Abstract Class

Example1

```
import java.util.*;
        abstract class Shape1
        { int a,b;
             Shape1(int x,int y)
           { a=x;
             b=y;
           }
        abstract void area();
         // abstarct method  {System.out.println("Area not defined");}
        }
        class Rectangle1 extends Shape1
        {       Rectangle1(int x,int y)
                   {  super(x,y);     }
        void area() {System.out.println("Area of Rectangle=="+a*b);}
        }
        class Triangle1 extends Shape1
        {     Triangle1(int x,int y)
               {  super(x,y);     }
        void area() {System.out.println("Area of  Triangle=="+0.5*a*b);}
        }
        class Abstract1{
        public static void main(String ar[])
                {   Scanner sc= new Scanner(System.in);
                    System.out.println("Enter triangle sides");
                    int a=sc.nextInt();
                    int b=sc.nextInt();
         Triangle1 t= new Triangle1(a,b);
          System.out.println("Enter Rectangle sides");
          int c=sc.nextInt();
          int d=sc.nextInt();
          Rectangle1 r=new Rectangle1(c,d);
            t.area();
            r.area();
      }
}
```

Example2

```
import java.util.*;

abstract class ShapeTest

{ double a,b;

          ShapeTest(double x,double y)

                 {  a=x;

                     b=y;     }

         abstract void area();

}

class RectangleTest extends ShapeTest

    {       RectangleTest(double x, double y)

              {  super(x,y);     }

           void area() {System.out.println("Area of Rectangle=="+a*b);}
    }
class TriangleTest extends ShapeTest
    {     TriangleTest(double x, double y)
            {  super(x,y);     }
        void area() {System.out.println("Area of  Triangle=="+0.5*a*b);}
   }
class CircleTest extends ShapeTest
   {       CircleTest(double x, double y)
           {  super(x,y);     }
        void area() {System.out.println("Area of  Circle=="+a*b*b);}
   }
class Abstract2{  public static void main(String ar[])
         {   Scanner sc= new Scanner(System.in);
              System.out.println("Enter triangle sides");
             double a=sc.nextDouble();
             double b=sc.nextDouble();
             TriangleTest tr= new TriangleTest(a,b);
              System.out.println("Enter Rectangle sides");
```

```
                 double l=sc.nextDouble();

                 double w=sc.nextDouble();

                 RectangleTest rt=new RectangleTest(l,w);

                System.out.println("Enter vale of Pi and Radius of circle");

                double p=sc.nextDouble();

                double r=sc.nextDouble();

                CircleTest   cr=new CircleTest(p,r);

                  tr.area();

                  rt.area();

                  cr.area();
            }}
```

Example3

```
    abstract class Sum

     {

      /* These two are abstract methods, the child class

       * must implement these 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......");
```

```
  }
}
//Regular class extends abstract class
class AbstractSum extends Sum{
  /* If I don't provide the implementation of these two methods, the
   * program will throw compile time error.
   */
  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[]){
       Sum obj = new AbstractSum();
       System.out.println(obj.sumOfTwo(3, 7));
       System.out.println(obj.sumOfThree(4, 3, 19));
       obj.disp();
  }
}
```

Abstract Class Constructor

```
abstract class Base {
            Base()
            {
                System.out.println("Base Constructor Called");
            }
            abstract void fun();
        }
class Derived extends Base {
            Derived()
            {
                System.out.println("Derived Constructor Called");
            }
            void fun()
            {
                System.out.println("Derived fun() called");
            }
        }
class AbstractConstructor {
            public static void main(String args[])
            {
                Derived d = new Derived();
                        d.fun();
            }
        }
```

Interface

Use of interface: 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();
}
```

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.

