**Abstract Classes and Methods in Java**

A class that is declared using “abstract” keyword is known as abstract class. It may or may not include abstract methods which means in abstract class you can have concrete methods (methods with body) as well along with abstract methods ( without an implementation, without braces, and followed by a semicolon). An abstract class can not be **instantiated** (you are not allowed to create **object** of Abstract class).

**Abstract class declaration**  
Specifying **abstract keyword** before the class during declaration, makes it abstract. Have a look at below code:

// Declaration using abstract keyword

abstract class AbstractDemo{

// Concrete method: body and braces

public void myMethod(){

//Statements here

}

// Abstract method: without body and braces

abstract public void anotherMethod();

}

Since abstract class allows concrete methods as well, it does not provide 100% abstraction. You can say that it provides partial abstraction. **[Interfaces](http://beginnersbook.com/2013/05/java-interface/" \o "Interface in java with example programs" \t "_blank)**are used for 100% abstraction (full [**abstraction**](http://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/))

Remember two rules:  
1) If the class is having few abstract methods and few concrete methods: declare it as abstract class.  
2) If the class is having only abstract methods: declare it as interface.

**Error!! – Object creation of abstract class is not allowed**  
As discussed above, we cannot instantiate an abstract class. The following code throws an error.

abstract public class AbstractDemo{

public void myMethod(){

System.out.println("Hello");

}

abstract public void anotherMethod();

}

public class ConcreteDemo{

public void anotherMethod() {

System.out.print("Abstract method");

}

public static void main(String args[])

{

//Can't create object of abstract class - error!

AbstractDemo obj = new AbstractDemo();

obj.display();

}

}

Output:

Unresolved compilation problem: Cannot instantiate the type AbstractEx1

Note: The class that extends the abstract class, have to implement all the abstract methods of abstract class, else they can be declared abstract in the class as well.

**Why we need an abstract class?**

Let me explain this with an example. Suppose there is a class Animal and there are few other classes like Cat, Dog and Horse. These classes extends Animal class so basically they are having few common habits(methods in technically) which they are inheriting from Animal class. Now, if you have understood the above example then you would have been able to figure out that **creating object of Animal class has no significance** as you can’t judge that the **new** object of Animal class will represent which animal. Hence for such kind of scenarios we generally creates an **abstract class** and later**concrete classes** extends these classes and overrides their methods accordingly and can have their own methods as well.

**Abstract vs Concrete**

A class which is not abstract is referred as **Concrete class**. In the above example which I explained – Animal is a abstract class and Cat, Dog andHorse are concrete classes.

**Key Points:**

1. An abstract class has no use until unless it is extended by some other class.
2. If you declare an **abstract method** (discussed below) in a class then you must declare the class abstract as well. you can’t have abstract method in a **non-abstract class**. It’s vice versa is not always true: If a class is not having any abstract method then also it can be marked as abstract.
3. Abstract class can have non-abstract method (concrete) as well.

**Abstract methods**

Well, we already discussed about abstract methods in the above section. Lets take few examples to understand it better.

**syntax:**

public abstract void display();

Points to remember about abstract method:  
1) Abstract method has no body.  
2) Always end the declaration with a **semicolon**(;).  
3) It must be [**overridden**](http://beginnersbook.com/2014/01/method-overriding-in-java-with-example/). An abstract class must be extended and in a same way abstract method must be overridden.  
4) Abstract method must be in a abstract class.

**Note:** The class which is extending abstract class must override (or implement) all the abstract methods.

**Example of Abstract class and method**

abstract class Demo1{

public void disp1(){

System.out.println("Concrete method of abstract class");

}

abstract public void disp2();

}

class Demo2 extends Demo1{

/\* I have given the body to abstract method of Demo1 class

It is must if you don't declare abstract method of super class

compiler would throw an error\*/

public void disp2()

{

System.out.println("I'm overriding abstract method");

}

public static void main(String args[]){

Demo2 obj = new Demo2();

obj.disp2();

}

}

Output:

I'm overriding abstract method

# Interface in java with example programs

Earlier we discussed about [**abstract classes**](http://beginnersbook.com/2013/05/java-abstract-class-method/) that are used to achieve partial abstraction(hiding irrelevant details from user). In this tutorial we are going to discuss about interfaces which are used for achieving full abstraction. We will discuss what is an interface, what is the significance of it, when and how to use it.

### What is an interface

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body). Also, the variables declared in an interface are public, static & final by default. We will discuss these points in detail later in this post.

