Abstract Classes & Interfaces

Abstract Classes

- We can define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method.
- Sometimes you will want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.
- abstract method:

```
abstract type name(parameter-list);
```

- Any class that contains one or more abstract methods must also be declared abstract.
- To declare a class abstract, you simply use the **abstract** keyword in front of the class keyword at the beginning of the class declaration.

Abstract Classes

- There can be no objects of an abstract class.
- An abstract class cannot be directly instantiated with the new operator.
- Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be declared abstract itself.
- An abstract class can implements a concrete method.

Abstract Classes

- Example:

```
abstract class A {
            abstract void show();
            void display()
            {
                        System.out.println("Hello, I am display() in abstract class A");
            }
}
class B extends A {
            void show() {
                        System.out.println("Hello, I am show() in class B" );
            }
            public static void main(String args []) {
                        B obj=new B();
                        obj.show();
                        obj.display();
            }
}
```

Abstract Classes

- Example:

```
abstract class A {
        abstract void show();
        void display()
        {
                System.out.println("Hello, I am display() in abstract class A");
        }
}
class B extends A {
        void show() {
                System.out.println("Hello, I am show() in class B" );
        }
        public static void main(String args []) {
                B obj=new B();
                obj.show();
                obj.display();
        }
}
```

Output:

```
Hello, I am show() in class B
Hello, I am display() in abstract class A
```

Abstract Classes

- We cannot declare abstract constructors, or abstract static methods.
- Methods declared as final cannot be overridden.
- Although abstract classes cannot be used to instantiate objects, they can be used to create object references, because Java’s approach to run-time polymorphism is implemented through the use of superclass references.
- It is illegal to declare a class as both abstract and final since an abstract class is incomplete by itself and relies upon its subclasses to provide complete implementations.
- If a class that has one or more abstract methods, it must be declared abstract.
- However an abstract class may or may not have abstract methods.

Interfaces

- If you want to specify what a class must do, but not how it does it.
- Interfaces lack instance variables, and, as a general rule, their methods are declared without any body.
- 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.
- All methods and variables are implicitly public.
- All variables are implicitly final and static.
- The methods that implement an interface must be declared public.

Interfaces

- An interface is defined much like a class.

```
Access_modifier interface name_of_interface {
        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;
}
```

- Methods that are declared have no bodies. They end with a semicolon after the parameter list.
- Each class that includes such an interface must implement all of the methods.

Interfaces

- Once an interface has been defined, one or more classes can implement that interface.

```
interface interfacename{
void method(param-list);
}
```

```
class classname [extends superclass] [implements interface [,interface...]] {
// class-body
}
```

Interfaces

- Once an interface has been defined, one or more classes can implement that interface.

```
interface A {
void show(int i);
}

class B implements A{
    public void show(int i)
    {
        System.out.println(“show() called with " + p);
    }
}
```

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7) An abstract class can be extended using keyword **extends**.	An interface can be implemented using keyword **implements**.
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

Interfaces

- Interfaces Can Be Extended:
- One interface can inherit another by use of the keyword extends.
- The syntax is the same as for inheriting classes.
- When a class implements an interface that inherits another interface, it must provide implementations for all methods required by the interface inheritance chain.

Interfaces

- Interfaces Can Be Extended:

```
interface A                              interface B
{                                        {
        void show();                             void display();
}                                        }
class C implements A, B {
        public void show() {
                System.out.println("Hello, I am show()" );
        }
        public void display() {
                System.out.print("hello, I am display()");
        }
        public static void main(String args []) {
                C obj=new C();
                obj.show();
                obj.display();
        }
}
```