```
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()
```

MyInterface
Demo

```
  {
          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

Note

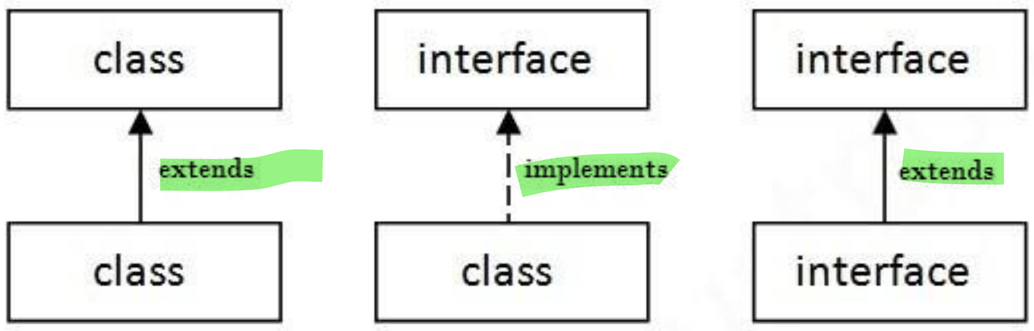
- We can’t instantiate an interface in java. That means we cannot create the object of an interface.
- Interface provides full abstraction as none of its methods have body. On the other hand, abstract class provides partial abstraction as it can have abstract and concrete (methods with body) methods both.
- **implements** keyword is used by classes to implement an interface.
- While providing implementation in class of any method of an interface, it needs to be mentioned as public.
- Class that implements any interface must implement all the methods of that interface, else the class should be declared abstract.
- Interface cannot be declared as private, protected or transient.
- All the interface methods are by default abstract and public.
- Variables declared in interface are public, static and final by default.

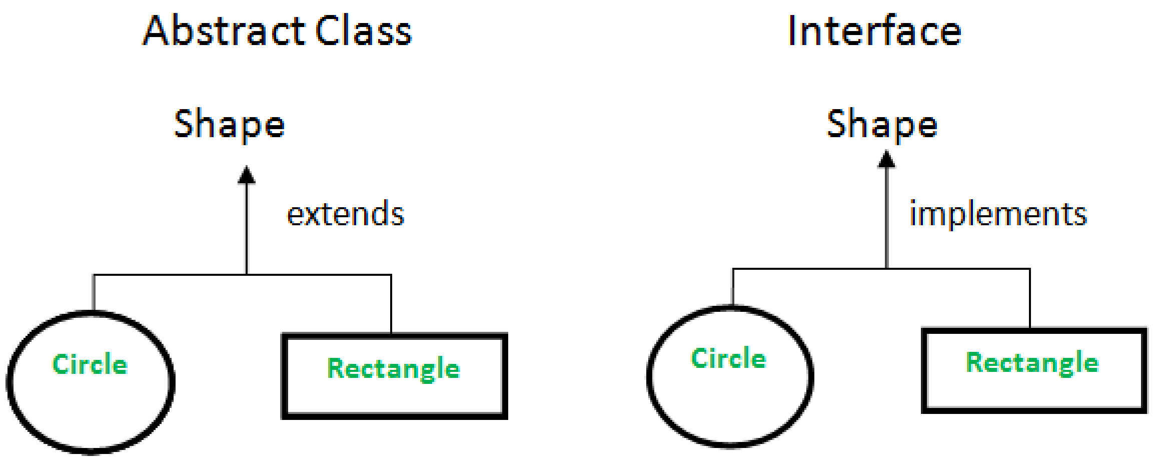
```
interface I1
{   int a=10;
   public int a=10;
   public static final int a=10;
   final int a=10;
```

```
              static int a=10;  }
```
All of the above statements are identical.

```
interface  MyInterface
          {
                int NO=10;
                 void method();
          }
```
OR

```
interface  MyInterface
        {
            public static final int NO=10;
            public abstract void method();
        }
```
Above both are equivalent.





- 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.

- 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 cannot be re-initialized.

```
class Sample implements Try
{
  public static void main(String args[])
  {
    x=20;                              //compile time error
  }
}
```

- An interface can extend any interface but cannot implement it. Class implements interface and interface extends interface.

- A class can implement any number of interfaces.

- 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 One
{
   public void aaa();
}
interface Two
{
   public void aaa();
}
class InterfaceTest2 implements One,Two
```

```
{
  public void aaa()
  {
      System.out.println("AAA");
  }
  public static void main(String args[])
  {
      InterfaceTest2 it=new InterfaceTest2();
              it.aaa();
   }
}
```

- A class can not 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()                    // CT error
  {
  }
  public int aaa()                      // CT error
  {
  }
  public static void main (String args[])
      {
      }        // cob.aaa();
}
```

- 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);                    //ambiguous x
       System.out.println(A.x);
       System.out.println(B.x);
    }
}
```

Interface & Abstract class

```
interface A

{       void m1();

         void m2();

         void m3();

         void m4();

