**JAVA Notes-2nd**

MSc-IT (2nd Semester)

Standard College of Girls

Compiled By: Umair Yaqub

# Java – Interfaces

An interface is a reference type in Java, it is similar to class, it is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways:

* An interface can contain any number of methods.
* An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
* The byte code of an interface appears in a **.class** file.
* Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

* You cannot instantiate an interface.
* An interface does not contain any constructors.
* All of the methods in an interface are abstract.
* An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
* An interface is not extended by a class; it is implemented by a class.
* An interface can extend multiple interfaces.

**Declaring Interfaces:**

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface:

**Example:**

Below given is an example of an interface:

/\* File name : NameOfInterface.java \*/

import java.lang.\*;

//Any number of import statements

public interface NameOfInterface

{

 //Any number of final, static fields

 //Any number of abstract method declarations\

}

Interfaces have the following properties:

* An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
* Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
* Methods in an interface are implicitly public.

**Example:**

/\* File name : Animal.java \*/

interface Animal {

 public void eat();

 public void travel();

}

**Implementing Interfaces:**

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

/\* File name : MammalInt.java \*/

public class MammalInt implements Animal{

 public void eat(){

 System.out.println("Mammal eats");

 }

 public void travel(){

 System.out.println("Mammal travels");

 }

 public int noOfLegs(){

 return 0;

 }

 public static void main(String args[]){

 MammalInt m = new MammalInt();

 m.eat();

 m.travel();

 }

}

This would produce the following result:

ammal eats

ammal travels

When overriding methods defined in interfaces there are several rules to be followed:

* Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
* The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
* An implementation class itself can be abstract and if so interface methods need not be implemented.

When implementation interfaces there are several rules:

* A class can implement more than one interface at a time.
* A class can extend only one class, but implement many interfaces.
* An interface can extend another interface, similarly to the way that a class can extend another class.

# Java - Packages

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations ) providing access protection and name space management.

Some of the existing packages in Java are::

* **java.lang** - bundles the fundamental classes
* **java.io** - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.

**Creating a package:**

While creating a package, you should choose a name for the package and include a **package** statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

The **package** statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

To compile the Java programs with package statements you have to do use -d option as shown below.

javac -d **Destination\_folder file\_name**.java

Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder

**Example:**

Let us look at an example that creates a package called **animals**. It is a good practice to use names of packages with lower case letters to avoid any conflicts with the names of classes, interfaces.

Below given package example contains interface named *animals*:

/\* File name : Animal.java \*/

package animals;

interface Animal {

 public void eat();

 public void travel();

}

Now, let us implement the above interface in the same package *animals*:

package animals;

/\* File name : MammalInt.java \*/

public class MammalInt implements Animal{

 public void eat(){

 System.out.println("Mammal eats");

 }

 public void travel(){

 System.out.println("Mammal travels");

 }

 public int noOfLegs(){

 return 0;

 }

 public static void main(String args[]){

 MammalInt m = new MammalInt();

 m.eat();

 m.travel();

 }

}

Now compile the java files as shown below:

$ javac -d . Animal.java

$ javac -d . MammalInt.java

Now a package/folder with the name **animals** will be created in the current directory and these class files will be placed in it.

You can execute the class file with in the package and get the result as shown below.

$ java animals.MammalInt

ammal eats

ammal travels