**Default methods:**

Until JDK1.8 whatever the method we declared inside an interface is public abstract methods and data members are public static final data members.

But from JDK1.8 onwards inside an interface we can declare default concrete methods.

These methods are also called as defender methods.

We declare default methods with the keyword "default"

Whatever the default methods we declare inside the interface are available for all implementation classes.

Based on our requirement we can invoke these default methods or we can change the definition of these default methods by doing overriding.

**What is the main advantage of default method?**

ans: Without effecting implementation classes we can add new functionality to the interface. (backward compatibility).

Note: we can’t override java.lang.Object class methods as default methods inside interface. If we try to do so then we will get compilation error.

interface Calc{

public default int hashCode(){ return 0; }

}

above code gives compilation error.

Reason: object class methods are available by default to every class. Hence we cannot bring those methods again as default methods.

**Default methods in multiple inheritance:**

interface Runner{

public default void doIt(){

System.out.println("I am running");

}

}

interface Executor{

public default void doIt(){

System.out.println("I am doing");

}

}

class RunExec implements Runner, Executor{

}

compile the above class you will get c error.

But below class will not give c error

class RunExec implements Runner, Executor{

public void doIt(){

System.out.println("do it from run exec");

}

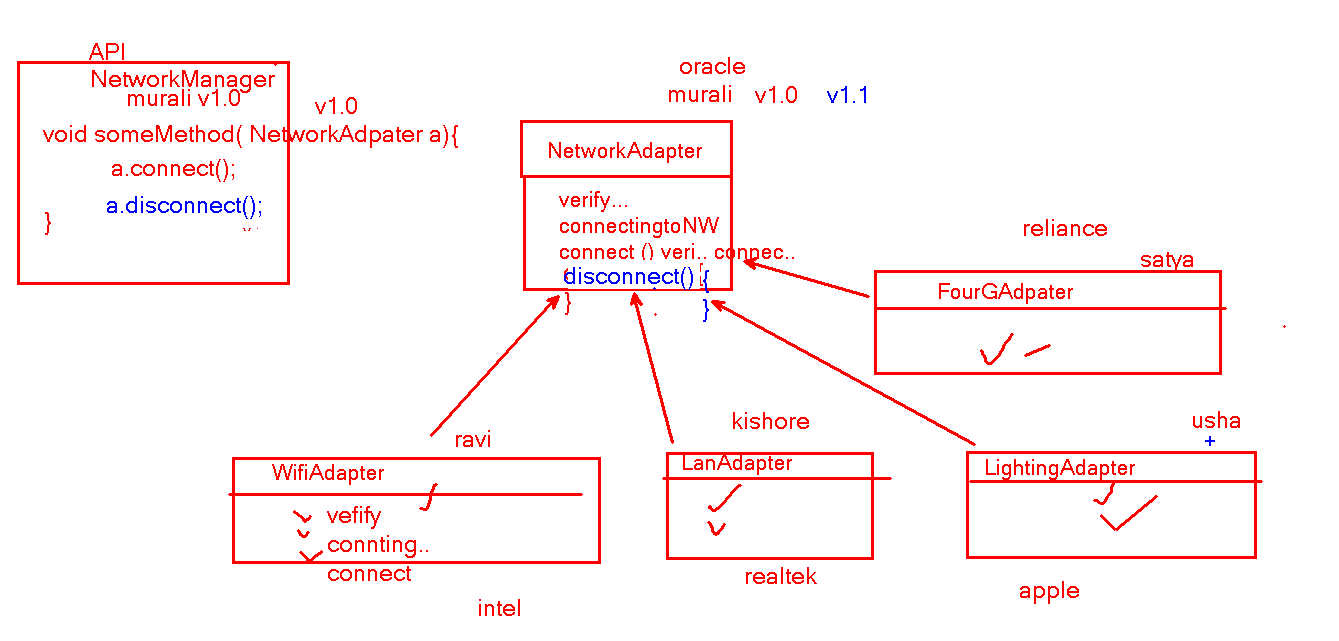
}

If two/more interfaces contain same default method (with same signature) then there may be a chance that a class may implement those interfaces. In such case the implementing class must provide def / override that default method.

Even though we can add concrete methods in the form of default methods to the interface, it will not be equals to abstract class.

Important note: interface with default method is not an abstract class.

|  |  |
| --- | --- |
| **Interface with default methods** | **Abstract class** |
| Inside interface every variable is always public static final and there is no chance of instance variable | Inside abstract class there may be a chance of instance variable which are required to the child class. |
| Interface never talks about state of the object | Abstract class can talk about state of the object |
| Inside interface we cannot declare constructors | Inside abstract class we can declare constructors. |
| Inside interface we cannot declare instance and static blocks | Inside abstract class we can declare instance and static blocks. |
| Inside interface we cannot override object class methods | Inside abstract class we can override object class methods. |
| Functional interface with default methods can refer lambda expression | Abstract class cannot refer lambda expression. |



**Static methods inside interface**

We can write static methods inside interface.

Static methods in interfaces are used to define utility functions.

The static methods defined inside the interface are not available to the implementation classes.

So by using implementation class name or implementation class reference we can not call static methods.

We can call static methods only by using interface name.

interface Foo{

static void fun(){

System.out.println("Foo fun method");

}

}

class FooImpl implements Foo{

}

public class StaticMethodDemo{

public static void main(String[] args){

Foo.fun();//allowed

FooImpl.fun(); // not allowed

FooImpl f = new FooImpl();

f.fun(); // not allowed

Foo foo = f;

foo.fun(); // not allowed

}

}

because of static methods feature we can write main() method inside the interface as well.

Bar.java

public interface Bar{

public static void main(String[] args){

System.out.println("Main inside interface");

}

}

javac Bar.java

java Bar