**ABSTRACTION**

**Definition:**

Abstraction represents the essential feature of a system, partially or without involving the complexity. In java, abstraction is implemented using abstract class and interfaces. Implementing the class will define actual working.

**Abstract class:**

A method without a body is called as abstract method

The abstract method should be defined using keyword **“abstract”.**

If a class contains abstract keyword then the class is called **abstract class.**

**Concrete class:**

A method which contains body is called concrete method

If a class contains only concrete methods is called concrete class

**Important points:**

1. An abstract class can have any number of abstract methods/concrete methods or it can contain only abstract methods or only concrete methods or mix of both.
2. If class contains at least one abstract method then the class is called abstract class, but it’s not necessary that an abstract class should compulsory contain abstract.
3. An abstract method should only be non-static.
4. A concrete method can be both static and non-static.
5. We Cannot instantiate abstract class.
6. If you want to create the object of abstract class then inherit the members of abstract class and override it in sub class.
7. An abstract method should be extended to the sub class, the sub class should implemented all method(abstract) or else the sub class should also be declared as abstract class

Abstract method

Ex: void test();

Concrete method:

Ex: void test(){

}

Program:

package com.qsp.pack1;

abstract class B//class is abstracted because it has an abstract method

{

abstract void print();//no body, since it called abstract method.

//abstract static void print();.//no abstract cannot be static because its cannot be overriden, illegal combination

static void disp()//concrete method can be static or non static

{

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

}

}

class C extends B// inheriting the all 2 methods

{

void print()

{

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

}

}

class Demo55

{

public static void main(String args[])

{

C c1=new C();

c1.print();// it will access the print method

c1.disp();//since disp() is static cannot be accessed using reference variable, so complier will replace it with class name C, C.disp()

B.disp();//calling static method of an abstract using abstract class name is also possible

}

}

O/P:

Print() method

disP() method

disP() method

Program:

package com.qsp.pack1;

abstract class C

{

static int i;

int j;

static final double PI=3.14;

static void print()

{

System.out.println("i= " + i);

}

void disp()

{

System.out.println("j= " +j);

}

}

class D extends C{

}

class Demo56

{

public static void main(String args[])

{

D d1=new D();

d1.disp();

D.print();

System.out.print("-------------------");

System.out.println(d1.j);//we are accessing non static variable through a reference variable

System.out.println(d1.i);//we are accessing static variable,but here we are using a reference variable, which doesnt not make any sense, so Compiler will replace it as D.i

System.out.println(D.i);

System.out.println("-------------");

C.print();//calling the static method of a an abstract class which is possible

D.print();

}

}

O/P:

j= 0

i= 0

-------------------0

0

0

-------------

i= 0

i= 0

Program:

package com.qsp.pack1;

abstract class A

{

abstract void test();

abstract public void test1();

//abstract private void test2();//private members are not inherited so we cannot override it.

abstract protected void test3();

}

class B extends A

{

public void test()//over ridding the test() method

{

System.out.println("test() of implementation in class B");

}

public void test1()//over ridding the test()1 method

{

System.out.println("test1() of implementation in class B");

}

private void test2()// this is private method belongs to only this class not inherited from A

{

System.out.println("test2() of implementation in class B");

}

protected void test3()//over ridding the test()3 method

{

System.out.println("test3() of implementation in class B");

}

}

Program:

package com.qsp.pack1;

abstract class Demo57{

static int i=55;

static void print(){

System.out.println("i= " + i);

}

abstract void disp();

public static void main(String args[]){

Demo57.print();

}

}

O/P:

i= 55

Note:

Abstract class can contain a main method.

Program:

package com.qsp.pack1;

abstract class A{

int i;

A(int i){

System.out.println("hello");

this.i=i;

}

}

class B extends A{

B(){

super(100);

}

}

class Demo58{

public static void main(String args[]){

B b1=new B();

System.out.println(b1.i);

}

}

O/P:

hello

100

**INTERFACES**

Definition:

An Interface is a type and it’s a definition block which contains only abstract methods

It is like 100% abstract class.

