Course: Object Based Modeling
Code: CS-33105
Branch: MCA-3

Lecture 11: Interfaces

Dr. J Sathish Kumar (JSK)
(Faculty & Coordinator)

Department of Computer Science and Engineering

Motilal Nehru National Institute of Technology Allahabad, Prayagraj-211004

Interfaces

- Using the keyword **interface**, you can fully abstract a class’ interface from its implementation.
- That is, using **interface**, you can specify what a class must do, but not how it does it. Interfaces are syntactically similar to classes, but they lack instance variables, and, as a general rule, their methods are declared without any body.
- In practice, this means that you can define interfaces that don’t make assumptions about how they are implemented.
- Once it is defined, any number of classes can implement an **interface**.
- Also, one class can implement any number of interfaces.
- To implement an interface, a class must provide the complete set of methods required by the interface.
- However, each class is free to determine the details of its own implementation.
- By providing the **interface** keyword, Java allows you to fully utilize the “one interface, multiple methods” aspect of polymorphism.

Defining an Interface

- An interface is defined much like a class. This is a simplified general form of an interface:

```
access 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;
}
```

Implementing Interfaces

- Once an **interface** has been defined, one or more classes can implement that interface.
- To implement an interface, include the **implements** clause in a class definition, and then create the methods required by the interface.
- The general form of a class that includes the **implements** clause looks like this:

 class *classname* [extends *superclass*] [implements *interface* [,*interface*...]]

```
{
        // class-body
}
```

Example #1

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

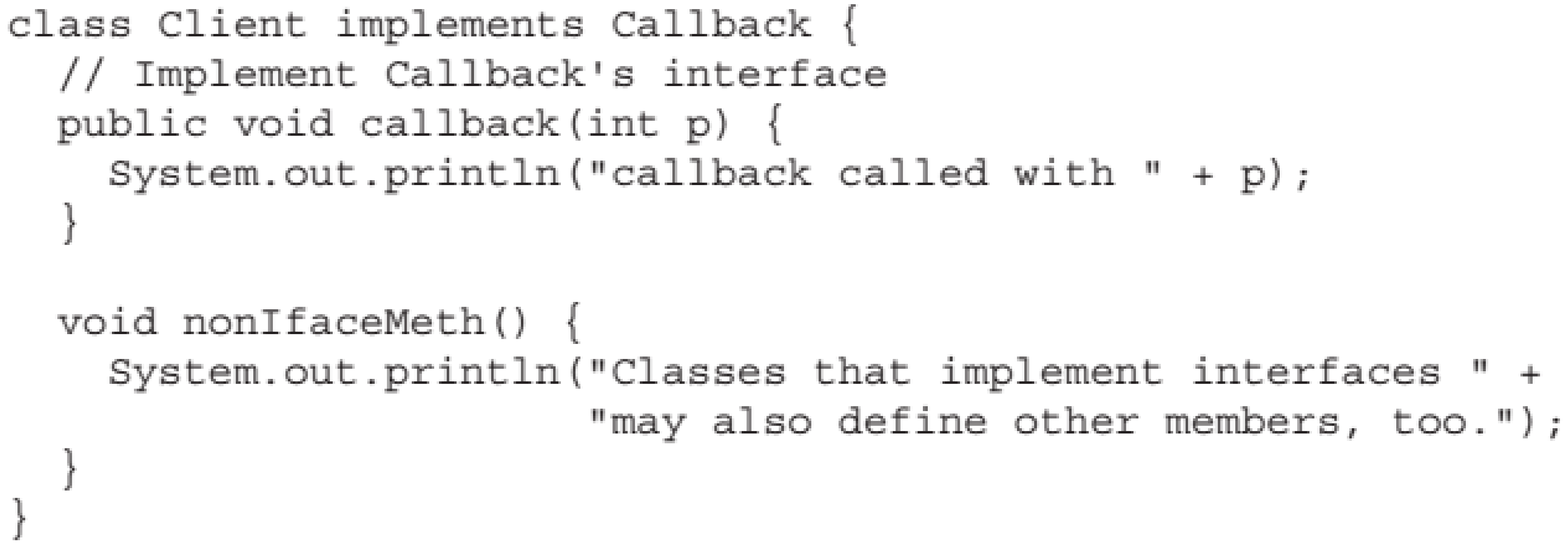
```
class Client implements Callback {
  // Implement Callback's interface

public void callback(int p) {

  System.out.println("callback called with " + p);
 }
}
```

```
class TestIface {
  public static void main(String args[]) {
    Callback c = new Client();
    c.callback(42);
  }
}
```

The output of this program is shown here:

```
callback called with 42
```

Example #2



```
class Client implements Callback {
  // Implement Callback's interface
  public void callback(int p) {
    System.out.println("callback called with " + p);
  }

  void nonIfaceMeth() {
    System.out.println("Classes that implement interfaces " +
                       "may also define other members, too.");
  }
}
```

Example #3

```
// Another implementation of Callback.
class AnotherClient implements Callback {
  // Implement Callback's interface
  public void callback(int p) {
    System.out.println("Another version of callback");
    System.out.println("p squared is " + (p*p));
  }
}
```

Now, try the following class:

```
class TestIface2 {
  public static void main(String args[]) {
    Callback c = new Client();
    AnotherClient ob = new AnotherClient();

    c.callback(42);

    c = ob; // c now refers to AnotherClient object
    c.callback(42);
  }
}
```

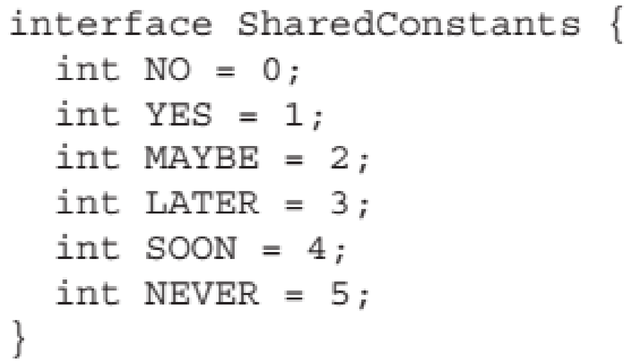
The output from this program is shown here:

```
callback called with 42
Another version of callback
p squared is 1764
```

Nested Interfaces

- 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**.
- Applying Interfaces best example is stack – Refer the example in Core java
- Variables in Interfaces
  - This is similar to using a header file in C/C++ to create a large number

  of **#defined** constants or **const** declarations



```
interface SharedConstants {
  int NO = 0;
  int YES = 1;
  int MAYBE = 2;
  int LATER = 3;
  int SOON = 4;
  int NEVER = 5;
}
```

Nested Interfaces

```
// A nested interface example.

// This class contains a member interface.
class A {
  // this is a nested interface
  public interface NestedIF {
    boolean isNotNegative(int x);
  }
}

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

```
class NestedIFDemo {
  public static void main(String args[]) {

    // use a nested interface reference
    A.NestedIF nif = new B();

    if(nif.isNotNegative(10))
      System.out.println("10 is not negative");
    if(nif.isNotNegative(-12))
      System.out.println("this won't be displayed");
  }
}
```

Example #4

Notice that A defines a member interface called NestedIF and that it is declared public.
Next, B implements the nested interface by specifying implements A.NestedIF

Example #5

Interfaces Can Be Extended

- One interface can inherit another by use of the keyword **extends**.

```
// One interface can extend another.
interface A {
  void meth1();
  void meth2();
}

// B now includes meth1() and meth2() -- it adds me
interface B extends A {
  void meth3();
}

// This class must implement all of A and B
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();
          }
        }
```