

## INTERFACE

OOPS USING JAVA

1

## Where and why we should use **interface**

Let see a scenario:

- ❖ All branches of a bank must have **some common facilities** that they **extend to their customers**. These facilities include **opening a Savings account and issuing a Vehicle Loan, House Loan, and Gold Loan**.
- ❖ So the head office has decided to create a **contract** that **defines the facilities that every bank branch must implement**.

OOPS USING JAVA

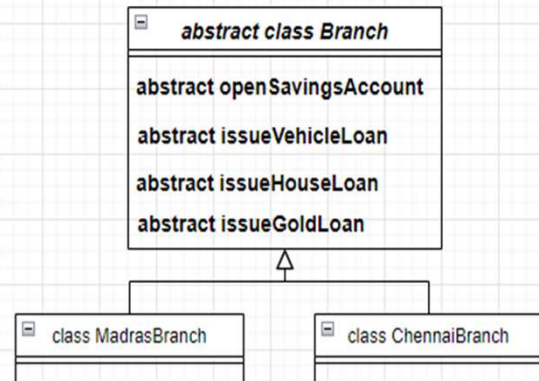
2

## Where and why we should use **interface**

based on the scenario we can solve by:

We will create an abstract class Branch and define **four abstract methods**:

- Opening savings account
- Issuing Vehicle Loan
- Issuing House Loan
- Issuing Gold Loan



Then the concrete subclass of the Branch will **override all the methods and give their own implementation**.

## Where and why we should use **interface**

That's ok

But after some time:

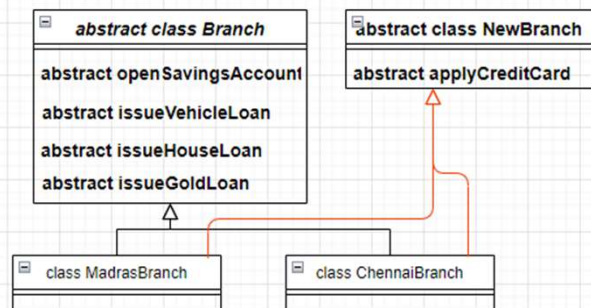
**The bank** has decided to **add more facilities** like issuing a **credit card** to the customers and decided to implement it across all the branches. How to achieve it?

## Where and why we should use **interface**

We will **create a new abstract class** and **define a new abstract method** i.e **issuing the credit card** and the concrete **subclass** of the **Branch** will be the **child of the new abstract class** also.

Is it possible in **Java**?

**NO**

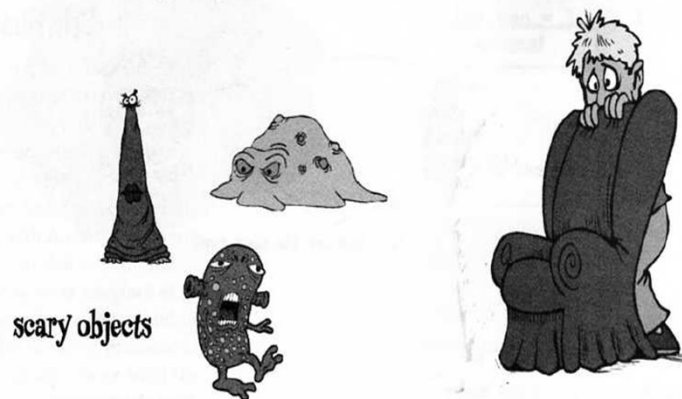


## Interface

**Interface** contains **method prototypes** (body less methods) and **constants**.

When you create an interface, you're defining a **contract** for **what a class can do**, without saying anything about **how the class will do it**.

**What does a new Animal() object look like?**



## What is an **Interface**

---

- ❖ A programmer's tool for **specifying certain behaviors** that an object must have in order to be useful for some particular task.
- ❖ An interface in Java is a **blueprint of a class**.
- ❖ **Can contain only** constants (final variables) and abstract method (no implementation).
- ❖ Use when a number of classes **share a common interface**.
- ❖ Each class **should implement** the interface.

