Interfaces

Interfaces: Creating Interface, Extending Interface, Interface Vs Abstract class.

Interface in java

We know the abstract class which is used for achieving partial abstraction. Unlike abstract class an interface is used for full abstraction.
Abstraction is a process where you show only "relevant" data and "hide" unnecessary details of an object from the user.

Or
An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

What is an interface in Java?
Interface looks like a 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.

What is the use of interface in Java?
As mentioned above they are used for full abstraction. Since methods in interfaces do 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 allow you to extend more than one class, However you can implement more than one interfaces in your class.

Syntax:
Interfaces are declared by specifying a keyword "interface". E.g.:

```
interface MyInterface
{
   /* All the methods are public abstract by default
    * As you see they have no body
    */
   public void method1();
public void method2(); }
```

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

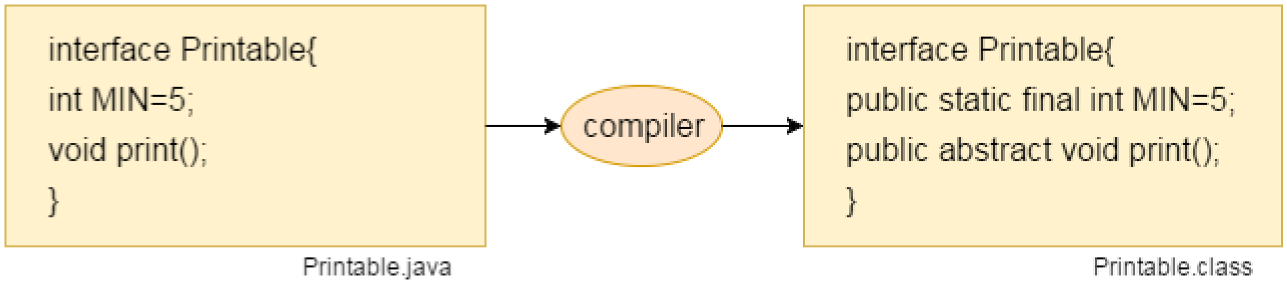
```
interface <interface_name>{

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

internal addition by the compiler

The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.

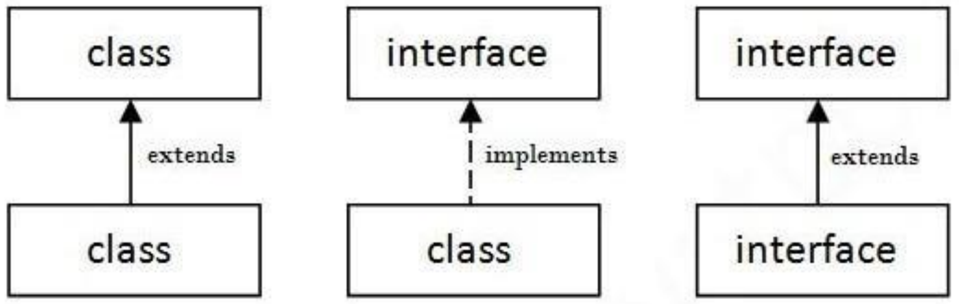


```
interface Printable{
int MIN=5;
void print();
}
```

```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Example of an Interface in Java

This is how a class implements an interface. It has to provide the body of all the methods that are declared in interface or in other words you can say that class has to implement all the methods of interface.

Do you know? class implements interface but an interface extends another interface.

```
interface MyInterface
{
   /* compiler will treat them as:
    * public abstract void method1();
    * public abstract void method2();
    */
   public void method1();
   public void method2();
}
class Demo implements MyInterface
{
   /* This class must have to implement both the abstract methods
    * else you will get compilation error
    */
   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 Demo();
         obj.method1();
   }
}
```

Output:

```
implementation of method1
```

Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
interface printable{
void print();
}

class A6 implements printable{   public void
print(){System.out.println("Hello");}     public static
void main(String args[]){
A6 obj = new A6();
obj.print();
```

```
 }
}
```

Output:

```
Hello
```

Java Interface Example: Drawable

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.

File: TestInterface1.java

```java
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDra
wable()
d.draw();
}}
```

Output:

```
drawing circle
```

Java Interface Example: Bank

Let's see another example of java interface which provides the implementation of Bank interface.

File: TestInterface2.java

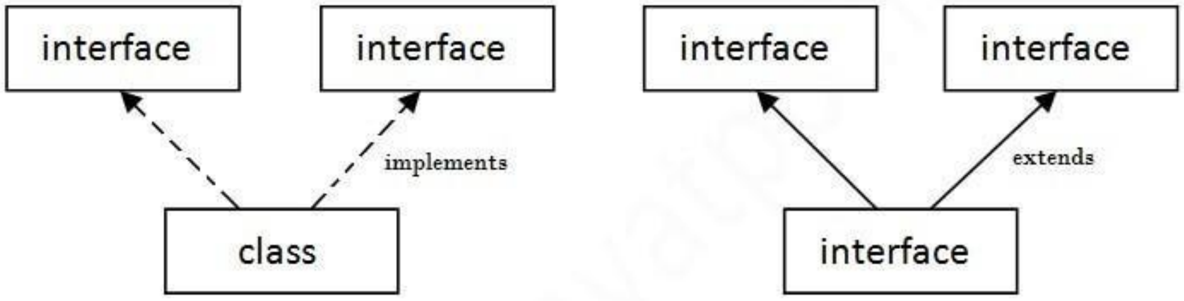
```
interface Bank{   float
rateOfInterest();
}
class SBI implements Bank{   public float
rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{   public
float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```

Output:

```
ROI: 9.15
```

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

```
Output:Hello
       Welcome
```

Q) Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class. For example:

```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }
}
```

Output:

```
Hello
```

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

Program source code :

```
package multipleInheritancebyInterfaces;

public interface Home
{
  void homeLoan();
 }
public interface Car
{
  void carLoan();
}
public interface Education
{
  void educationLoan();
 }
public class Loan implements Home, Car, Education
{
// Multiple inheritance using multiple interfaces.
public void homeLoan()
  {
     System.out.println("Rate of interest on home loan is 8.5%");
  }
public void carLoan()
{
   System.out.println("Rate of interest on car loan is 9.25%");
 }
public void educationLoan()
{
   System.out.println("Rate of interest on education loan is 10.45%");
 }
public static void main(String[] args)
{
   Loan l = new Loan();
l.homeLoan();
    l.carLoan();
    l.educationLoan();
  }
}
```

Output:

```
       Rate of interest on home loan is 8.5%
       Rate of interest on car loan is 9.25%
       Rate of interest on education loan is 10.45%
```

The above program contains three interfaces such as Home, Car, and Education. The class Loan inherits these three interfaces. Thus, we can achieve multiple inheritance in java through interfaces.

Example:

```
package multipleInheritancebyInterface;  public
interface AA
{
  void m1();
 }
public interface BB
{
  void m1();
}
public class Myclass implements AA, BB
{
  public void m1()
  {
     System.out.println("Hello Java");
  }
public static void main(String[] args)
```

```
{
  Myclass mc = new Myclass();      mc.m1();
 }
```

```
        }
```

As you can see in the above example, AA and BB interface have the same method name but there will be no confusion regarding which method is available to the implementation class.

Since both methods in interfaces have no body, and the body is provided in the implementation class i.e, its implementation is provided by class Myclass. Thus, we can use methods of AA and BB interfaces without any confusion in a subclass.

Interface inheritance

A class implements an interface, but one interface extends another interface.

```
interface Printable{
void print();
}
interface Showable extends Printable{   void
show();
}
class TestInterface4 implements Showable{   public
void print(){System.out.println("Hello");}   public
void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Output:

```
Hello
Welcome
```

Interface and Inheritance

As discussed above, an interface can not implement another interface. It has to extend the other interface. 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 Inf2 as well as Inf1.

