Week 7 - LAQ's

Instructions

Explain method overriding in Java.

Method overriding in Java is a feature that allows a subclass (child class) to provide a specific implementation of a method that is already defined in its superclass (parent class). This is a fundamental concept in object-oriented programming that supports runtime polymorphism, enabling a program to decide at runtime which method to execute based on the object type.

Key Points of Method Overriding:

1. Same Method Signature: The overriding method in the subclass must have the same name, return type, and parameter list as the method in the superclass. This is what distinguishes it as an override rather than an overload.

2. Access Modifiers: The access level of the overriding method can be the same or more permissive than the method in the superclass. For example, if the superclass method is protected, the overriding method can be protected or public, but not private.

3. Use of super Keyword: Within the overriding method, you can use the super keyword to call the superclass's version of the method if needed.

4. Dynamic Method Dispatch: Method overriding is a key mechanism for achieving dynamic method dispatch, which allows Java to determine at runtime which method to invoke based on the actual object type, rather than the reference type.

5. Final Methods: If a method is declared as final in the superclass, it cannot be overridden in the subclass.

6. Static Methods: Static methods cannot be overridden. If a subclass defines a static method with the same name and parameters as a static method in the superclass, it is considered method hiding, not overriding.

Example of Method Overriding:

Here's a simple example to illustrate method overriding:

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

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

class Cat extends Animal {
    @Override
```

```
    void sound() {
        System.out.println("Cat meows");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        Animal myCat = new Cat();

        myDog.sound(); // Output: Dog barks
        myCat.sound(); // Output: Cat meows
    }
```

Explanation of the Example:

1. Superclass: The Animal class has a method sound().
2. Subclasses: The Dog and Cat classes extend Animal and override the sound() method to provide specific implementations.
3. Dynamic Method Dispatch: In the main method, we create references of type Animal that point to Dog and Cat objects. When we call the sound() method on these references, Java determines at runtime which method to execute based on the actual object type (i.e., Dog or Cat).

Benefits of Method Overriding:

- Polymorphism: It allows for polymorphic behavior, where a single interface can be used to represent different underlying forms (data types).
- Code Reusability: It promotes code reusability and extensibility, allowing subclasses to provide specific implementations while still adhering to a common interface defined by the superclass.

In summary, method overriding is a powerful feature in Java that enhances the flexibility and maintainability of code by allowing subclasses to define their own behavior while still conforming to the structure defined by their parent classes.