**ABSTRACTION**

**Definition:**

Abstraction represents the showing only **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.

NOTE: Inside the abstract class, when we create an abstract method, we use a **semicolon (;)** instead of **curly brackets ({}).**

Example: using smart phones without knowing how it is made

**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 ke**yword 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**
3. An **abstract method** should only be **non-static**.
4. We **Cannot instantiate abstract class**.
5. If you want to **create the object of abstract** class then **inherit the members** of abstract class and **override it in sub class**.
6. 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**

**class** Sony {

}

**abstract** **class** TV **extends** Sony {

**abstract** **void** display();

**abstract** **void** speaker();

**void** usb() {

}

//abstract void internet();

}

**class** Bravia **extends** TV {

**void** display() {

System.***out***.println("display : 1080 px");

}

**void** speaker() {

System.***out***.println("speaker : 50 W");

}

}

**class** Bravia4k **extends** TV {

**void** display() {

System.***out***.println("display : 1920 px");

}

**void** speaker() {

System.***out***.println("speaker : 150 W");

}

}

**class** simple **extends** TV {

**void** display() {

System.***out***.println("display : 780 px");

}

**void** speaker() {

System.***out***.println("speaker : 10 W");

}

}

**public** **class** Run6 {

**public** **static** **void** main(String[] args) {

Bravia b = **new** Bravia();

//TV tv = new TV();

}

}

**INTERFACES**

Definition:

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

It is like **100% abstract class.**

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.
3. Java file can have both class definition block and 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 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).
9. The class which implements interface , while defining the abstract methods of interface , we should make inherited method as **public**. Since in class by default the type of method is ‘**default’**, and in interface by default the methods are ‘**public’**, so when class inheriting interface the visibility of method we should explicitly convert the access specifier of method from default to public because so the visibility of default type is less compare to public- cannot convert from **higher visibility to lower visibility**.

interface Idemo

{

Void display)

}

Class ABC implements Idemo

{

}

We cannot instantiate (create an object) an interface.

An interface can extend one or more other interfaces.

clas ---> class [extends][is-a relationship]

\* class ----> interface [implements]

\* interfcae ----> interface [extends][has-a relationship]

\* interface ----> class [It is not possible]

Ex:

**interface** IParent1 {

**int** ***a*** = 10;

**void** test**();// public and abstract**

}

**interface** IParent2 **extends** IParent1 {

**void** test();

}

**class** Sub **implements** IParent1, IParent2 {

**public** **void** test() **{ //default**

System.***out***.println("Test method");

}

}

**public** **class** Run7 {

**public** **static** **void** main(String[] args) {

Sub s1 = **new** Sub();

s1.test();

}

}

|  |  |
| --- | --- |
| **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 method**s. |
| In abstract class, we can **have static, non-static, and final variables.** | In interfaces, by default variable are static and **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** |