}

abstract class B implements A

   {    public void m3()

           {System.out.println("I am in m3() ");}

   }

class C extends B          {

public void m1(){System.out.println("I am in m1()   " ); }

public void m2(){System.out.println("I am in m2()   ");    }

public void m4(){System.out.println("I am in m4()  ");   }

                                      }
```

```
        class InterfaceAbstractDemo                {
                public static void main(String args[])   {
                                A a=new C();
                                a.m1();
                                a.m2();
                                a.m3();
                                a.m4();                    }}
```

Default Method in Interface

The concept of default method is introduced after JDK7. The general syntax of default method in interface is as follows:

[illegible]

```
        interface MyInterfaceTest
          {        default RetrunType  methodName()
                                   {   Statement(s);   }
          }
```

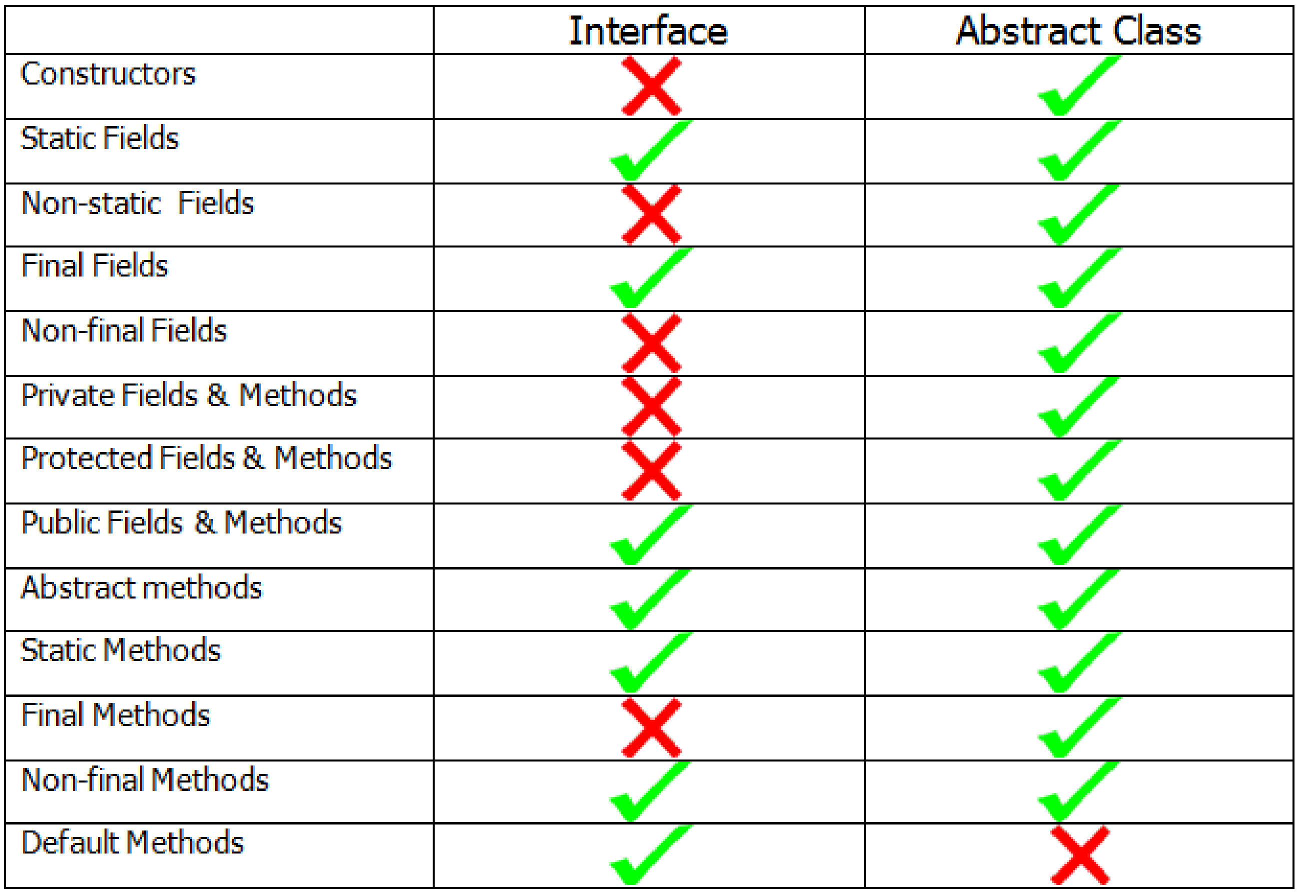
```
interface MyInterfaceTest
          {      void display();
                  default void defaultMethod()
                          {System.out.println("This is Default Method ");
                                                         }
          }
    class  DefaultMethod implements MyInterfaceTest
          {     public void display()
                     {System.out.println("  display method def");    }
```

