Web Technology 9
Interfaces

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Interface

Interface

- An interface is similar to a class. It can have variables and method signatures; it cannot have any method implementation
- Using interface, we specify what a class must do, but not how it does this

```
access interface name {
                type method-name1(parameter-list);
                type method-name2(parameter-list);
                        ...
                final type var-name1 = value1;
                final type var-nameM = valueM;
                        ...
                }
```

Interface...

Interface...

- Two types of access:
  - **public** - interface may be used anywhere in a program
  - **default** - interface may be used in the current package only
- Interface methods have no bodies - they end with the semicolon after the parameter list. They are essentially abstract methods
- An interface may include variables, but they must be final, static and initialized with a constant value
- *In a public interface, all members are implicitly public*

Interface Implementation

Interface Implementation

- A class implements an interface if it provides a complete set of methods defined by this interface:
  - any number of classes may implement an interface
  - one class may implement any number of interfaces
- Each class is free to determine the details of its implementation
- General format of a class that includes the implements clause:

```
access class className extends superClassName
implements interface1, interface2, ..., interfaceN {
...
}
```

Access is public or default

Interface Implementation...

Interface Implementation...

- If a class implements several interfaces, they are separated with a comma
- If a class implements two interfaces that declare the same method, the same method will be used by the clients of either interface
- The methods that implement an interface must be declared public
- The type signature of the implementing method must match exactly the type signature specified in the interface definition
- *A class implementing an interface must provide a definition for each method in the interface or itself be declared as abstract*

Interface Implementation...

- Declaration of the *Callback* interface:

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

- *Client* class implements the Callback interface:

```
class Client implements Callback {
            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.");
                        }
            }
```

Interface as a Type

Interface as a Type

- Variable may be declared with interface as its type:

  interface MyInterface {...}
  ...
  MyInterface mi;

- The variable of an interface type may reference an object of any class that implements this interface:

  class MyClass implements MyInterface {...}
  MyInterface mi = new MyClass();

Interface as a Type...

Interface as a Type...

- Using the interface type variable, we can call any method in the interface:

```
interface MyInterface {
void myMethod(...);
...
}

class MyClass implements MyInterface {...}
...
MyInterface mi = new MyClass();
...
mi.myMethod();
```

- The correct version of the method will be called based on the actual instance of the interface being referred to

Interface as a Type...

```
interface Callback {
                void callback(int p);
                }
        class Client implements Callback {
                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);
                }       }
```

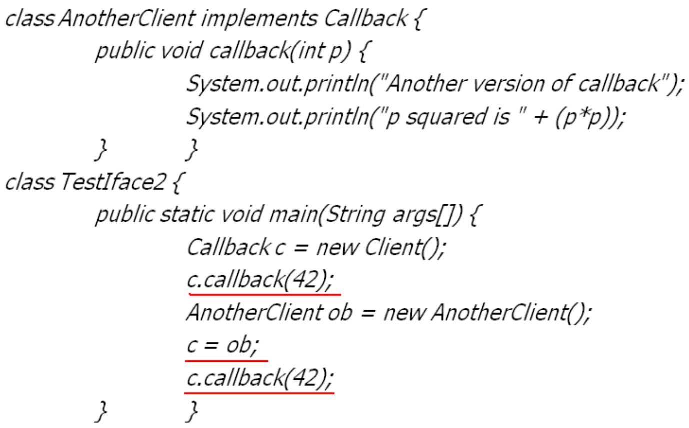
Interface as a Type...

- Call through an interface variable is one of the key features of interfaces:
    - the method to be executed is looked up at run-time
    - the calling code can dispatch through an interface without having to know anything about the callee

Interface as a Type...

Interface as a Type...

- Allows classes to be created later than the code that calls methods on them



```
class AnotherClient implements Callback {
        public void callback(int p) {
                System.out.println("Another version of callback");
                System.out.println("p squared is " + (p*p));
        }       }
class TestIface2 {
        public static void main(String args[]) {
                Callback c = new Client();
                c.callback(42);
                AnotherClient ob = new AnotherClient();
                c = ob;
                c.callback(42);
        }       }
```

Binding

Binding

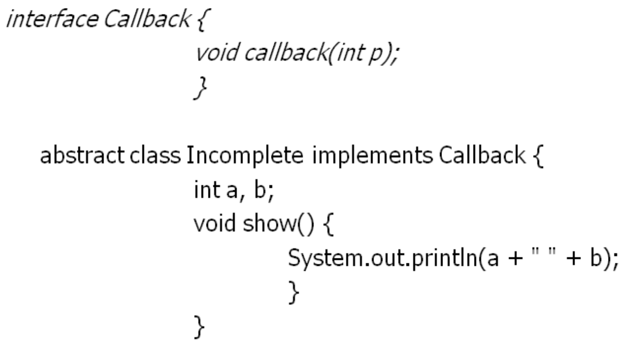
- **Compile-Time Method Binding**:
  - Normally, in order for a method to be called from one class to another, both classes must be present at compile time
  - This implies:
    - a static, non-extensible classing environment
    - functionality gets pushed higher and higher in the class hierarchy to make them available to more sub-classes