### What is the use of interfaces

As stated above they are used for abstraction. Since methods in interfaces does not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not support multiple inheritance, using interfaces we can achieve this as a class can implement more than one interfaces, however it cannot extend more than one classes.

**Declaration**  
Interfaces are declared by specifying a keyword “interface”. E.g.:

interface MyInterface

{

/\* All the methods are public abstract by default

\* Note down that these methods are not having body

\*/

public void method1();

public void method2();

}

### Interface Implementation

This is how a class implements an interface. It has to provide the body of all the methods that are declared in interface.  
Note: Class implements interface but an interface extends another interface.

interface MyInterface

{

public void method1();

public void method2();

}

class XYZ implements MyInterface

{

public void method1()

{

System.out.println("implementation of method1");

}

public void method2()

{

System.out.println("implementation of method2");

}

public static void main(String arg[])

{

MyInterface obj = new XYZ();

obj. method1();

}

}

Output:

implementation of method1

### Interface and Inheritance

As I discussed above that one interface can not implement another interface. It has to extend the other interface if required. See the below example where we have two interfaces Inf1 and Inf2. Inf2 extends Inf1 so If class implements the Inf2 it has to provide implementation of all the methods of interfaces Inf1 and Inf2.

public interface Inf1{

public void method1();

}

public interface Inf2 extends Inf1 {

public void method2();

}

public class Demo implements Inf2{

public void method1(){

//Implementation of method1

}

public void method2(){

//Implementation of method2

}

}

In the above program my “Demo” class is implementing only one interface “Inf2″ however it has to provide the implementation of all the methods of interface “Inf1″ too, because interface Inf2 extends Inf1.

**Key points:** Here are the key points to remember about interfaces:  
1) We can’t instantiate an interface in java.

2) Interface provides complete [**abstraction**](http://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/) as none of its methods can have body. On the other hand, [**abstract class**](http://beginnersbook.com/2013/05/java-abstract-class-method/) provides partial abstraction as it can have abstract and concrete(methods with body) methods both.

3) implements keyword is used by classes to implement an interface.

4) While providing implementation in class of any method of an interface, it needs to be mentioned as public.

5) Class implementing any interface must implement all the methods, otherwise the class should be declared as “abstract”.

6) Interface cannot be declared as private, protected or transient.

7) All the interface methods are by default **abstract and public**.

8) Variables declared in interface are **public, static and final** by default.

interface Try

{

int a=10;

public int a=10;

public static final int a=10;

final int a=10;

static int a=0;

}

All of the above statements are identical.

9) Interface variables must be initialized at the time of declaration otherwise compiler will through an error.

interface Try

{

int x;//Compile-time error

}

Above code will throw a compile time error as the value of the variable x is not initialized at the time of declaration.

10) Inside any implementation class, you cannot change the variables declared in interface because by default, they are public, static and final. Here we are implementing the interface “Try” which has a variable x. When we tried to set the value for variable x we got compilation error as the variable x is public static **final** by default and final variables can not be re-initialized.

Class Sample implements Try

{

public static void main(String arg[])

{

x=20; //compile time error

}

}

11) Any interface can extend any other interface but cannot implement it. Class implements interface and interface extends interface.

12) A **class** can implements any **number of interfaces**.

13) If there are having **two or more same methods** in two interfaces and a class implements both interfaces, implementation of one method is enough.

interface A

{

public void aaa();

}

interface B

{

public void aaa();

}

class Central implements A,B

{

public void aaa()

{

//Any Code here

}

public static void main(String arg[])

{

//Statements

}

}

14) Methods with same signature but different return type can’t be implemented at a time for two or more interfaces.

interface A

{

public void aaa();

}

interface B

{

public int aaa();

}

class Central implements A,B

{

public void aaa() // error

{

}

public int aaa() // error

{

}

public static void main(String arg[])

{

}

}

15) Variable names conflicts can be resolved by interface name e.g:

interface A

{

int x=10;

}

interface B

{

int x=100;

}

class Hello implement A,B

{

public static void Main(String arg[])

{

System.out.println(x); // reference to x is ambiguous both variables are x

System.out.println(A.x);

System.out.println(B.x);

}

}

#### Benefits of having interfaces:

Following are the advantages of interfaces:

1. Without bothering about the implementation part, we can achieve the security of implementation
2. In java, [**multiple inheritance**](http://beginnersbook.com/2013/05/java-multiple-inheritance/) is not allowed, However by using interfaces you can achieve the same . A class can extend only one class but can implement any number of interfaces. It saves you from Deadly Diamond of Death(DDD) problem.

