Constructor in Abstract Class:-

Parent class method by default is available to the child class, so method overloading is possible but the parent class constructor is not by default available to the Child. So, the Inheritance concept is not applicable to constructors, and hence overriding concept also not applicable to constructors. But constructor overloading is possible.

We can take any number of constructors in a java class including abstract class also but we can't take a constructor inside the interface.

Abstract class constructor will be executed for every child class object creation to perform initialization of child class object only.

Whenever we are creating a child class object then the respective constructor will execute and that constructor implicitly calls the parent class constructor.

Example:-1

```
abstract class Parent
{
Parent()
{
System.out.println(this.hashCode());
}
}
class Child extends Parent
{
Child()
{
System.out.println(this.hashCode());//11394033
}
}
class Test
{
public static void main(String[] args)
{
Child c=new Child();
System.out.println(c.hashCode());//11394033
}
}
```

```
Example:-2
abstract class Parent
{
Parent(int x)
{
System.out.println(x);
System.out.println(this.hashCode());
}
}
class Child extends Parent
{
Child()
{
super(10);
System.out.println(this.hashCode());//11394033
}
}
class Test
{
public static void main(String[] args)
{
Child c=new Child();
System.out.println(c.hashCode());//11394033
}
}
```

INTERFACE

Interface:-

Definition:-1 Any service requirement specification (srs) is considered as interface.

Example:-If an object wants to performed serialization then corresponding class must be implements serialization interface then object eligible to performed serialization. For serialization service requirement specification, we need serialization interface.

Definition:-2 The interface is highlighting set of services but hiding implementation implements .

Example:- In our house we are using a television, we are able to see the picture and change the channel vary easily but we don't know what the internal implementation.

Definition:-3 Interface is also one type of class that contains only abstract method it is extension of abstract class hence interface considered as hundred percent or pure abstract class.

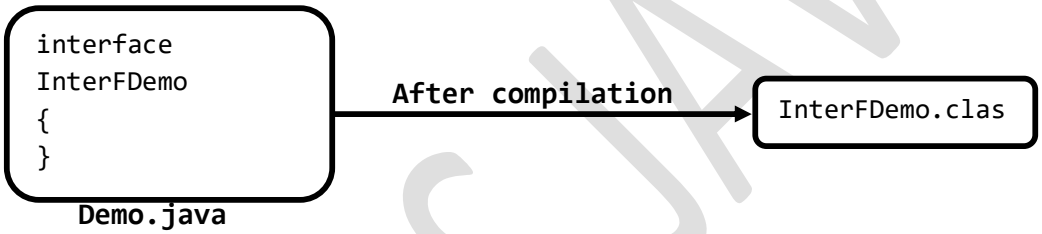
Note:-1 we can declare interface by using interface keyword.

Syntax:-

```
interface interface-name
{
}
```

Note:- After compilation interface it will generate .class file and .class file name is similar to the interface name.

Example:-



Note:-If we don't know anything about the implementation of method such type of requirement we used interface.

Example:-

```
interface   InterfaceDemo
{
public abstract void m1();
public abstract void m2();
}
```

If we know partial about the implementation of method, such type of requirement we used abstract class.

Example:-

```
abstract class AbstractDemo
{
public void m1()
{
```

```
System.out.println("Hello");
}
public abstract void m2();
}
```

If we know completely about the implementation of a method, such type of a requirement we used concrete class.

Example:-

```
class Test
{
public void m1()
{
System.out.println("Hello m1");
}
public void m2()
{
System.out.println("Hello m2()");
}
}
```

Note:-1 A class can extends only one class at a time if we are trying to extends more than one class then we will get compile time error.

```
class Test1
{
}
```
valid

```
class Test2 extend Test1
{
}
```
valid

```
class Test2 extends Test1,Test2
{
}
```
invalid

Note:-2 An interface can extends any number of interface simultaneously.

```
interface Inf1
{
}
```
valid

```
interface Inf2 extends Inf1
{
}
```
valid

```
interface Inf3 extends Inf1 ,Inf2
{
}
```
valid

Note:-3 **A class can implements any number interface simultaneously.**

```
interface Inf1
{
}
```

valid

```
interface Inf2
{
}
```

valid

```
class Test implements Inf1,Inf2
{
}
```

valid

Note:-4 **A class can extends another class and can implements any number of interface simultaneously**

```
interface Inf1
{
}
```

```
interface Inf2
{
}
```

```
class Test1
{
}
```

```
class Test2 extends Test1 implements Inf1,Inf2
  {
  }
```

Interface method:-

Point:-1 **Inside interface every method by default public and abstract whether we declaring or not**

Exapmle:-

```
interface InterFaceDemo
{
        void m1();
        void m2();
}
```

After compilation

```
interface InterFaceDemo
{
        public abstract void m1();
        public abstract void m2();
}
```

Point:-2 **Every method present inside the interface is by default public and abstract so at the time of method implementation we should compulsory method declare as public otherwise we will get compile time error.**

Example:-

```
interface Intef1
{
public abstract void m1();
public abstract void m2();
}
```

```
abstract class Test1 implements Intef1
{
public void m1()
{
System.out.println("m1() method");
}
}


class Test2 extends Test1
{
public void m2()
{
System.out.println("m2() method");
}
public static void main(String[]args)
{
Test2 t2=new Test2();
t2.m1();
t2.m2();
}
}
```

Result:- m1() method
m2() method

Note :-If Test1 m1() method and Test2 m2() method is not declare as public then we will get compile time error.