- **Run-Time Method Binding**:
  - Interfaces support dynamic method binding
  - Interface disconnects the method definition from the inheritance hierarchy:
    - interfaces are in a different hierarchy from classes
    - it is possible for classes that are unrelated in terms of the class hierarchy to implement the same interface

Interface and Abstract Class

Interface and Abstract Class

- A class that claims to implement an interface but does not implement all its methods must be declared abstract



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

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

Interface Variables

Interface Variables

- *Variables declared in an interface must be constants*
- A technique to import shared constants into multiple classes:
  - declare an interface with variables initialized to the desired values
  - include that interface in a class through implementation
- As no methods are included in the interface, the class does not implement anything except importing the variables as constants

Interface Variables...

```
import java.util.Random;
        interface SharedConstants {
                int NO = 0;
                int YES = 1;
                int MAYBE = 2;
                int LATER = 3;
                int SOON = 4;
                int NEVER = 5;
                }
class Question implements SharedConstants {
                Random rand = new Random();
                int ask() {
                        int prob = (int) (100 * rand.nextDouble());
                        if (prob < 30)  return NO;
                        else if (prob < 60) return YES;
                        else if (prob < 75) return LATER;
                        else if (prob < 98) return SOON;
                        else return NEVER;
                        }
                }
```

Interface Variables...

```
class AskMe implements SharedConstants {
        static void answer(int result) {
                switch(result) {
                        case NO:  System.out.println("No"); break;
                        case YES:  System.out.println("Yes"); break;
                        case MAYBE: System.out.println("Maybe"); break;
                        case LATER: System.out.println("Later"); break;
                        case SOON:  System.out.println("Soon"); break;
                        case NEVER: System.out.println("Never"); break;
                }       }
        public static void main(String args[]) {
                Question q = new Question();
                answer(q.ask());
                answer(q.ask());
                answer(q.ask());
                answer(q.ask());
                }
        }
```

Interface Inheritance

Interface Inheritance

- One interface may inherit another interface

```
interface MyInterface1 {
void myMethod1(...);
}
interface MyInterface2 extends MyInterface1 {
void myMethod2(...);
}
```

- ***When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain***

```
class MyClass implements MyInterface2 {
void myMethod1(...) {...}
void myMethod1(...) {...}
...
}
```

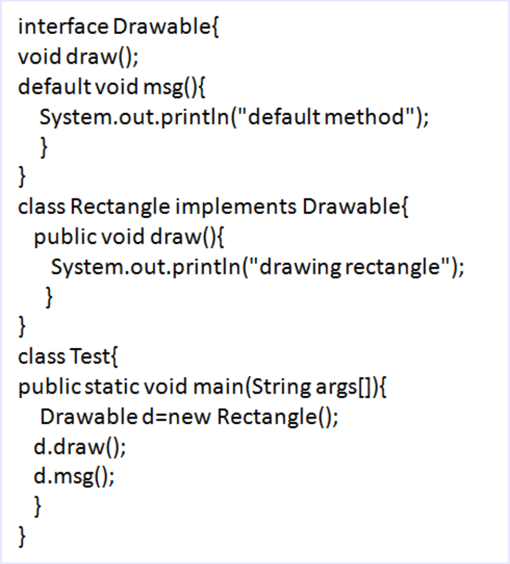
Interface Inheritance...

```
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();
                }
        }
```

Default Method in Interface

Default Method in Interface

Since Java 8, we can have default method in interface

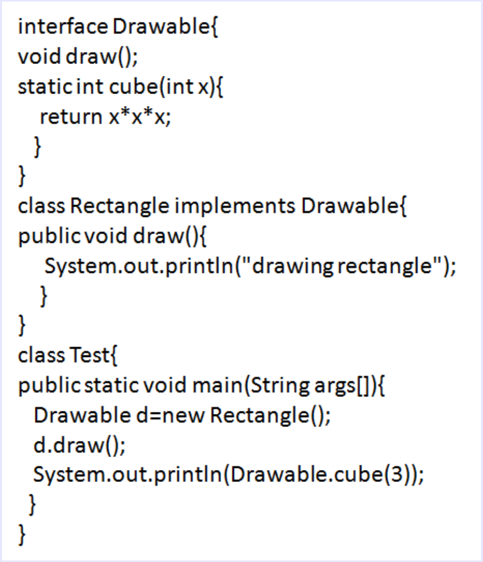


```
interface Drawable{
void draw();
default void msg(){
   System.out.println("default method");
   }
}
class Rectangle implements Drawable{
  public void draw(){
    System.out.println("drawing rectangle");
   }
}
class Test{
public static void main(String args[]){
   Drawable d=new Rectangle();
  d.draw();
  d.msg();
  }
}
```

Static Method in Interface

Static Method in Interface

Since Java 8, we can have static method in interface



```
interface Drawable{
void draw();
static int cube(int x){
   return x*x*x;
  }
}
class Rectangle implements Drawable{
public void draw(){
    System.out.println("drawing rectangle");
   }
}
class Test{
public static void main(String args[]){
  Drawable d=new Rectangle();
  d.draw();
  System.out.println(Drawable.cube(3));
 }
}
```