INTERFACES

What Is an Interface?

As you've already learned, objects define their interaction with the outside world through the methods that they expose. Methods form the object's *interface* with the outside world; the buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off.

In its most common form, an interface is a group of related methods with empty bodies. A bicycle's behavior, if specified as an interface, might appear as follows:

```
interface Bicycle {

    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

To implement this interface, the name of your class would change (to a particular brand of bicycle, for example, such as ACMEBicycle), and you'd use the implements keyword in the class declaration:

```
class ACMEBicycle implements Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

   // The compiler will now require that methods
   // changeCadence, changeGear, speedUp, and applyBrakes
   // all be implemented. Compilation will fail if those
   // methods are missing from this class.
```

```
    void changeCadence(int newValue) {
        cadence = newValue;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }

    void printStates() {
        System.out.println("cadence:" +
            cadence + " speed:" +
            speed + " gear:" + gear);
    }
}
```

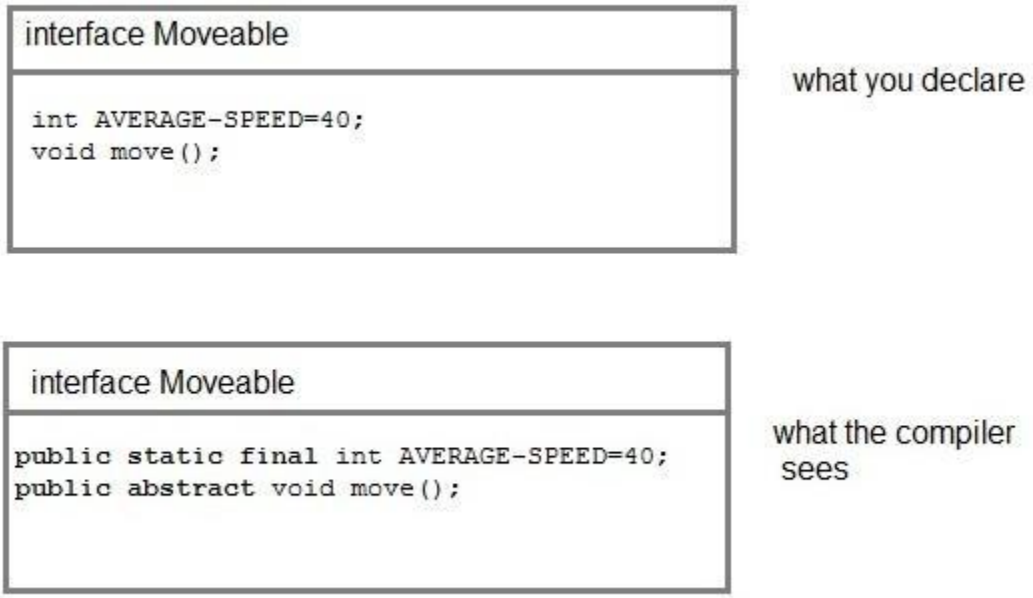
Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

Interface is a pure abstract class. They are syntactically similar to classes, but you cannot create instance of an **Interface** and their methods are declared without any body. Interface is used to achieve complete **abstraction** in Java. When you create an interface it defines what a class can do without saying anything about how the class will do it.

Syntax: interface interface_name { }

Example of Interface

```
interface Moveable {
        int AVERAGE-SPEED=40;
        void move();
}
```



NOTE : Compiler automatically converts methods of Interface as public and abstract, and the data members as public, static and final by default.

Rules for using Interface

- Methods inside Interface must not be static, final, native or strictfp.
- All variables declared inside interface are implicitly public static final variables(constants).
- All methods declared inside Java Interfaces are implicitly public and abstract, even if you don't use public or abstract keyword.
- Interface can extend one or more other interface.
- Interface cannot implement a class.
- Interface can be nested inside another interface.

Example of Interface implementation

```
interface Moveable{
 int AVG-SPEED = 40;
 void move();
}

class Vehicle implements Moveable{
 public void move() {
  System .out. print in ("Average speed is"+AVG-SPEED");
 }
 public static void main (String[] arg) {
  Vehicle vc = new Vehicle();
  vc.move();
 }
}
```

Output: Average speed is 40.

Interfaces supports Multiple Inheritance

Though classes in java doesn't support multiple inheritance, but a class can implement more than one interface.

```
interface Moveable {
 boolean isMoveable();
}

interface Rollable{
 boolean isRollable
}

class Tyre implements Moveable, Rollable {
 int width;

 boolean isMoveable() {
  return true;
 }

 boolean isRollable() {
  return true;
 }
 public static void main(String args[]) {
  Tyre tr=new Tyre();
  System.out.println(tr.isMoveable());
  System.out.println(tr.isRollable());
 }
}
```

Output:

```
true
true
```

Interface extends other Interface:

Classes implements interfaces, but an interface extends other interface.

```
interface NewsPaper{
 news();
}

interface Magazine extends NewsPaper{
 colorful();
}
```

Difference between an interface and an abstract class?

Abstract class	**Interface**
Abstract class is a class which contain one or more abstract methods, which has to be implemented by its sub classes.	Interface is a Java Object containing method declaration but no implementation. The classes which implement the Interfaces must provide the method definition for all the methods.
Abstract class is a Class prefix with an abstract keyword followed by Class definition.	Interface is a pure abstract class which starts with interface keyword.
Abstract class can also contain concrete methods.	Whereas, Interface contains all abstract methods and final variable declarations.
Abstract classes are useful in a situation that Some general methods should be implemented and specialization behavior should be implemented by child classes.	Interfaces are useful in a situation that all properties should be implemented.