Point:-3 when we implementing an interface then provide implementation of all abstract inside implementation class if implementation class is unable to provide implementation of all abstract then declare implementation class as abstract and next level child class will be provide implementation if the child class also unable to provide implementation then declare the child class as abstract and this process is continue until we provide implementation of all abstract method of interface.

Example:-

```
interface InterFaceDemo
{
public abstract void m1();
```

```
public abstract void m2();
public abstract void m3();
}

abstract class Test1 implements InterFaceDemo
{
public void m1()
{
System.out.println("Test1 m1() method");
}
}
abstract class Test2 extends Test1
{
public void m2()
{
System.out.println("Test2 m2() method");
}
}
class Test3 extends Test2
{
public void m3()
{
System.out.println("Test3 m3() method");
}
public static void main(String[]args)
{
Test3 t3=new Test3();
t3.m1();
t3.m2();
t3.m3();
}
}
```

Result:-
```
Test1 m1() method
Test2 m2() method
Test3 m3() method
```

Interface Variable:- Every variable present in Interface is by default public static final whether we declaring or not.

The main objective of interface variable to defined requirement constant every implementation class.

Syntax of Interface variable:-

```
public static final data type variable_name;
```

we can access in any implementation class.

Without creating of object we can access in any implementation class.

Implementation class can read only interface variable but can't chance if any one implementation class change value then remaining class will be effected with this change.

Example:-

```
interface InterFDemo
{
 int x=10;
}
```

After compilation

```
interface InterFDemo
{
public static final int x=10;
}
```

Note:-1 Interface variable we should compulsory initialized at the time of declaration otherwise we will get compile time error.

Example:-

```
interface InterFDemo
{
 int x;
}
```

Result:-

```
Test.java:3: error: = expected
        int x;
```

Note:-2 Interface variable we can't change in implementation class. If we will try to change then we will get compile time error.

Example:-

```
interface InterFDemo
{
int x=11;
}
class Test implements InterFDemo
{
```

```
public static void main(String[]args)
{
x=111;
}
}
```

Result:- Test.java:9: error: cannot assign a value to final variable x
x=111;

interface naming conflict:-

Case1:-If two interface contains same method signature and same return type then implementation class provide implementation only one method.

Example:-

```
interface InfFirst
{
public abstract void m1();
}
interface InfSecond
{
public abstract void m1();
}
class TestInterface implements InfFirst,InfSecond
{
public void m1()
{
System.out.println("m1 abstract method");
}
public static void main(String[]args)
{
TestInterface t=new TestInterface();
t.m1();
}
}
```

Result:- m1 abstract method

Case:-2 If two interface contain a method with same signature but different return type then implementation class can't provide implementation of both method (if return type is not co-variant type) because inside class we can't declare two method with same signature otherwise we will get compile time error. So in this case we can't implements two interface simultaneously.

In java we can implement any number of interface simultaneously but in this case more than one interface contains a method with same signature

Case3:-If two interface contain a method with same signature but different argument type then we can provide implementation of both method and these method access overloaded method.

Example:-

```
interface InfFirst
{
public abstract void m1(int x);
}
interface InfSecond
{
public abstract void m1(String str);
}
class TestInterface implements InfFirst,InfSecond
{
public void m1(int x)
{
System.out.println("m1 int-arg method");
}
public void m1(String str)
{
System.out.println("m1 String-arg method");
}
public static void main(String[]args)
{
TestInterface t=new TestInterface();
t.m1(12);
t.m1("vikas");
}
}
Result:- m1 int-arg method
m1 String-arg method
```

Interface variable naming conflict:-

Case1:-If two interface contain a variable with same name then there may be chance of ambiguity in implementation class but by using interface name we can solve this problem.

```
Example:-
interface InfFirst
{
int x=11;
}
interface InfSecond
{
int x=12;
}
class TestInterface implements InfFirst,InfSecond
{
public static void main(String[]args)
{
//System.out.println(x);//line-13
System.out.println(InfFirst.x);
System.out.println(InfSecond.x);
}
}
```

Result:- 11
12

Note:- If we are not commenting line-13 then we will get compile time error saying

```
Test.java:13: error: reference to x is ambiguous
System.out.println(x);
both variable x in InfFirst and variable x in InfSecond match
```

Marker Interface:- If an interface does not contain any member (method and variable) or no field or empty interface such types of interface we can say marker interface or ability interface or tagged interface.

