many types of inheritance are there?
Types of inheritance in Java:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance (through interfaces)
5. Hybrid Inheritance (combination).

Q8. Can a class extend by itself in Java?
No, a class cannot extend itself because it would cause infinite recursion and logical errors.

Q9. Can we reduce the visibility of the inherited or overridden method?
No, visibility cannot be reduced while overriding. For example, a public method in the parent must remain public in the child.

Q10. How do you override a private method in Java?
Private methods cannot be overridden because they are not visible to subclasses. However, you can define a new method in the subclass with the same name (not overriding but hiding).

Q11. Why use Inheritance?

- Promotes code reuse.
- Supports method overriding (runtime polymorphism).
- Establishes "is-a" relationship.

Q12. What is the General format of Inheritance?

class Parent { }

class Child extends Parent { }

Q13. Why Multiple Inheritance is not supported in Java?
Because of the **diamond problem**: ambiguity occurs when two parent classes have the same method, and the child class doesn't know which one to inherit. Java solves this using **interfaces**.

Q14. What is Polymorphism?
Polymorphism means "many forms." In Java, it allows one task to be performed in multiple ways, such as method overloading and overriding.

Q15. What is Compile-time Polymorphism (Static Polymorphism)?
It is achieved through **method overloading** and resolved by the compiler.

Example:

```
class A {
  void sum(int a, int b){ System.out.println(a+b); }
  void sum(double a, double b){ System.out.println(a+b); }
}
```

Q16. What is Run-time Polymorphism (Dynamic Polymorphism)?
It is achieved through **method overriding** and resolved by the JVM at runtime.

Example:

```
class A { void show(){ System.out.println("Parent"); } }
class B extends A { void show(){ System.out.println("Child"); } }
A obj = new B(); obj.show();  // prints Child
```

Q17. What is Method Overloading?
Method overloading means defining multiple methods with the same name but different parameter lists.

Example:

```
void add(int a, int b) { }
void add(double a, double b) { }
```

Q18. What is Method Overriding?
When a subclass provides its own implementation of a method already defined in the parent class.

Example:

```
class A { void show(){ System.out.println("Parent"); } }
class B extends A { void show(){ System.out.println("Child"); } }
```

Q19. What are three ways to overload a method?

1. Different number of parameters.
2. Different data types of parameters.
3. Different order of parameters.

Q20. What is the Invalid case of method overloading?
Changing only the return type is invalid, because the compiler cannot distinguish between the methods.

Q21. What is Type Promotion?
When smaller data types are automatically converted into larger ones in overloaded methods.

Example:

void show(double x) { }

obj.show(10); // int promoted to double

Q22. What is the advantage of method overriding?
It supports **runtime polymorphism**, allowing a subclass to provide specific behavior for methods defined in the parent class.

Q23. What are Access Modifiers?
Access modifiers control visibility:

- public → accessible everywhere
- protected → accessible in package + subclass
- default → package-level access
- private → within class only

Q24. What is the advantage of Polymorphism?
Polymorphism improves flexibility and reusability. It allows writing code that can work on objects of different classes through a common interface.

Q25. What is Information Hiding?
It is the principle of hiding implementation details and exposing only essential features through methods.

Q26. Give any real-life Example of Information Hiding.
ATM Machine: You interact with buttons, but the internal logic of how cash is processed is hidden.

Q27. What is Encapsulation?
Encapsulation is bundling of data (variables) and behavior (methods) into a single unit (class).

Q28. How we can achieve encapsulation in Java?

- Declare variables as private.
- Provide public getter and setter methods.

Example:

class Student {

```
  private String name;

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

  public String getName(){ return name; }

}
```

Q29. What is Abstraction?
Abstraction hides implementation details and shows only functionality to the user. It is achieved using **abstract classes** and **interfaces**.

Q30. What are Abstract Classes?
Abstract classes are classes declared with the abstract keyword that cannot be instantiated but can contain abstract (without body) and non-abstract methods.

Example:

```
abstract class Shape { abstract void draw(); }

class Circle extends Shape { void draw(){ System.out.println("Circle"); } }
```

PART-III: SAMPLE LONG QUESTIONS

Q1. What is Inheritance? Explain the types of Inheritance with examples.

- **Definition:** Inheritance is a mechanism where one class acquires the properties and behaviors of another class using the extends keyword.
- **Types:**
    1. **Single Inheritance** → One child, one parent.
    2. **Multilevel Inheritance** → Child inherits from parent, which itself inherits from another class.
    3. **Hierarchical Inheritance** → Multiple classes inherit from one parent.
    4. **Multiple Inheritance (via interface)**.
    5. **Hybrid Inheritance (combination)**.

Example (Single Inheritance):

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

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

Q2. Explain the Difference between Method Overloading and Overriding in Java.

- **Method Overloading:**
    - Same name, different parameter list.
    - Happens at **compile-time**.
    - Example:
    - void show(int x){}
    - void show(double y){}
- **Method Overriding:**

- Same name, same parameter list in subclass.
- Happens at **runtime**.
- Example:
- class A{ void show(){} }
- class B extends A{ void show(){} }

Q3. Explain the three ways to overload a method with examples.

1. **By changing the number of arguments.**
2. **By changing the type of arguments.**
3. **By changing the order of arguments.**

Example:

```
void add(int a, int b){}

void add(double a, double b){}

void add(int a, double b){}
```

Q4. What is Encapsulation? Explain the advantages of Encapsulation.

- **Definition:** Binding data and methods into a single unit (class).
- **How:** Declare fields private, provide public getter and setter methods.
- **Advantages:**
    1. Security (restricts direct access).
    2. Flexibility (controlled access).
    3. Easy maintenance.
    4. Improves code reusability.

Example:

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

Q5. What are Abstract Classes? Explain these classes with examples.

- **Definition:** Classes that cannot be instantiated and are declared with abstract keyword.
- **Use:** Provide base for subclasses with partial implementation.
- **Features:** Can contain abstract methods (no body) and normal methods.

Example:

```
abstract class Shape {
  abstract void draw();
  void info(){ System.out.println("Shape Info"); }
}
class Circle extends Shape {
  void draw(){ System.out.println("Drawing Circle"); }
}
```