Note:

Java supports 3 non primitive types

class type

Array type

Interfaces type

Ex:

interface interfacename

{

---------------

---------------- //interface body

----------------

}

Notes:

1. An interface type starts with a keyword “interface”.
2. A java file can contain only interface definition block. After compilation of interface definition block, compiler generates .Class file for the interface.
3. Java file can have both class definition block and interface definition block. After compilation of class definition block and interface definition block, compiler creates .Class file for Interface definition block
4. Inside interface definition block concrete methods are not allowed.
5. The interfaces should contain only abstract methods(˘which is non-static)
6. By default the methods in interfaces are abstract and public, Other than “public” we cannot use any other access specifier in interface block.
7. All variables in an interface must be public static and final i.e. it should declare only constants and should be initialized at the declaration time.
8. A method cannot be declared as static in interface
9. A class cannot extend more than one class (java doesn’t support multi inheritance through classes)but a class can implement more than 1 interface(Ex: class A implement Idemo1,Idemo2).
10. We cannot instantiate (create an object) an interface.
11. An interface can extend one or more other interfaces.
12. An interface cannot extend anything other than interfaces.
13. An interface cannot implement another interfaces or class.

If we create an interface,

Ex:

interface Idemo

{

}

Compiler will takes it as,

abstract interface Idemo

{

}

Program A:

package com.qsp.pack1;

abstract interface Idemo// if we don’t specify **abstract** also, complier automatically add **abstract**

{

void test1();// if we don’t specify **public abstract** also, complier automatically add **“public abstract”**

public abstract void test2();

}

class A implements Idemo

{

public void test1(){ // we should mention **“public”** compulsory, if not complier will take it as default

System.out.println("test1() implements in class A");

}

public void test2() // same here also

{

System.out.println("test2() implements in class A");

}

void disp()

{

test2();// can call directly

}

public static void main(String args[])

{

A a1=new A();

a1.test1();

a1.test2();

a1.disp();// which internally calls test2()

}

}

class Demo50

{

public static void main(String args[])

{

A a1=new A();

a1.test1();

a1.test2();

}

}

**Note:** when we compile it we get 3 .class files, we can execute A.class and Demo50.class as it both has main method

O/P:

C:\practice\_java\bin>java com.qsp.pack1.A

test1() implements in class A

test2() implements in class A

test2() implements in class A

C:\practice\_java\bin>java com.qsp.pack1.Demo50

test1() implements in class A

test2() implements in class A

Program B:

package com.qsp.pack1;

interface IDemo2

{

void test1();

void test2();

}

abstract class B implements IDemo2

{

public void test1()

{

System.out.println("test1() can be implemented in class B");

}

}

class C extends B

{

public void test2()

{

System.out.println("test2() can be implemented in class c");

}

}

class Demo60

{

public static void main(String...args)

{

C c1=new C();

c1.test1();

c1.test2();

}

}

**Note:**

First we are implementing the all the methods to the class B and we are over ridding only one method and declaring that class B as abstract. Later, we extending the B to class C and we are over ridding the remaining method, we are writing creating Demo60 class and when we create an object both methods are loaded.

O/P:

C:\practice\_java\bin>java com.qsp.pack1.Demo60

test1() can be implemented in class B

test2() can be implemented in class c

**NOTE:**

**A**. Interface can extend only interface and interface can extend more than one interface

**B**. Class can extend only class and implements interface, Class can extend only one class and implements more than one interface.

|  |  |
| --- | --- |
| **ABSTRACT** | **INTERFACE** |
| An Abstract class should be declared with the keyword “abstract” | An interface should start with the keyword “interface” and by default it’s abstract. |
| An abstract class can contain both concrete and abstract methods | An interface contains only abstract methods |
| In abstract class, the method is explicitly declared as abstract | In interface, by default all the methods are abstract. |
| In abstract class, we can use all access specifiers except private. | In interface we can use only public access specifier. |
| In abstract class, abstract methods are non-static and the concrete methods are both static and non-static. | Interface has only non-static methods. |
| In abstract class, we can have static, non-static, and final variables. | In interfaces, we can have only final variables (Ex: public static final int I ;) |
| In abstract class, when a concrete method is added it will not affect the sub class, but when we add an abstract method, it will affect the sub class | In interface, if you add new method then all the classes which implements the interfaces will fail/break |