**Difference Between Abstract Class and Interface in Java**

I have already covered basics of **Abstract class and Interface** while discussing [**OOPs concepts**](http://beginnersbook.com/2013/04/oops-concepts/) and in following posts:

1. [**abstract class in java**](http://beginnersbook.com/2013/05/java-abstract-class-method/)
2. [**java – interface**](http://beginnersbook.com/2013/05/java-interface/)

In this post we will discuss **difference between Abstract Class and Interface in Java with examples.**

|  |  |  |
| --- | --- | --- |
|  | **abstract Classes** | **Interfaces** |
| 1 | abstract class can extend only one class or one abstract class at a time | interface can extend any number of interfaces at a time |
| 2 | abstract  class  can extend from a class or from an abstract class | interface can extend only from an interface |
| 3 | abstract  class  can  have  both  abstract and concrete methods | interface can  have only abstract methods |
| 4 | A class can extend only one abstract class | A class can implement any number of interfaces |
| 5 | In abstract class keyword ‘abstract’ is mandatory to declare a method as an abstract | In an interface keyword ‘abstract’ is optional to declare a method as an abstract |
| 6 | abstract  class can have  protected , public and public abstract methods | Interface can have only public abstract methods i.e. by default |
| 7 | abstract class can have  static, final  or static final  variable with any access specifier | interface  can  have only static final (constant) variable i.e. by default |

Each of the above difference between **Abstract class vs Interface**is explained with an example below –

**Abstract class vs interface**

**Difference No.1:**

**Abstract class can extend only one class or one abstract class at a time**

class Example1{

public void display1(){

System.out.println("display1 method");

}

}

abstract class Example2{

public void display2(){

System.out.println("display2 method");

}

}

abstract class Example3 extends Example1{

abstract void display3();

}

class Example4 extends Example2{

public void display3(){

System.out.println("display3 method");

}

}

class Demo{

public static void main(String args[]){

Example4 obj=new Example4();

obj.display3();

}

}

Output:

display3 method

**Interface can extend any number of interfaces at a time**

//first interface

interface Example1{

public void display1();

}

//second interface

interface Example2 {

public void display2();

}

//This interface is extending both the above interfaces

interface Example3 extends Example1,Example2{

}

class Example4 implements Example3{

public void display1(){

System.out.println("display2 method");

}

public void display2(){

System.out.println("display3 method");

}

}

class Demo{

public static void main(String args[]){

Example4 obj=new Example4();

obj.display1();

}

}

Output:

display2 method

**Difference No.2:**

**Abstract class can be inherited by a class or an abstract class**

class Example1{

public void display1(){

System.out.println("display1 method");

}

}

abstract class Example2{

public void display2(){

System.out.println("display2 method");

}

}

abstract class Example3 extends Example2{

abstract void display3();

}

class Example4 extends Example3{

public void display2(){

System.out.println("Example4-display2 method");

}

public void display3(){

System.out.println("display3 method");

}

}

class Demo{

public static void main(String args[]){

Example4 obj=new Example4();

obj.display2();

}

}

Output:

Example4-display2 method

**Interfaces can be extended only by interfaces. Classes has to implement them instead of extend**

interface Example1{

public void display1();

}

interface Example2 extends Example1{

}

class Example3 implements Example2{

public void display1(){

System.out.println("display1 method");

}

}

class Demo{

public static void main(String args[]){

Example3 obj=new Example3();

obj.display1();

}

}

Output:

display1 method

**Difference No.3**

**Abstract class can have both abstract and concrete methods**

abstract class Example1 {

abstract void display1();

public void display2(){

System.out.println("display2 method");

}

}

class Example2 extends Example1{

public void display1(){

System.out.println("display1 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**Interface can only have abstract methods, they cannot have concrete methods**

interface Example1{

public abstract void display1();

}

class Example2 implements Example1{

public void display1(){

System.out.println("display1 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**Difference No.4**

**A class can extend only one abstract class at a time**

abstract class Example1{

public void display1(){

System.out.println("display1 method");

}

}

abstract class Example2{

abstract void display2();

}

class Example3 extends Example1{

public void display3(){

System.out.println("display3 method");

}

}

class Demo{

public static void main(String args[]){

Example3 obj=new Example3();

obj.display3();

