What are default methods?

Before Java 8, we could only declare abstract methods in an interface. However, Java 8 introduced the concept of default methods. **Default methods are methods that can have a body**. The most important use of default methods in interfaces is to provide additional functionality to a given type without breaking down the implementing classes.

Before Java 8, if a new method was introduced in an interface then all the implementing classes used to break. We would need to provide the implementation of that method in all the implementing classes.

However, sometimes methods have only single implementation and there is no need to provide their implementation in each class. In that case, we can declare that method as a default in the interface and provide its implementation in the interface itself.

Syntax of default methods

Let's understand the syntax of default methods through an example. Here, we have an interface with one abstract and one default method:

```
public interface Vehicle {

    void cleanVehicle();

    default void startVehicle() {
        System.out.println("Vehicle is starting");
    }
}
```

Now we will create a class which implements the vehicle interface.

Car.java
Vehicle.java

```
public class Car implements Vehicle {
    @Override
    public void cleanVehicle() {
        System.out.println("Cleaning the vehicle");
    }

    public static void main(String args[]){
        Car car = new Car();
        car.cleanVehicle();
        car.startVehicle();
    }
}
```

Run Save Reset

As shown above, our class needs to implement only the abstract method. When we call the default method, the code defined in the interface is executed.

How to resolve issues raised due to the default method

Although default methods are very good additions to Java and make developing a lot easier, they have one caveat that needs to be considered while coding.

To see this caveat, Let's look at an example. Here, we have two interfaces with a default method of the same name, i.e., `printSomething()`.

InterfaceA:

```
public interface InterfaceA {

    default void printSomething() {
        System.out.println("I am inside A interface");
    }
}

```

InterfaceB:

```
public interface InterfaceB {

    default void printSomething() {
        System.out.println("I am inside B interface");
    }
}

```

Now we will define a `Main` class that will implement both these interfaces. Before we proceed further I would like you to think about below questions:

1. Do we need to implement the `printSomething()` method in the `Main` class? Will the class compile if we don't?
2. If some class calls the `printSomething()` method from the object of `Main` class then which implementation will be called? Will it call the method defined in interfaceA or interfaceB?

Before I answer these questions let us create the `Main` class that will implement both the interfaces.

Main.java
InterfaceB.java
InterfaceA.java

```
public class Main implements InterfaceA, InterfaceB {

}

```

Run Save Reset

The above class will not compile because of the Diamond problem in Java. To resolve the compilation issue, we will have to implement the `printSomething()` method as shown below:

Main.java
InterfaceB.java
InterfaceA.java

```
public class Main implements InterfaceA, InterfaceB {

    @Override
    public void printSomething() {

        //Option 1 -> Provide our own implementation.
        System.out.println("I am inside Main class");

        //Option 2 -> Use existing implementation from interfaceA or interfac
        InterfaceA.super.printSomething();
        InterfaceB.super.printSomething();
    }

    public static void main(String args[]){
         Main main = new Main();
         main.printSomething();
    }
}

```

Run Save Reset