```
        }

    class DefaultMethodDemo

      {           public static void main(String args[])

                 {          DefaultMethod dm  =new DefaultMethod();

                                      dm.display();

                                     dm.defaultMethod() ;

                 }

      }
```

Sr. No	**Abstract**	**Interface**
1	An abstract class can extends only one class or one abstract class at a time	An interface can extends 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 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
7	Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
8	Abstract class can have static methods, main method and constructor.	Interface can have static methods, main method but NO constructor.
9	Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
10	The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.

11	General Syntax:
	public abstract class **ClassName**
	{
	public abstract void methodName(argument(s));
	}
		General Syntax:
		public interface **InterfaceName**
		{
		void abstarctMethodName(arguments(s));
		}

Summary of Abstract class and interface



	Interface	Abstract Class
Constructors	✗	✓
Static Fields	✓	✓
Non-static Fields	✗	✓
Final Fields	✓	✓
Non-final Fields	✗	✓
Private Fields & Methods	✗	✓
Protected Fields & Methods	✗	✓
Public Fields & Methods	✓	✓
Abstract methods	✓	✓
Static Methods	✓	✓
Final Methods	✗	✓
Non-final Methods	✓	✓
Default Methods	✓	✗

Use of Final Keyword in Java

The final keyword in java is used to restrict the user. The java final keyword can be used in many contexts as following.

Final Variable: The final keyword can be applied with the variables; the value of final variable ca not be changed. A final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

- **The blank final variable can be initialized only in constructor.**

```
  class BlankFinalVariable
 {
    final int finalvar;                    //blank final variable
          BlankFinalVariable()
                {
                 finalvar=50;
                 System.out.println(finalvar);
             }
         public static void main(String args[])
            {
              new BlankFinalVariable();
          }
 }
```

- A **static final variable** that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

```
class StaticFinalVariable
{
  static final int sfvar;                     //static blank final variable
  static   { sfvar=20;}
  public static void main(String args[])
    {
    System.out.println(StaticFinalVariable.sfvar);
 }
}
```

- We cannot change the value of any **parameter declared as final**.

```
class FinalParameter
  {
   int cube(final int n){
   //n=n+2;                                  // can't change
   return(n*n*n);
  }
  public static void main(String args[])
    {
    FinalParameter obj=new FinalParameter();
    int x=obj.cube(4);
```

```
            System.out.println("Cube=="+x);     }  }
```

Using final to Prevent Overriding: Methods declared as final cannot be overridden. The following fragment illustrates final:

```
class A {
final void meth() {
System.out.println("This is a final method.");
}
}
class B extends A {
void meth() {                              //CT ERROR! Can't override.
System.out.println("Illegal!");
}
}
```

Using final to Prevent Inheritance: Sometimes you will want to prevent a class from being inherited. To do this, precede the class declaration with final. Declaring a class as final implicitly declares all of its methods as final, too. As you might expect, it is illegal to declare a class as both abstract and final since an abstract class is incomplete by itself and relies upon its subclasses to provide complete implementations.

Here is an example of a final class:

```
      final class A {
              //   Statement(s)
             }
   // The following class is illegal.
   class B extends A {                  //CT ERROR! Can't subclass A
             //          Statement(s)
             }
```

As the comments imply, it is illegal for B to inherit A since A is declared as final.

- **The final method is inherited but cannot be overridden it.**

```
class  Parent{
 final void display1(){System.out.println("Parent class method is running...");}
                    }
class Child extends Parent
          {      void display2()
                       {   System.out.println("child class method is running...");   }
          }
class FinalMethodInheritanceDemo
          {
            public static void main(String args[])
            {
                          //new Child().display1();
                          //new Child().display2();
                                          // OR
                           Child obj=new Child();
                            obj.display1();
                            obj.display2();
            }
          }
```