Q1. What is the difference between a class and an object? Provide a real-world example.

Answer:
A **class** is a blueprint or template that defines the properties (fields) and behaviors (methods) of an object, while an **object** is an instance created from that class. In other words, a class describes **what an object is**, and an object represents **a concrete realization of that description**.

Example:
A Car class may define attributes such as color, brand, and speed, and behaviors such as drive() and brake().
An actual car, such as “a red Toyota with speed 60 km/h,” is an object created from the Car class.

Q2. If you create a parameterized constructor in a class, what happens to the default constructor? What must you do if you still need it?

Answer:
If a class defines a **parameterized constructor**, Java will **not automatically generate a default constructor**. If the default (no-argument) constructor is still needed, it must be **explicitly defined** in the class.

Example:

```
public ClassName() {
   // default constructor
}
```

Q3. What are the four access modifiers in Java? List them from most restrictive to least restrictive.

Answer:
The four access modifiers in Java, from **most restrictive to least restrictive**, are:

1. private – accessible only within the same class
2. default (no modifier) – accessible within the same package
3. protected – accessible within the same package or subclasses
4. public – accessible from anywhere

Q4. Explain the difference between method overloading and method overriding.

Answer:

- **Method overloading** occurs when multiple methods in the **same class** have the same method name but **different parameter lists**. It is resolved at **compile time**.
- **Method overriding** occurs when a subclass provides a **specific implementation** of a method that is already defined in its parent class. It is resolved at **runtime**.

Overloading improves flexibility within a class, while overriding enables polymorphic behavior across class hierarchies.

Q5. Can you override a final method? Can you override a private method? Explain why or why not.

Answer:

- A **final method cannot be overridden** because it is explicitly marked to prevent modification in subclasses.
- A **private method cannot be overridden** because it is not accessible to subclasses. Instead, a method with the same name in a subclass is considered a new method, not an override.

Q6. What is the difference between static polymorphism and dynamic polymorphism? When does each occur?

Answer:

- **Static polymorphism** occurs at **compile time** and is achieved through **method overloading**.
- **Dynamic polymorphism** occurs at **runtime** and is achieved through **method overriding**, where the method call is resolved based on the object's actual type.

Q7. Why does Java not support multiple inheritance with classes? How can you achieve multiple inheritance in Java?

Answer:
Java does not support multiple inheritance with classes to avoid ambiguity issues such as the **Diamond Problem**, where multiple parent classes define the same method. Java achieves multiple inheritance through **interfaces**, where a class can implement multiple interfaces and provide its own implementations of their methods.

Q8. Consider the following code:

List<Integer> lst = new ArrayList<>();

Which principle of OOP does this demonstrate? Explain.

Answer:
This code demonstrates the principle of **polymorphism**. The variable is declared using the interface type List, while the actual object is an instance of ArrayList. This allows the implementation to be changed easily without affecting the rest of the code, improving flexibility and maintainability.