Interface

- In Java, only single inheritance is permitted. However, Java provides a construct called an interface which can be implemented by a class.
- Using the keyword interface, you can fully abstract a class' interface from its implementation.
- Using interface, we specify what a class must do, but not how it does this.
- An interface is syntactically similar to a class, but it lacks instance variables and its methods are declared without any body.
- An interface is defined with an interface keyword.

Interface

- Interfaces are similar to abstract classes.
- In effect using interfaces gives us the benefit of multiple inheritance without many of it's problems.
- Interfaces are compiled into bytecode just like classes.
- Interfaces cannot be instantiated.
- Interfaces can contain only abstract methods and constants.

Interface Vs Abstract Class

	Interface	Abstract class
Fields	Only constants	Constants and variable data
Methods	No implementation allowed (no abstract modifier necessary)	Abstract or concrete
Inheritance	A subclass can implement many interfaces	A subclass can inherit only one class

Interface Format

- General format:
- [access – Specifier] interface interfacename {

  type method-name1(parameter-list);

  type method-name2(parameter-list);

  …

  type var-name1 = value1;

  type var-nameM = valueM;

  …
- }

Interface Comments

- Two types of access:

1) public – interface may be used anywhere in a program
2) 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 Comments

- Some hints about interface

1. Interface method should be public and abstract.
2. Interface fields should be public and final.
3. Use the Keyword interface to define an interface.
4. If you define a public interface with name myInterface the java file should be named as myInterface.java (Similar to public class definition rules).
5. A class implementing an interface should use the keyword implements.
6. No objects can be created from an interface.
7. Interfaces don't have constructors as they can't be initiated
8. An Interface can extends one or more interfaces.
9. You can define a reference of type interface but you should assign to it an object instance of class type which implements that interface.

Interface Implementation

- A class implements an interface if it provides a complete set of methods defined by this interface.

1) any number of classes may implement an interface

2) one class may implement any number of interfaces

- Each class is free to determine the details of its implementation.
- Implementation relation is written with the implements keyword.

Implementation Format

- General format of a class that includes the implements clause:
- access class name extends super-class implements interface1, interface2, …, interfaceN {

  …

  }
- Access is public or default.

Interface and Class

- An interface is similar to a class in the following ways:
- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The bytecode of an interface appears in a .class file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

Interface and Class

- an interface is different from a class in several ways, including:
- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Implementation Comments

- 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.

Example: Interface

- Declaration of the Callback interface:
- interface Callback {

```
    void callback(int param);
    }
```

- Client class implements the Callback interface:
- class Client implements Callback {

```
    public void callback(int p) {
        System.out.println("callback called with " + p);
    }
```

- }

More Methods in Implementation

- An implementing class may also declare its own methods:
- 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

- 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();

Call Through Interface Variable

- 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.

Example: Call Through Interface

- Declaration of the Callback interface:
- interface Callback {

```
    void callback(int param);
```

- }
- Client class implements the Callback interface:
- class Client implements Callback {

```
    public void callback(int p) {
    System.out.println("callback called with " + p);
    }
```

- }

Example: Call Through Interface

- TestIface declares the Callback interface variable, initializes it with the new Client object, and calls the callback method through this variable:
- class TestIface {

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

- }

Call Through Interface Variable

- Call through an interface variable is one of the key features of interfaces:

1) the method to be executed is looked up dynamically at run-time

2) the calling code can dispatch through an interface without having to know anything about the callee

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

Example: Interface Call

- Another implementation of the Callback interface:
- class AnotherClient implements Callback {

```
    public void callback(int p) {
    System.out.println("Another version of
                                callback");
    System.out.println("p squared is " + (p*p));
    }
```

- }

Example: Interface Call

- Callback variable c is assigned Client and later AnotherClient objects and the corresponding callback is invoked depending on its value:
- class TestIface2 {

  public static void main(String args[]) {

  Callback c = new Client();

  c.callback(42);

  AnotherClient ob = new AnotherClient();

  c = ob;

  c.callback(42);

  }
- }

Interface Inheritance

- One interface may inherit another interface.
- The inheritance syntax is the same for classes and interfaces.
- 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.

Inheritance and Implementation

- When a class implements an interface that inherits another interface, it must provide implementations for all inherited methods:
- class MyClass implements MyInterface2 {
- void myMethod1(…) { … }
- void myMethod2(…) { … }
- …
- }

Example: Interface Inheritance

- Consider interfaces A and B.
- interface A {
    void meth1();
    void meth2();
- }
- B extends A:
- interface B extends A {
    void meth3();
- }

Example: Interface Inheritance

- MyClass must implement all of A and B methods:
- 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().");
        }
```

- }

Example: Interface Inheritance

- Create a new MyClass object, then invoke all interface methods on it:
- class IFExtend {

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

- }

Interface Vs Abstract

Interface	Abstract
All methods in an interface are implicitly abstract.	An abstract class may contain both abstract and non-abstract methods.
A class may implement a number of Interfaces.	A class can extend only one abstract class.
In order for a class to implement an interface, it must implement all its declared methods.	A class may not implement all declared methods of an abstract class.
Variables declared in a Java interface is by default final.	An abstract class may contain non-final variables.
Members of a Java interface are public by default.	A member of an abstract class can either be private, protected or public.