

Interface

By

Dr. Tabbasum Naz

What is Interface

- **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.

Bicycle Interface

- A bicycle's behavior, if specified as an interface, might appear as follows:

```
interface Bicycle {  
    void changeGear(int newValue);  
    void speedUp(int increment);  
    void applyBrakes(int decrement);  
}
```

Implement Bicycle Interface

- 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 {  
    // remainder of this class  
    // implemented as before  
}
```

Interface (Real World Examples)

Interfaces define and standardize the ways in which things such as people and systems can interact with one another. For example, the controls on a radio serve as an interface between radio users and a radio's internal components. The controls allow users to perform only a limited set of operations (e.g., changing the station, adjusting the volume, choosing between AM and FM), and different radios may implement the controls in different ways (e.g., using push buttons, dials, voice commands). The interface specifies *what* operations a radio must permit users to perform but does not specify *how* the operations are performed. Similarly, the interface between a driver and a car with a manual transmission includes the steering wheel, the gear shift, the clutch pedal, the gas pedal and the brake pedal. This same interface is found in nearly all manual transmission cars, enabling someone who knows how to drive one particular manual transmission car to drive just about any manual transmission car. The components of each individual car may look different, but their general purpose is the same—to allow people to drive the car.

Software objects also communicate via interfaces. A Java interface describes a set of methods that can be called on an object, to tell the object to perform some task or return some piece of information.

Implementing an Interface (contd..)

- 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.

Implementing an Interface

To use an interface, a concrete class must specify that it **implements** the interface and must declare each method in the interface with the signature specified in the interface declaration. A class that does not implement all the methods of the interface is an abstract class and must be declared **abstract**. Implementing an interface is like signing a contract with the compiler that states, “I will declare all the methods specified by the interface or I will declare my class **abstract**.”



Common Programming Error 10.6

Failing to implement any method of an interface in a concrete class that implements the interface results in a compilation error indicating that the class must be declared abstract.

Shape Interface Example

```
public class Main {  
  
    public static void main(String[] args) {  
        shape circleshape=new circle();  
        circleshape.Draw();  
    }  
}  
  
interface shape  
{  
    public String baseClass="shape";  
    public void Draw();  
}  
  
class circle implements shape  
{  
    public void Draw() {  
        System.out.println("Drawing Circle here");  
    }  
}
```


Car Interface Example

```
public interface OperateCar {  
  
    // constant declarations, if any  
  
    // method signatures  
  
    // An enum with values RIGHT, LEFT  
    int turn(Direction direction,  
             double radius,  
             double startSpeed,  
             double endSpeed);  
    int changeLanes(Direction direction,  
                   double startSpeed,  
                   double endSpeed);  
    int signalTurn(Direction direction,  
                  boolean signalOn);  
    int getRadarFront(double distanceToCar,  
                    double speedOfCar);  
    int getRadarRear(double distanceToCar,  
                   double speedOfCar);  
    .....  
    // more method signatures  
}
```

Car Implementation

```
public class OperateBMW760i implements OperateCar {

    // the OperateCar method signatures, with implementation --
    // for example:
    int signalTurn(Direction direction, boolean signalOn) {
        // code to turn BMW's LEFT turn indicator lights on
        // code to turn BMW's LEFT turn indicator lights off
        // code to turn BMW's RIGHT turn indicator lights on
        // code to turn BMW's RIGHT turn indicator lights off
    }

    // other members, as needed -- for example, helper classes not
    // visible to clients of the interface
}
```

Interface Payable Example

- Example: pg (525) PDF 538

Static Keyword

- static variables
- static methods

static variable

- It is a variable which **belongs to the class** and **not to object**(instance)
- Static variables are **initialized only once** , at the start of the execution . These variables will be initialized first, before the initialization of any instance variables
- A **single copy** to be shared by all instances of the class
- A static variable can be **accessed directly** by the **class name** and doesn't need any object
- Syntax : ***<class-name>.<variable-name>***

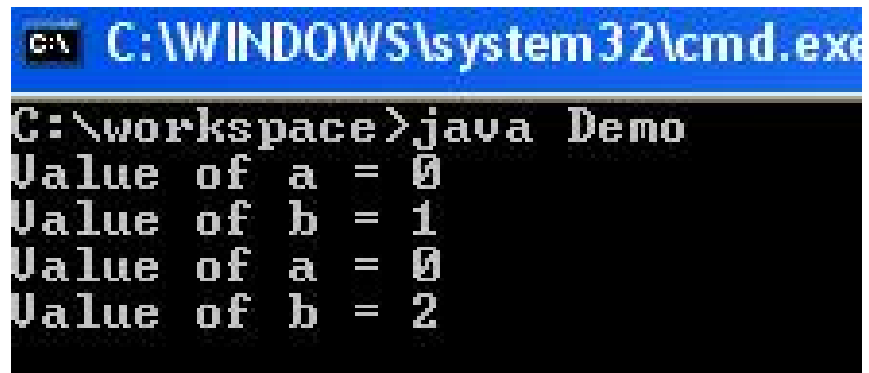
static method

- It is a method which **belongs to the class** and **not** to the **object**(instance)
- A static method **can access only static data**. It can not access non-static data (instance variables)
- A static method **can call only** other **static methods** and can not call a non-static method from it.
- A static method can be **accessed directly** by the **class name** and doesn't need any object
- Syntax : **<class-name>.<method-name>**
- A static method cannot refer to "this" or "super" keywords in anyway

Static Example

```
01. class Student {
02.     int a; //initialized to zero
03.     static int b; //initialized to zero only when class is loaded not for each object created.
04.
05.     Student(){
06.         //Constructor incrementing static variable b
07.         b++;
08.     }
09.
10.
11.     public void showData(){
12.         System.out.println("Value of a = "+a);
13.         System.out.println("Value of b = "+b);
14.     }
15.     //public static void increment(){
16.     //a++;
17.     //}
18.
19. }
20.
21. class Demo{
22.     public static void main(String args[]){
23.         Student s1 = new Student();
24.         s1.showData();
25.         Student s2 = new Student();
26.         s2.showData();
27.         //Student.b++;
28.         //s1.showData();
29.     }
30. }
```

Static Example Output



```
C:\WINDOWS\system32\cmd.exe
C:\workspace>java Demo
Value of a = 0
Value of b = 1
Value of a = 0
Value of b = 2
```

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the command "C:\workspace>java Demo" being executed. The output of the program is displayed on the following four lines: "Value of a = 0", "Value of b = 1", "Value of a = 0", and "Value of b = 2".

Thank You