When we implements marker interface then our object get some extra ability this extra ability provided by jvm.

By using marker interface programming make simple. Marker interface:-

A) Cloneable(I)
B) Serializable(I)
C) Remote(I)
D) RandomAccess(I) etc

Adopter class :- It is normal java class they provide empty implementation of all method of interface.

Example:-

```
interface InterDemo
{
public abstract void m1();
public abstract void m2();
public abstract void m3();
public abstract void m4();
public abstract void m5();
}
class AdopterDemo
{
public void m1(){}
public void m2(){}
public void m3(){}
public void m4(){}
public void m5(){}
}
```

If we implementing an interface then we should provide implementation of all method its compulsory. whether it need or not so it increase the length of the code and reduce readability of the code overcome these we used adopter class.

Difference between Abstract class and Interface.

Abstract class	Interface
1.abstact class is partial abstract because it contain both abstract and non abstract method	1.Interface is hundred percent abstract because inside Interface all method must be abstract .
2.abstact class declare with abstract modifier	2.interface declared with interface keyword
3. Abstract class method not need to public and abstract.	3. interface method must be public and abstract whether we declared or not
4.Every variable present inside abstract class not need be public, static ,final.	4.Every method present inside the Interface is by default public, static and final.
5.abstract class method not need to initialized at the time of method declaration.	5. interface method must initialized at time of declaration.

6.Inside abstract class we can declare main method and constructor	6.Inside interface we can't declare main method and constructor.
7.Inside abstract class we can declare instance block and static block.	7.Inside interface we can't declare static block and instance block.

Default Method inside Interface:-

Until 1.7 version onwards inside interface we can declare only public abstract methods and public static final variables (every method present inside interface is always public and abstract whether we are declaring or not).

Every variable declared inside interface is always public static final whether we are declaring or not.

But from 1.8 version onwards we can declare default concrete methods also inside interface, which are also known as defender methods or virtual extension methods.

We can declare default method with the "default" keyword:-

Syntax of default method:-

```
default return_type method_name()
{
//Method body.
}
```

Example:-

```
default void m1()
{
System.out.println ("Default Method");
}
```

Interface default methods are by-default available to all implementation classes. Based on requirement we can use default method directly or we can perform override.

Example:-

```
interface Interf {
default void m1() {
System.out.println("Default Method");
}
}
```

```
class Test implements Interf
{
public static void main(String[] args)
{
Test t = new Test();
t.m1();
}
}
```

The main advantage of default methods is without effecting implementation classes we can add new functionality to the interface (backward compatibility).

Override Default Method:- In the implementation class we can provide complete new implementation or we can call any interface method as follows interface_name.super.m1();

Example:-1

```
interface Interface1 {
default void m1() {
System.out.println("Interface1 Default Method");
}
}

interface Interface2{
default void m1() {
System.out.println("Interface1 Default Method");
}
}

class Test implements Interface1, Interface2
{
public void m1()
{
System.out.println("Test Class Method"); //OR Interface1.super.m1();
}
public static void main(String[] args)
{
Test t = new Test();
t.m1();
}
}
```

Private Methods in Interface:-

Need of private Methods inside interface:

If several default methods having same common functionality then there may be a chance of duplicate code (Redundant Code).

Eg:

```
public interface DataBaseLogging
{
//Abstract Methods List
default void loginInformation(String message)
{
Step1: Connect to DataBase
Setp2: Login Information Message
Setp3: Close the DataBase connection
}
default void loginWarning(String message)
{
Step1: Connect to DataBase
Setp2: Login Warning Message
Setp3: Close the DataBase connection
}
default void loginError(String message)
{
Step1: Connect to DataBase
Setp2: Login Error Message
Setp3: Close the DataBase connection
}
}
```

In the above code all log methods having some common code, which increases length of the code and reduces readability. It creates maintenance problems also. To overcome these problems from java 1.9 version we used a private method inside the interface.

Declare private Methods inside interface:-

The repeated code or common code of default method we declare inside private method and we can call that private method from every default method.

```
public interface DataBaseLogging
{
//Abstract Methods List
default void loginInformation(String message)
{
log(message,"INFO");
}
default void loginWarning(String message)
{
```

```
log(message,"WARN");
}
default void loginError(String message)
{
log(message,"ERROR");
}
private void log(String msg)
{
Step1: Connect to DataBase
Step2: Log Message with the Provided logLevel
Step3: Close the DataBase Connection
}
}
```

Demo Program for private instance methods inside interface:-

private instance methods will provide code reusability for default methods.

```
interface Interface1
{
default void m1()
{
m3();
}
default void m2()
{
m3();
}
private void m3()
{
System.out.println("common functionality of methods m1 & m2");
}
}

class Test implements Interface1
{
public static void main(String[] args)
{
Test t = new Test();
t.m1();
t.m2();
//t.m3(); ==>CE
}
}
```

Output:-

```
C:\Users\Admin\Desktop>java Test
common functionality of methods m1 & m2
common functionality of methods m1 & m2
```