## What is an **Interface**

---

- ❖ Java Interface also **represents the IS-A relationship**.
- ❖ Since Java 8, we can have **default** and **static methods** in an interface.
- ❖ Since Java 9, we can have **private methods** in an interface.

## Properties of an Interface

- ❖ An interface defines a contract for a class.
- ❖ Objects can't be created for interface.
- ❖ In an interface, all methods are **implicitly public and abstract** and variables are **implicitly public, static, and final**.
- ❖ The class which implements the interface has to provide definitions for all abstract methods.
- ❖ If at least one abstract method of the interface has not been overridden by the class that implemented the interface, make it abstract.
- ❖ Inheritance is possible in an interface and it supports multiple inheritance.

## Interface Example

For example, you might specify Driver interface as part of a Vehicle object hierarchy.

- A human can be a Driver, but so can a Robot.



## Interface Declaration

```
public interface Driver
{
    // Go Wheel (double amount) Declaration
    void turnWheel(double amount);
    // Methods Declaration – only method body
    void pressBrake(double amount);
}
```

Syntax for Interface  
Example  
Declaration

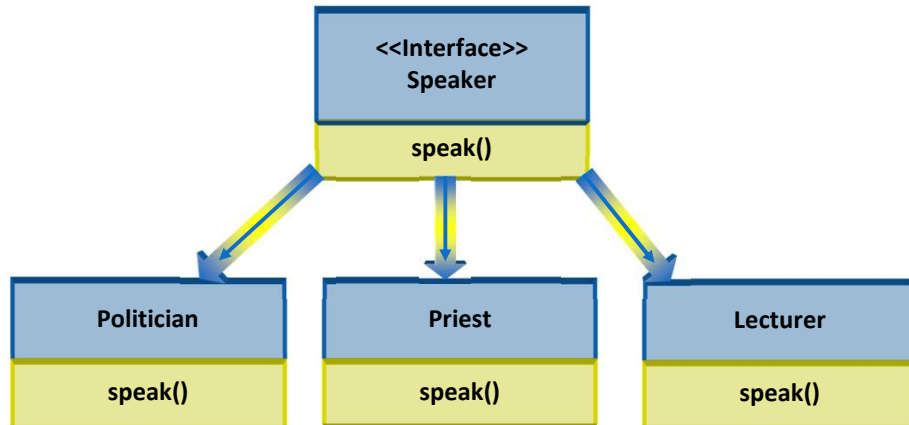
## Interface Implementation

- ❖ Interfaces are used like super-classes who properties are inherited by classes. This is achieved by creating a class that implements the given interface as follows:

Syntax for Interface  
Example  
Implementation

```
class ClassName implements Interface1, Interface2, ..., InterfaceN
{
    public class BusDriver extends Person implements Driver
    {
        // Body of Class
        // include each of the two methods from Driver
    }
}
```

## Other Example of Interface



OOPS USING JAVA

13

## Interface Example

```

interface Animal
{
    int LEGS=4; //public static final LEGS=4;
    void eat(); //public abstract void eat();
    void sound(); //public abstract void sound();
}
class Dog implements Animal
{
    public void eat()
    {
        System.out.println("EAT MEAT ");
    }
    public void sound()
    {
        System.out.println("BARK ");
    }
}
  
```



class Cow implements Animal

```

{
    public void eat() {
        System.out.println("EAT GRASS ");
    }
    public void sound() {
        System.out.println("MOO ");
    }
}
  
```

The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

OOPS USING JAVA

14

## Fields w.r.t interfaces

By default, all interface fields are **public static final**.

Hence, All of the following field declarations are equal:

- a) **public static final** int SIZE = 10; //Declaring public static final explicitly is redundant but not error.
- b) static final int SIZE = 10;
- c) final int SIZE = 10;
- d) int SIZE = 10;

All interface variables must be initialized in the declaration section.

