



# CS205 Object Oriented Programming in Java

## Module 3 - More features of Java (Part 2)

**Prepared by**

**Renetha J.B.**

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

# Topics



- Introduction:
  - ☑ **Interfaces.**

# Interface



- Interface can be created using the keyword **interface**.
- Interfaces are syntactically similar to classes.
- Interface **does not have instance variables**.
- The **methods** in interface are declared without any body.
  - **Interface** never implements methods.
- Any number of classes can **implement** an **interface**.
- One class can **implement** any number of interfaces.
  - This helps to achieve multiple inheritance.

# Interface(contd.)



- To implement an interface,
  - a class must create and define the complete set of methods that are declared by the interface.
- Each class can have its own implementation of the methods.
- By providing the interface keyword, Java allows you to fully utilize the “one interface, multiple methods” aspect of polymorphism.
- Interfaces support **dynamic method resolution at run time**.

# Interface(contd.)



- General form of an interface:

```
accessspecifier interface name {  
    return-type method-name1(parameter-list);  
    return-type method-name2(parameter-list);  
    type final-varname1 = value;  
    type final-varname2 = value;  
    // ...  
    return-type method-nameN(parameter-list);  
    type final-varnameN = value;  
}
```

# Interface(contd.)



- When **no access specifier** is included, then it has **default** access.
  - the interface is only available to other members of the package in which it is declared.
- The **methods** are declared have **no bodies**. They end with a semicolon after the parameter list.
- They are **abstract methods**.
- Each class that includes an interface must implement all of the methods.
- Variables are implicitly **final** and **static**, meaning they cannot be changed by the implementing class.
  - They must also be initialized.
- All methods and variables are implicitly **public**

# Example



```
interface Callback {  
    void show(int param);  
}
```

# Implementing Interfaces



- After an interface has been defined, one or more classes can implement that interface.
  - For that include the **implements** clause in a class definition
- General form of a class that includes the **implements** clause  
`class classname [extends superclass] [implements interface [,interface...]]`  
`{`  
`// class-body`  
`}`  
`//square bracket denotes optional`
- If a class implements more than one interface, the interfaces are separated with a comma
- When we implement an interface method, it must be declared as **public**.





```
interface Callback
{
void show(int param);
}

class Sample implements Callback
{
public void show(int p)
{
    System.out.println("show p= " + p);
}

//other methods
}
```

Here **Callback** is an interface The class Sample implements that interface. So **Sample** class should define the method in **Callback** , show (int param)

# Accessing Implementations Through Interface References



- We can declare variables as **object references** that use an interface rather than a class type.
  - Any instance of any class that implements the declared interface can be referred to by such a variable
- interfacename obj=object of implementing class;



```
interface Callback
{
    void show(int param);
}
class Sample implements Callback
{
    public void show(int p)
    {
        System.out.println("show p= " + p);
    }
    //other methods
}
```

```
class Test{
    public static void main(String args[])
    {
        Callback c = new Sample();
        c.show(42);
    }
}
```

Here c is an interface reference variable .It has only has knowledge of the methods declared by its **interface declaration**.

# Partial Implementations



- If a class includes an interface but **does not fully implement the methods required by that interface**, then that **class** must be declared as **abstract**.

```
interface Callback {  
    void show(int param);  
}
```

```
abstract class Incomplete implements Callback {  
    int a, b;  
    void display()  
    {  
        System.out.println("display");  
    }  
}
```

>>Here the class Incomplete does not implement **show()** in the **interface** **Callback**. So the class Incomplete is **abstract** class

# Nested Interfaces



- An interface can be declared a member of a class or another interface. Such an interface is called a **member interface** or a **nested interface**.
- A nested interface can be declared as public, private, or protected.
  - The top level interface must either be declared as **public** or use the **default** access level.

# Nested Interfaces



- If we want to use a *nested interface outside of its enclosing scope*, the nested interface must be qualified by the name of the class or interface of which it is a member.

# Nested Interfaces



```
class A {  
    // this is a nested interface  
    public interface NestedIF  
    {  
        boolean isNotNeg(int x);  
    }  
}  
class B implements A.NestedIF {  
    public boolean isNotNeg (int x)  
    {  
        return x < 0 ? false: true;  
    }  
}
```

```
class NestedIFDemo {  
    public static void main(String args[])  
    {  
        A.NestedIF nif = new B();  
        if(nif.isNotNeg(10))  
            System.out.println("10 is not negative");  
    }  
}
```

# Applying Interfaces





# Variables in Interfaces



- When we include an interface in a class (using “implement” the interface), all of those **variable** names in the interface will be in scope as **constants**.
  - That is they are imported to class name space as **final** variables.
-

## INTERFACES Can Be Extended



```
import java.util.Random;

interface Interf {
    int NO = 0;
    int YES = 1;
}

class Question implements Interf
{
    Random rand = new Random();
    int ask() {
        int prob = (int) (100 * rand.nextDouble());
        if (prob < 50)
            return NO; // 30%
        else
            return YES;
    }
}
```

```
class AskMe implements Interf
{
    static void answer(int result) {
        switch(result) {
            case NO:
                System.out.println("No");
                break;
            case YES:
                System.out.println("Yes");
                break; } }
    public static void main(String args[])
    {
        Question q = new Question();
        answer(q.ask()); }
}
```

# Interfaces Can Be Extended



- One interface can inherit another by use of the keyword **extends**.
- When a class *class1* implements an interface *interface1* that inherits another interface *interface2*, then *class1* must provide implementations for all methods defined within the interface inheritance chain(both *interface1* and *interface2*).



// One interface can extend another-E.g.

```
.  
interface A {  
    void meth1();  
    void meth2();  
}  
interface B extends A {  
    void meth3();  
}  
class MyClass implements B  
{  
    public void meth1()  
    {  
        System.out.println("Implement  
            meth1().");  
    }  
}
```

```
    public void meth2() {  
        System.out.println("Implement meth2().");  
    }  
    public void meth3() {  
        System.out.println("Implement meth3().");}  
}
```

```
class IFExtend {  
    public static void main(String arg[])  
    {  
        MyClass ob = new MyClass();  
        ob.meth1();  
        ob.meth2();  
        ob.meth3();  
    }  
}
```

# Reference



- **Herbert Schildt, Java: The Complete Reference, 8/e, Tata McGraw Hill, 2011.**