JAVA 8 – Static and Default Methods in Interface

Interface world before JAVA 8

All Java developers learnt this sentence in early days of their practice “Java interface can only contain methods which are declared as public and abstract in nature.” This statement was absolute true before Java 8, which made some significant changes the way interface can be used in Java framework.

Introduction of Default and Static Methods in Interface

Java 8 introduces concept of “Default Methods” and “Static Methods” (Similar what we have for class) for interfaces, which allows developer to enhance existing interface used in their applications without breaking prior implementation of that interface. This new feature of Java 8 is used extensively in core Java framework packages.

Why Introduced? What are the benefits for this approach?

One can debate that what was wrong with Interface having only abstract methods? Why did they change it and make it more complex than before?

Well answer is developers at oracle tried to simplify things. They want to have all related methods at one place instead of having separate base utility class for interface and then other classes extend that class and overrides applicable methods. Another reason is they want to enhance core framework to support new features like lambda expressions, Stream API without breaking millions of lines of code.

With this approach, Developers can add new default and static methods to interface and they don’t need to change all implementing classes of it.

Default Methods

“Default Methods” looks like methods declared in a class but it’s defined with body in an interface and indicated by modifier **default**. As its name suggests these methods are available to all classes who implements an interface. Please note that “Default” methods are public by default like other methods declared in an interface, so no need to specify public on it.

```
public interface BankCustomer {
    // Abstract methods
    Customer getCustomer(String id);

    void setCustomer(String id);

    //default method
    default String getAccountInfo(String id) {
        // return account info of customer
    }
}
```

In an above example we have two abstract methods like an old interface and one default method with its body. If you try to just declare default method without its body then it will throw compilation error.

Now all implementation classes can decide to override this method or just ignore it. (Without any errors of missing method overridden of an interface)

```
public class PersonalAccountHolder implements BankCustomer {
    @Override
    public Customer getCustomer(String id) {
        //return customer
    }

    @Override
    public void setCustomer(String id) {
        //set customer
    }
}
```

Although **getCustomer()** and **setCustomer()** are must to implement. It is optional to override default method **getAccountInfo()** and application will not complain at the time of compilation. However developer may decide to override this method when necessary and you can even redefine it as an abstract method in an abstract class, which in turn will force all child classes of that abstract class to implement it.

```
public class BusinessAccountHolder implements BankCustomer{
    @Override
    public Customer getCustomer(String id) {
        // return customer
    }

    @Override
    public void setCustomer(String id) {
        // set customer
    }

    // overriding default methods allows you to handle different logic for class
    @Override
    public String getAccountInfo(String id) {
        // return account details of customer
    }
}
```

```
public abstract class PersonalAccountHolder implements BankCustomer {
    public abstract String getAccountInfo(String id);
}
```

Now all classes which extends PersonalAccountHolder must need to implement getAccountInfo method.

If another interface implements the interface with default methods then child interface can redefine default methods and all classes implementing child interface will use implementation of child interface instead of the one specified by the parent interface.

When multiple interfaces have default method with same signature then we will have multiple inheritance problem in implementing class. In such case class will have to provide implementation for default method or explicitly specify interface name before calling default method.

```
public interface BankCustomer {
    default String getAccountInfo(String id) {
        // return account info of customer
        return null;
    }
}


public interface CreditUnionCustomer {
    default String getAccountInfo(String id){
        // return account info of customer
        return null;
    }
}

public class PersonalAccountHolder implements BankCustomer,CreditUnionCustomer {
    @Override
    public String getAccountInfo(String id) {
        // implement default method here
    }
}
```

Or if we want to invoke implementation of an interface then we can do so as follows,

```
public class PersonalAccountHolder implements BankCustomer,CreditUnionCustomer {
    @Override
    public String getAccountInfo(String id) {
        return BankCustomer.super.getAccountInfo(id);
    }
}
```

Special Note:

You cannot use default methods to override any non-final methods of Object class as Object is base class of all java classes. For example,

```
default int hashCode(){

}
```

When you do something like this it will throw compilation error like “Default method ‘hashcode’ overrides a member of ‘java.lang.Object’”.

Takeaway Points:

1) Default methods will help us to add new functionality to application framework without worrying about braking implementation classes.
2) With Default methods in an interface we can eliminate need of utility class at some places and have all relevant code at the same place.
3) Default methods are not mandatory to be overridden by implementing classes.

4) Default methods in Java interface also called as “Defender Methods” or “Virtual Extension Methods”.

Static Methods

We already know this: “Static method is associated with a class and every instance of class shares same static method and we can call those methods with class name itself.” With Java 8, Oracle introduced something similar for interfaces. Now we can declare static methods in an interface itself and call them by using interface name from all the classes which implements that interface.

Prior to Java 8, we had to create utility classes to define static methods, but now Java has the support for static methods in interface itself, which helps to keep code clean and understandable. With this approach it’s easy to organize helper methods in your application framework.

Syntax is similar to “default” methods we just need to use “static” keyword instead of “default”.

```
public interface BankCustomer {
    default String getAccountInfo(String id) {
        // return account info of customer
        return null;
    }

    static int totalBankCustomerCnts(){
        // return customer count
    }
}
```

In the given example, we have static method “totalBankCustomerCnts” defined in an interface. We can call them in implementation class as below:

```
public class PersonalAccountHolder implements BankCustomer,CreditUnionCustomer {
    @Override
    public String getAccountInfo(String id) {
        return BankCustomer.super.getAccountInfo(id);
    }

    // Calling static method with Interface name itself without creating an instance
    int custCount = BankCustomer.totalBankCustomerCnts();
}
```

When you try to call static method through an instance it will throw compilation error.

When class overrides static methods of its interface, we can add method with same signature like static method of an interface but that is not overriding actual method as static method can only called by an Interface itself.

Similar like default method we cannot define static method for Object class method, if we do so then we will get compilation error for it.

Special Note: Java functional interface can have multiple default and static methods.

Takeaway Points:

1) Static methods are part of interface and cannot be used with implementation class objects.
2) Static methods are introduced in an interface to remove utility classes such as famous "Collections" class for example, Collections.sort() and all static methods of it moved to corresponding interface.
3) Static method can be used effectively to create utility methods in an interface.

Abstract Class vs Default/Static methods

You must be thinking this sound like an Abstract Class. You are right at some level as they narrowed down gap between Interface and Abstract Class. But still you need to remember that, Abstract class can have instance variables and constructor whereas Interface can only define constants and abstract, default and static methods.

Final Words

Many Java framework interface has been updated extensively by adding default and static methods to them. Some notable Interfaces are Iterator, Collection, Comparator, Iterable.