In Java, the diamond problem is a well-known issue in multiple inheritance scenarios, where a class inherits from two classes that have a common ancestor. This can lead to ambiguity about which inherited method to use. Java avoids this problem with classes by not supporting multiple inheritance of classes. However, with the introduction of default methods in interfaces in Java 8, a similar issue can arise.

Diamond Problem with Interfaces
In Java 8, interfaces can have default methods, which are methods with a body. This means that a class can inherit multiple default methods from different interfaces, potentially leading to ambiguity if those interfaces have methods with the same signature.

Example of the Diamond Problem
Consider the following example where two interfaces have default methods with the same signature:

Java

Collapse
```java
interface InterfaceA {
    default void show() {
        System.out.println("InterfaceA show");
    }
}

interface InterfaceB {
    default void show() {
        System.out.println("InterfaceB show");
    }
}

class ImplementingClass implements InterfaceA, InterfaceB {
    @Override
    public void show() {
        // Resolving the ambiguity by explicitly choosing which interface's method to call
        InterfaceA.super.show();
        InterfaceB.super.show();
    }
}

public class DiamondProblemExample {
    public static void main(String[] args) {
        ImplementingClass obj = new ImplementingClass();
        obj.show();
    }
}
```
Explanation
Ambiguity: The ImplementingClass inherits the show method from both InterfaceA and InterfaceB. This creates ambiguity because the Java compiler cannot determine which show method to use.

Resolution: The ambiguity is resolved by overriding the show method in ImplementingClass and explicitly specifying which interface's default method to call using InterfaceName.super.methodName().

Key Points
Default Methods: Allow interfaces to have method implementations, which can lead to ambiguity if multiple interfaces provide the same default method.

Explicit Resolution: When a class implements multiple interfaces with conflicting default methods, the class must override the method and explicitly choose which default method to call.

No Multiple Inheritance of State: Java still does not support multiple inheritance of state (fields), so the diamond problem is limited to method inheritance with default methods.

Interview Questions
What is the diamond problem in Java, and how does it relate to default methods in interfaces?
How can you resolve method ambiguity when implementing multiple interfaces with default methods?

Why does Java not support multiple inheritance of classes, and how do default methods in interfaces provide a workaround?
By understanding how to handle the diamond problem with default methods, developers can effectively use multiple interfaces in Java 8 without encountering ambiguity issues. This allows for more flexible and modular code design.