```
interface Inf1{
  public void method1();
}

interface Inf2 extends Inf1 {
  public void method2();
}

public class Demo implements Inf2{
  /* Even though this class is only implementing the
* interface Inf2, it has to implement all the methods
* of Inf1 as well because the interface Inf2 extends Inf1
   */
   public void method1(){
          System.out.println("method1");
   }
   public void method2(){
          System.out.println("method2");
   }
   public static void main(String args[]){
          Inf2 obj = new Demo();
          obj.method2();
   }
}
```

In this program, the class Demo only implements interface Inf2, however it has to provide the implementation of all the methods of interface Inf1 as well, because interface Inf2 extends Inf1.

Why default methods?

Let's take a scenario to understand why default methods are introduced in Java.

Suppose, we need to add a new method in an interface.

We can add the method in our interface easily without implementation. However, that's not the end of the story. All our classes that implement that interface must provide an implementation for the method.

If a large number of classes were implementing this interface, we need to track all these classes and make changes in them. This is not only tedious but error-prone as well.

To resolve this, Java introduced default methods. Default methods are inherited like ordinary methods.

Let’s take an example to have a better understanding of default methods.

java 8 Default Method in Interface

Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

File: TestInterfaceDefault.java **Example :**

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

Output:

```
drawing rectangle
default method
```

Example : Default Method

```
interface      Polygon   {
void getArea();
   default void getSides() {
      System.out.println("I can get sides of polygon.");
}
}

class Rectangle implements Polygon {
   public void getArea() {
int length = 6;       int
breadth = 5;
      int area = length * breadth;
      System.out.println("The area of the rectangle is "+area);
   }

   public void getSides() {
      System.out.println("I have 4 sides.");
}
}

class Square implements Polygon {
   public void getArea() {
int length = 5;
      int area = length * length;
      System.out.println("The area of the square is "+area);
}
}
```

```
class Main {
  public static void main(String[] args) {
    Rectangle r1 = new Rectangle();
    r1.getArea();
    r1.getSides();

    Square s1 = new Square();
    s1.getArea();
  }
}
```

Output

```
The area of the rectangle is 30
I have 4 sides
The area of the square is 25
```

In the above example, we have created an interface Polygon. Polygon has a default method getSides() and an abstract method getArea().
The class Rectangle then implements Polygon. Rectangle provides an implementation for the abstract method getArea() and overrides the default method getSides().
We have created another class Square that also implements Polygon. Here, Square only provides an implementation for the abstract method getArea().

Java 8 Static Method in Interface

Since Java 8, we can have static method in interface. Let's see an example:

File: TestInterfaceStatic.java

```
interface Drawable{
void draw();
static int cube(int x){return x*x*x;}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}

class TestInterfaceStatic{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}}
```

Output:

```
drawing rectangle
27
```

Nested Interface in Java

Note: An interface can have another interface which is known as a nested interface. We will learn it in detail in the nested classes chapter. For example:

```
interface printable{
 void print();
 interface MessagePrintable{
   void msg();
 }
}
```

Nested interfaces

An interface which is declared inside another interface or class is called nested interface. They are also known as inner interface. For example Entry interface in collections framework is declared inside Map interface, that's why we don' use it directly, rather we use it like this: `Map.Entry`.

extends Keyword in Interface

Similar to classes, interfaces can extend other interfaces. The `extends` keyword is used for extending interfaces. For example,

```
interface Line {
   //members of Line interface
}

interface Polygon extends Line {
   //members of Polygon interface and Line interface
}
```

In the above example, the interface extends the interface. Now, if a class implements `Polygon` of `Polygon`le implement `Line` both `Line` and Polygon. classes

implementing multiple interfaces. For example, nd mult

```
interface A {
   ...
} interface B {
   ...
}

Interface C extends A, B {
   ...
}
```

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.
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
```

Example

```
// Filename: Sports.java public
interface Sports {
   public void setHomeTeam(String name);
public void setVisitingTeam(String name);
}

// Filename: Football.java
public interface Football extends Sports {
public void homeTeamScored(int points);    public
void visitingTeamScored(int points);    public void
endOfQuarter(int quarter);
}

// Filename: Hockey.java
public interface Hockey extends Sports {
   public void homeGoalScored();    public void
visitingGoalScored();    public void endOfPeriod(int period);
public void overtimePeriod(int ot);
}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Abstract Class vs Interface in Java 1.
Keyword(s) used:
a. Two keywords abstract and class are used to define an abstract class.
b. Only one keyword interface is used to define an interface.

2. **Keyword used by implementing class:**
a. To inherit the abstract class, we use the extends keyword.
b. To implement an interface, we can use the implements keyword.

3. **Variables:**
a. Abstract class can have final, non-final, static, and non-static variables.
b. Interface cannot have any instance variables. It can have only static variables.

4. **Initialization:**
a. The abstract class variable does not require to perform initialization at the time of declaration.
For example:

```
    public abstract class A {
int x; //  No error.
    }
```

b. Interface variable must be initialized at the time of declaration otherwise we will get compile time error.
For example: interface A { int x; // Compile time error because the blank final field x may not have been initialized.
}

5. **Method:**
a. Every method present inside an interface is always public and abstract whether we are declaring or not. That's why interface is also known as pure (100%) abstract class.
b. An abstract class can have both abstract and non-abstract (concrete) methods.

6. **Constructors:**
a. Inside an interface we cannot declare/define a constructor because the purpose of constructor is to perform initialization of instance variable but inside interface every variable is always static. Therefore, inside interface, the constructor concept is not applicable and does not require.

b. Since abstract class can have instance variables. Therefore, we can define constructors within the abstract class to initialize instance variables.

7. **Static and Instance blocks:**
a. We cannot declare instance and static blocks inside an interface. If you declare them, you will get compile time error.
b. We can declare instance and static blocks inside abstract class.

8. **Access specifiers:**
a. We cannot define any private or protected members in an interface. All members are public by default.
b. There is no restriction in declaring private or protected members inside an abstract class.

9. **Single vs Multiple inheritance:**
a. A class can extend only one class (which can be either abstract or concrete class).
b. A class can implement any number of interfaces.

10. **Default Implementation:**
a. An abstract class can provide a default implementation of a method. So, sub classes of an abstract class can just use that definition but sub classes cannot define that method.

b. An interface can only declare a method. All classes implementing interface must define that method.

11. **Difficulty in making changes:**
a. It is not difficult to make changes to the implementation of the abstract class. For example, we can add a method with default implementation and the existing subclass cannot define it.

b. It is not easy to make changes in an interface if many classes already implementing that interface. For example, suppose you declare a new method in interface, all classes implementing that interface will stop compiling because they do not define that method.

12. **Uses:**

a. If you do not know anything about the implementation. You have just requirement specification then you should go for using interface.

b. If you know about implementation but not completely (i.e, partial implementation) then you should go for using abstract class.

Abstract class	**Interface**
1) Abstract class can **have abstract**	Interface can have **only**
and non-abstract methods.	**abstract** methods. Since Java 8, it can have **default and static methods** also.
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.

9)**Example:**
public abstract class Shape{ public abstract void draw();
}
	Example:
	public interface Drawable{ void draw();
	}

Following are extra topics and example programs regarding interfaces

Java Interface Example Programs
Let's create a program where multiple classes implement the same interface to use constant values declared in that interface.
Program source code 1:

```
package interfacePrograms;  public
interface ConstantValues
{
// Declaration of interface variables.
   int x = 20;
int y = 30;
 }
```

```
public class Add implements ConstantValues
{
  int a = x;
int b = y;
 void m1()
 {
   System.out.println("Value of a: " +a);
   System.out.println("Value of b: " +b);
  }
void sum()
{
  int s = x + y;
  System.out.println("Sum: " +s);
 }
}
public class Sub implements ConstantValues
{
  void sub()
  {
    int p = y - x;
    System.out.println("Sub: " +p);
  }
}
public class Test
{
  public static void main(String[] args)
  {
    Add a = new Add();
a.m1();
     a.sum();
   Sub s = new Sub();
s.sub();
   }
}
```

```
Output:
       Value    of    a:    20
Value of b: 30
       Sum: 50
       Sub: 10
```

Let’s create a program where class B implements an interface A.

Program source code 2:

```
package interfacePrograms;
public interface A  {

  void msg(); // No body.
}
public class B implements A
{
 // Override method declared in interface.
public void msg()
    {
      System.out.println("Hello Java");
    }
 void show()
 {
    System.out.println("Welcome you");
  }
public static void main(String[] args)
{
   B b = new B();
b.msg();
    b.show(); // A reference of interface is pointing to objects of class B.