**Example:**

Public static final int SIZE = **10**; //declaration cum initialization

Public static final int SIZE; //C.E saying Missing Initialization

## Methods w.r.t interfaces

By default, all interface methods are **public abstract i.e.**, interface is a pure abstract class.

Hence, All of the following methods declarations are equal:

- a) **public abstract** void meth(); //Declaring public abstract explicitly is redundant but not error.
- b) public void meth();
- c) abstract void meth();
- d) void meth();

All methods in an interface are abstract, hence, interfaces cannot be instantiated. However, interfaces can be used as reference variables to achieve dynamic polymorphism.



## Constructors w.r.t **interfaces**

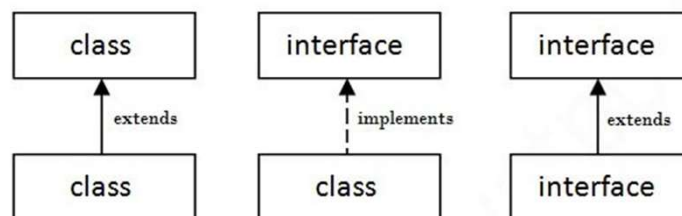
All variables in an interface are final constants, which must be initialized in the declaration section itself.

Also, all methods in an interface are abstract, such an interfaces cannot be instantiated.

Hence, Interface does not need constructors at all i.e., **interfaces never have constructors at all.**

## The relationship between **classes** and **interfaces**

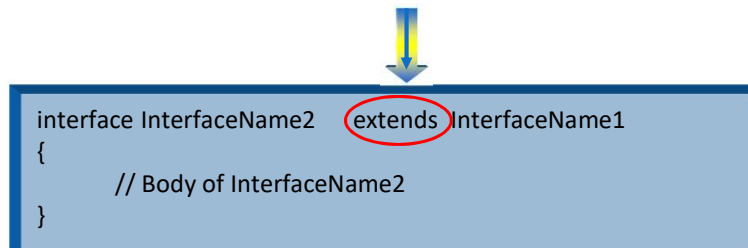
A class extends another class, an interface extends another interface, but a **class implements an interface.**



## Inheriting interfaces

Like classes, interfaces can also be extended. The new sub-interface will inherit all the members of the super-interface in the manner similar to classes.

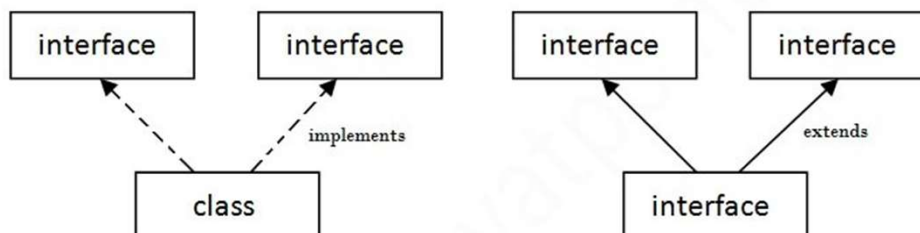
This is achieved by using the extends keyword.



```
interface InterfaceName2 extends InterfaceName1
{
    // Body of InterfaceName2
}
```

## Multiple inheritance in Java by **interface**

If a class implements **multiple interfaces**, or an interface extends multiple interfaces, it is known as **multiple inheritance**.



**Multiple Inheritance in Java**

## Test Your Self

1) What will be the output of the following code?

```
interface X
{
    void show();
}

class Y extends X
{
    public void show()
    {
        System.out.println("Inside Y Show ")
    }
}
```

## Test Your Self

2) what will be the output of the following code?

```
interface X
{
    void show();
}

class Y implements X
{
    void show()
    {
        System.out.println("Inside Y Show ");
    }
}
```

## Test Your Self

3) what will be the output of the following code?

```
interface X
{
    void show();
}
interface Y implements X
{
    void disp();
}

class Z extends Y
{
    public void show()
    {
        System.out.println(" Inside show ");
    }
}
```