3. Interfaces :: Declaring a Java Interface

The mechanism by which Java enables us to do this is an **interface**. An interface is a programmer-defined **reference type** (a class is also a programmer-defined reference type) but an interface **is not** a class. In particular, an interface may only declare:

1. Abstract methods (no implementation).
2. Static methods (implemented).
3. Constants.

Of these, Item 1 is the most relevant to our discussion so we will limit ourselves to declaring interfaces containing abstract methods.

The syntax for declaring an interface is:

$[\textit{accessibility-modifier}]_{\text{opt}}$ **interface** *name* {
  $[\textit{abstract method declarations}]_{\text{opt}}$
  $[\textit{static method declarations}]_{\text{opt}}$
  $[\textit{constant declarations}]_{\text{opt}}$
}

And like a class, the code for an interface named *IFace* is written in a source code file named *IFace.java*. The *accessibility-modifier* can be **public**, **protected**, or **private**. In this course we will always use **public**.

3. Interfaces :: Java Interface Rules :: Part I

The additional rules for interfaces are:

1. All abstract (and static) methods are **public** by default so we omit the public accessibility modifier in front of the method name.

```
// SomeInterface.java
public interface SomeInterface {
  ~~public~~ void anAbstractMethod(int);
}
```

2. All constants are **public**, **static**, and **final** by default so we omit those modifiers as well:

```
// SomeInterface.java
public interface SomeInterface {
  void anAbstractMethod(int);
  ~~public static final~~ int SIZE = 100;
}
```

3. Interfaces :: Java Interface Rules :: Part I (continued)

3. An interface is a **reference type** (like a **class**) and the interface type may be used anywhere it is legal to use the name of any other reference type. This statement would create an *ArrayList* of objects, the type of each being *SomeInterface*:

   ```
   ArrayList<SomeInterface> a = new ArrayList<>();
   ```

   This statement declares an object variable named *i* which is of the type *SomeInterface*:

   ```
   SomeInterface i;
   ```

   An interface type may be used to specify the type of a method parameter:

   ```
   public void someMethod(SomeInterface pParam) { ... }
   ```

4. An interface is not a class, i.e., we cannot instantiate objects of the interface type:

   ```
   SomeInterface i = new SomeInterface();  // Syntax error
   ```

   An interface is very similar to an **abstract class**. Remember that we cannot instantiate objects of abstract classes either. There are some fundamental differences between abstract classes and interfaces, it may be helpful in the beginning to think of an interface as simply an abstract class that does not: (1) implement any methods; and (2) does not declare any instance variables.