Chuwa - Java Backend 1212 Assignment

Delin "Angelina" Liang

14 Dec 2025

Question 1. What is the difference between a class and an object? Provide a real-world example.

ANSWER

A `Class` is a **blueprint/template** of a module. It's only a definition
An `Object` is the **instance** of the module, so it contains actual data and behavior

Real-world Example:
There is a `Class` of `Car`s that defines a "Car"should have the following properties: `numSeats, fuelCapacity, maxSpeed, Brand`, and can `Drive, Stop, Refuel`, etc.
And based on this design, one can build/create an actual `Toyota` car `myCar`, with `5 Seats, 13 gallon fuelCap` and `max 200 mi Speed`

Question 2. 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

Since the JVM would only provide the default constructor if one does not create their own parameterized constructor. Therefore, if you create a parameterized constructor in a class, the default constructor will no longer be generated.

If we still need a default constructor, you have to declare it yourself. For example:

```
public class User {
    public User() {
        // default constructor
    }

    public User(String name) {
        this.name = name;
    }
    // since we declare a parameterized constructor here
    // JVM will no longer create a default constructor for us
}
```

Question 3. What are the four access modifiers in Java? List them from most restrictive to least restrictive.

ANSWER

The four access modifiers from most restrictive to least restrictive are as follows:

- `private` : declarations are visible **within the class only**
- `default` : declarations are visible only **within the package (package private)**
- `protected` : declarations are visible **within the package or all subclasses**
- `public` : declarations are visible **everywhere**

Question 4. Explain the difference between method overloading and method overriding.

ANSWER

The difference between method overloading and method overriding is as follows:

- **Binding Time**: Overloading binds at compile time, while overriding binds at runtime
- **Performance Implications**: Overloading gives better performance due to compile-time resolution, and overriding gives lower performance due to runtime dispatch
- **Access Modifier Constraints**: `private` and `final` methods CAN be overloaded but CANNOT be overridden
- **Return Type**: Return type of method does not matter in case of method overloading, but they must be the same in the case of overriding.
- **Parameter Signature**: Arguments must be different in the case of overloading and must be the same in the case of overriding
- **Class Relationship**: Overloading must be done in the same class, while overriding requires the existence of both the base and derived(child) classes
- **Use Case**: Overloading is mostly used to increase the readability of the code, while overriding is mostly used to provide the implementation of the method that is already provided by its base class

Question 5. Can you override a `final` method? Can you override a `private` method? Explain why or why not.

ANSWER

Can you override a `final` method?
No
Because the `final` keyword is explicitly used to prevent subclasses from changing the parent class's implementation, ensuring the method's behavior remains consistent and unchanged, which is the opposite of the purpose of overriding–providing the child class an alternative behavior of the parent class method

Can you override a `private` method?
No
Because the core principle of overriding relies on inheritance, but private methods cannot be inherited by subclasses, which means one does not even have access to the private class, let alone write another version

Question 6. What is the difference between static polymorphism and dynamic polymorphism? When does each occur?

ANSWER

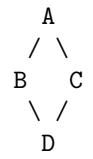
Static polymorphism happens through method **overloading** at **compile time** for speed but less flexibility
Dynamic polymorphism happens through method **overriding** at **runtime** which is slightly slower due to runtime lookup but more flexible

Question 7. Why does Java not support multiple inheritance with classes? How can you achieve multiple inheritance in Java?

ANSWER

In Java, multiple inheritance is not supported primarily to **avoid complexity and ambiguity problems**:

- **The Diamond Problem** (the core issue): For the following class structure:



  The compiler cannot determine which version of the method should be used when called on an instance of Class D, which is what is called ambiguity. Since Java is designed to be "simple and safe we would want to avoid this kind of problem

- **Simplicity over Complexity**: Similarly, since Java is intended to be simple and robust, we would want a class structure that is easier to maintain
- **Compiler/JVM Implementation Difficulty**: Supporting multiple inheritance of implementation makes the internal workings of the compiler and JVM significantly more complex, affecting areas like memory layout, casting, and method dispatch

Alternative:
In Java, you can use **interfaces(implements)** to achieve Multiple and Hybrid inheritance, which allows the class to inherit multiple sets of behaviors without the risks associated with multiple class inheritance

Question 8. Consider the following code:

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

Which principle of OOP does this demonstrate? Explain.

ANSWER

The provided code is demonstrating the concept of **Polymorphism** and **Abstraction**.
Detailed explanation is as follows:

- **Abstraction**: here, `List` is an abstract interface, the `ArrayList` is implementing from the interface, defining detailed behavior
- **Polymorphism**: The variable is declared using the interface type `List`, but the actual object created is an instance of `ArrayList`. This is an example showing that a parent type can refer to a child type, and also allows users to switch implementations later without changing any code (for example, switching `lst` from `ArrayList` to `LinkedList`.