Default Method Fundamentals

- An interface default method is defined similar to the way a method is defined by a **class**.
- The primary difference is that the declaration is preceded by the keyword **default**.
- For example, consider this simple interface:

Default Method Fundamentals

```
public interface MyIF {
  // This is a "normal" interface method declaration.
  // It does NOT define a default implementation.
  int getUserID();

  // This is a default method. Notice that it provides
  // a default implementation.
  default int getAdminID() {
    return 1;
  }
}
```

Default Method Fundamentals

- Because **getAdminID()** includes a default implementation, it is not necessary for an implementing class to override it.
- In other words, if an implementing class does not provide its own implementation, the default is used.
- For example, the **MyIFImp** class shown next is perfectly valid:

Default Method Fundamentals

```
// Implement MyIF.
class MyIFImp implements MyIF {
  // Only getUserID() defined by MyIF needs to be implemented.
  // getAdminID() can be allowed to default.
  public int getUserID() {
    return 100;
  }
}
```

Default Method Fundamentals

- The following code creates an instance of **MyIFImp** and uses it to call both **getUserID()** and **getAdminID( ).**

```
// Use the default method.
class DefaultMethodDemo {
  public static void main(String[] args) {

    MyIFImp obj = new MyIFImp();

    // Can call getUserID(), because it is explicitly
    // implemented by MyIFImp:
    System.out.println("User ID is " + obj.getUserID());

    // Can also call getAdminID(), because of default
    // implementation:
    System.out.println("Administrator ID is " + obj.getAdminID());
  }
}
```

Default Method Fundamentals

- It is both possible and common for an implementing class to define its own implementation of a default method.
- For example, **MyIFImp2** overrides **getAdminID( )**, as shown here:

Default Method Fundamentals

```
class MyIFImp2 implements MyIF {
  // Here, implementations for both getUserID( ) and getAdminID( ) are
  // provided.
  public int getUserID() {
    return 100;
  }

   public int getAdminID() {
     return 42;
   }
 }
```

Default Method Fundamentals

- Now, when **getAdminID( )** is called, a value other than its default is returned.

Use static Methods in an Interface

- Like **static** methods in a class, a **static** method defined by an interface can be called independently of any object.
- Thus, no implementation of the interface is necessary, and no instance of the interface is required in order to call a **static** method.

Use static Methods in an Interface

- Instead, a **static** method is called by specifying the interface name, followed by a period, followed by the method name.
- Here is the general form:

InterfaceName.staticMethodName

- Notice that this is similar to the way that a **static** method in a class is called.

Use static Methods in an Interface

- The following shows an example of a **static** method in an interface by adding one to **MyIF**, shown earlier. The **static** method is **getUniversalID( )**. It returns zero.

Use static Methods in an Interface

```
public interface MyIF {
  // This is a "normal" interface method declaration.
  // It does NOT define a default implementation.
  int getUserID();

  // This is a default method. Notice that it provides
  // a default implementation.
  default int getAdminID() {
    return 1;
  }

  // This is a static interface method.
  static int getUniversalID() {
    return 0;
  }
}
```

Use static Methods in an Interface

- The **getUniversalID( )** method can be called, as shown here:

```
int uID = MyIF.getUniversalID();
```

Private Interface Methods

- A private interface method can be called only by a default method or another private method defined by the same interface.
- Because a private interface method is specified private, it cannot be used by code outside the interface in which it is defined.
- This restriction includes subinterfaces because a private interface method is not inherited by a subinterface.

- The key benefit of a private interface method is that it lets two or more default methods use a common piece of code, thus avoiding code duplication.

Private Interface Methods

- For example, here is a further enhanced version of the Series interface that adds a second default method called **skipAndGetNextArray( ).**
- It skips a specified number of elements and then returns an array that contains the subsequent elements.
- It uses a private method called **getArray( )** to obtain an element array of a specified size

Private Interface Methods

```
// A further enhanced version of Series that includes two
// default methods that use a private method called getArray();
public interface Series {
  int getNext(); // return next number in series

  // Return an array that contains the next n elements
  // in the series beyond the current element.
  default int[] getNextArray(int n) {
    return getArray(n);
  }
```

Private Interface Methods

```
// Return an array that contains the next n elements
// in the series, after skipping elements.
default int[] skipAndGetNextArray(int skip, int n) {

  // Skip the specified number of elements.
  getArray(skip);

  return getArray(n);
}
```

Private Interface Methods

```
  // A private method that returns an array containing
  // the next n elements.
  private int[] getArray(int n) {
    int[] vals = new int[n];
    for(int i=0; i < n; i++) vals[i] = getNext();
    return vals;
  }

  void reset(); // restart
  void setStart(int x); // set starting value
}
```