   A a = new B();
a.msg();
 // a.show(); // Compile-time error because a reference of interface can only call methods declared in it and implemented by implementing class. show() method is not part of interface. It is part of class B. When you will call this method, the compiler will give a compile-time error. It can only be called when you create an object reference of class B.
   }
 }
```

Output:

```
       Hello Java
       Welcome you
       Hello Java
```

Multilevel Inheritance by Interface

Let’s make a program where we will create a multilevel inheritance by interface. One interface extends an interface, that interface extends another interface, and a class implements methods of all interfaces, we can achieve multilevel inheritance by interface.

Program source code 4:

```
package interfacePrograms;
public interface Continent
{
  void showContinent();
 }
public interface Country
{
  void showCountry();
 }
public interface State
{
  void showState();
}
public class City implements State
{
  public void showContinent()
  {
    System.out.println("Asia");
  }
public void showCountry()
{
  System.out.println("India");
 }
public void showState()
{
   System.out.println("Jharkhand");
 }
void showCity()
{
  System.out.println("Dhanbad");
 }
public static void main(String[] args)
{
  City c = new City();     c.showContinent();
   c.showCountry();
   c.showState();
   c.showCity();
  }
}
```

Output:

```
Asia
India
```

```
Jharkhand
Dhanbad
```

As you can observe that class City implements interface State. The interface State is inherited from interface Country and Country is inherited from interface Continent, class City also implements Country and Continent interfaces even though they are not explicitly included after the colon in the declaration of class City. The methods of all interfaces have been implemented in class City.

Java 8 Default method in Interface
Since Java 1.8 or above, Default methods are allowed where we can have method body in interface. But we will need to make it default method. Let's see an example program related to it.

Let us take an example program where we will declare a default method with a body in interface.
Program source code 7:

```
package interfacePrograms;
public interface AA
{
  void m1();
  default void m2()
  {
    System.out.println("default method");
   }  }
public class BB implements AA
{
  public void m1()
  {
     System.out.println("m1-AA");
  }
public static void main(String[] args)
{
   AA obj = new BB();
    obj.m1();
obj.m2();
  }
}
```

Output:

```
m1-AA
default method
```

java 8 Static Method in Interface
Since Java 1.8 or above, static methods are allowed in interface. Let's see an example.

Program source code 8:

```
package interfacePrograms;
public interface Perimeter
{    void msg();
  static int peri(int l, int b)
  {
     return 2*(l + b);
  }
}
public class Rectangle implements Perimeter
{
  public void msg()
  {
     System.out.println("Perimeter of rectangle");
  }
public static void main(String[] args)
{
   Perimeter p = new Rectangle();
p.msg();     int perimeter =
Perimeter.peri(20,30);
   System.out.println(perimeter);
  }
}
```

Output:

```
        Perimeter of rectangle 100
```

Can We have Interface without any Methods or Fields?
An interface without any fields or methods is called **marker interface**. There are several built-in Java interfaces that have no method or field definitions. For example, Serializable, Cloneable, and Remote all are marker interfaces. They act as a form of communication among class objects.

Final words
Hope that this tutorial has covered almost all important points related to **interface in java** with example programs. I hope that you will have understood java interface and enjoyed its programming.
Thanks for reading!!!
Next ⇒ Use of Interface in Realtime Application

Java Abstract Class Example Program Programs source code 1:
Let’s create an example program where we will deal with all the above points of abstract class in java.

```
package com.abstractProgram;  public abstract class AbstractClass { // Two
keyword used: Abstract & class.

// Declaration of final, non-final, static and instance variables.
int a; // Not require initialization.
```

```
    final int b = 20; // Final variable.
static int c = 30; // static variable.

// Declaration of abstract and non-abstract methods.
abstract void m1();     static void m2()
   {
      System.out.println("Static method in abstract class");
}
// Default implementation of instance method.
void m3() { // Concrete method.
     System.out.println("Instance method in abstract class");
    }
// Declaration of constructors to initialization of instance variable.
AbstractClass()
    {
      int a = 10;
      System.out.println("Value of a; "+a);
}
// Declaration of static and non-static blocks.
   static {
      System.out.println("Static block in abstract class");
   }
{
    System.out.println("Instance block in abstract class");
}
// Declaration of private & protected members.
private void m4()
   {
      System.out.println("Private method");
    }
   protected void m5()
   {
     System.out.println("Protected method");
   }  }
public class A extends AbstractClass
{
  void m1()
{
     System.out.println("Implementation of abstract method");
  }
}
public class AbstractTest
{
  public static void main(String[] args)
  {
    A a = new A();
   System.out.println("Value of b: " +a.b);
   System.out.println("Value of c: " +AbstractClass.c);
a.m1();
   AbstractClass.m2();
a.m3();
    a.m5();
  }
}
```

Output:

```
      Static block in abstract class
      Instance block in abstract class
      Value of a: 10
      Value of b: 20
      Value of c: 30
      Implementation of abstract method
      Static method in abstract class
      Instance method in abstract class
      Protected method
```

java Interface Example Program

Program source code 2:
Let’s take a simple example program to understand all the important points of interface.

```
package interfaceProgram;  public interface AA
{ // One keyword: interface.
  int x = 20; // Interface variable must be initialized at the time of declaration. By
default interface variable is public, static and final.
  void m1(); // By default, Interface method is public and static.

// Here, we cannot declare instance variables, instance methods, constructors, static,
and non-static block.
  }
public interface BB
{
  int   y   =   20;
void m2();
 }  public class CC implements AA, BB { // Multiple
Inheritance.

 public void m1()
 {
   System.out.println("Value of x: " +x);
   System.out.println("m1 method");
```

```
  }
public void m2()
{
   System.out.println("Value of y: " +y);
   System.out.println("m2 method");
 }  }
public class MyClass
{
  public static void main(String [] args)
  {
    CC c = new CC();
c.m1();
     c.m2();
  }
}
```

```
Output:
       Value of x: 20
m1 method
Value of y: 20
       m2 method
```

Final words

Hope that this tutorial has covered almost all important points related to **differences between abstract class and interface in java** with example programs. I hope that you will have liked this tutorial. Thanks for reading!!!

Difference Between Abstract Class and Interface in Java

In this article, we will discuss the **difference between Abstract Class and Interface in Java with examples**. I have covered the abstract class and interface in separate tutorials of OOPs Concepts so I would recommend you to read them first, before going though the differences.

1. Abstract class in java
2. Interface in Java

	Abstract Class	**Interface**
1	An abstract class can extend only one class or one abstract class at a time	An interface can extend any number of interfaces at a time
2	An abstract class can extend another concrete (regular) class or abstract class	An interface can only extend another interface
3	An abstract class can have both abstract and concrete methods	An interface can have only abstract methods
4	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
5	An abstract class can have protected and public abstract methods	An interface can have only have public abstract methods
6	An abstract class can have static, final or static final variable with any access specifier	interface can only have public static final (constant) variable

Each of the above mentioned points are explained with an example below:

Abstract class vs interface in Java

Difference No.1: Abstract class can extend only one class or one abstract class at a time

```java
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 Example3{
   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 extended(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
```

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();
   }
}
```

Output:

```
display1 method
```

Difference No.4: 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.5: Abstract class can have protected 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

```
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.6: 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 public static final (constant) variable

```
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();
   }
}
```

Difference between abstract class and interface Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Example of abstract class and interface in Java

Let's see a simple example where we are using interface and abstract class both.

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

```

```java
//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Output:

```
I am a
I am b
I am c
I am d
```

Key points about interfaces: Here are the key points to remember about interfaces:

1) We can't instantiate an interface in java. That means we cannot create the object of an interface
2) Interface provides full abstraction as none of its methods have body. On the other hand abstract class 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 that implements any interface must implement all the methods of that interface, else the class should be declared 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 throw 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 args[])
{
    x=20; //compile time error
 }
}
```

11) An interface can extend any interface but cannot implement it. Class implements interface and interface extends interface.
12) A **class** can implement any **number of interfaces**.
13) If there are **two or more same methods** in two interfaces and a class implements both interfaces, implementation of the method once 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 args[])
  {
      //Statements
   }
}
```

14) A class cannot implement two interfaces that have methods with same name but different return type.

```
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 args[])
   {

   }
}
```

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

```
interface A {     int x=10; } interface B {     int x=100; }
class Hello implements A,B
{
    public static void Main(String args[])
    {
       /* reference to x is ambiguous both variables are x
* so we are using interface name to resolve the
* variable
        */
       System.out.println(x);
       System.out.println(A.x);
       System.out.println(B.x);
    }
}
```