}

}

**A class can implement any number of interfaces at a time**

interface Example1{

public void display1();

}

interface Example2{

public void display2();

}

class Example3 implements Example1,Example2{

public void display1(){

System.out.println("display1 method");

}

public void display2(){

System.out.println("display2 method");

}

public void display3(){

System.out.println("display3 method");

}

}

class Demo{

public static void main(String args[]){

Example3 obj=new Example3();

obj.display1();

obj.display3();

}

}

**Difference No.5**

**In abstract class, the keyword ‘abstract’ is mandatory to declare a method as an abstract**

abstract class Example1{

public abstract void display1();

}

class Example2 extends Example1{

public void display1(){

System.out.println("display1 method");

}

public void display2(){

System.out.println("display2 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**In interfaces, the keyword ‘abstract’ is optional to declare a method as an abstract because all the methods are abstract by default**

interface Example1{

public void display1();

}

class Example2 implements Example1{

public void display1(){

System.out.println("display1 method");

}

public void display2(){

System.out.println("display2 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**Difference No.6**

**Abstract class can have protected , public and public abstract methods**

abstract class Example1{

protected abstract void display1();

public abstract void display2();

public abstract void display3();

}

class Example2 extends Example1{

public void display1(){

System.out.println("display1 method");

}

public void display2(){

System.out.println("display2 method");

}

public void display3(){

System.out.println("display3 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**Interface can have only public abstract methods i.e. by default**

interface Example1{

void display1();

}

class Example2 implements Example1{

public void display1(){

System.out.println("display1 method");

}

public void display2(){

System.out.println("display2 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**Difference No.7**

**Abstract class can have static, final or static final variables with any access specifier**

abstract class Example1{

private int numOne=10;

protected final int numTwo=20;

public static final int numThree=500;

public void display1(){

System.out.println("Num1="+numOne);

}

}

class Example2 extends Example1{

public void display2(){

System.out.println("Num2="+numTwo);

System.out.println("Num2="+numThree);

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

obj.display2();

}

}

**Interface can have only static final (constant) variable i.e. by default**

interface Example1{

int numOne=10;

}

class Example2 implements Example1{

public void display1(){

System.out.println("Num1="+numOne);

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

# Abstract method with examples in Java

In this tutorial we will see two examples of **abstract methods**:

## Example 1: abstract method in abstract class

**Note**: 1) Abstract class can also have regular(or concrete) methods along with [**abstract methods**](http://beginnersbook.com/2013/05/java-abstract-class-method/).  
2) Abstract methods do not have body, they just have prototype(method signature).  
3) Abstract methods must be implemented in the child class (if the class is not abstract) otherwise program will throw compilation error.

package beginnersbook.com;

//abstract class

abstract class Sum{

//abstract methods

public abstract int SumOfTwo(int n1, int n2);

public abstract int SumOfThree(int n1, int n2, int n3);

//Regular method

public void disp(){

System.out.println("Method of class Sum");

}

}

class AbstractDemo extends Sum{

public int SumOfTwo(int num1, int num2){

return num1+num2;

}

public int SumOfThree(int num1, int num2, int num3){

return num1+num2+num3;

}

public static void main(String args[]){

AbstractDemo obj = new AbstractDemo();

System.out.println(obj.SumOfTwo(3, 7));

System.out.println(obj.SumOfThree(4, 3, 19));

obj.disp();

}

}

Output:

10

26

Method of class Sum

## Example 2: abstract method in interface

All the methods of an [**interface**](http://beginnersbook.com/2013/05/java-interface/) are public abstract by default.

package beginnersbook.com;

//Interface

interface Multiply{

//abstract methods

public abstract int multiplyTwo(int n1, int n2);

/\* We need not to mention public and abstract

\* as all the methods in interface are

\* public and abstract by default

\*/

int multiplyThree(int n1, int n2, int n3);

/\*Regular (or concrete) methods are not allowed

\* in an interface.

\*/

}

class AbstractDemo2 implements Multiply{

public int multiplyTwo(int num1, int num2){

return num1\*num2;

}

public int multiplyThree(int num1, int num2, int num3){

return num1\*num2\*num3;

}

public static void main(String args[]){

AbstractDemo2 obj = new AbstractDemo2();

System.out.println(obj.multiplyTwo(3, 7));

System.out.println(obj.multiplyThree(1, 9, 0));

}

}

Output:

21

0