Abstract Class

The class which is declared using abstract as a keyword is known as an abstract class. The class which is not declared with abstract as keyword is known as a concrete class.

The method which has only definition and no declaration is known as an abstract method. The method which has both definition and declaration is known as a concrete method.

Inside an abstract class we can declare both abstract and concrete method. Inside concrete class we can declare only concrete methods.

Inside abstract class we can declare static, non-static, blocks and constructors. Abstract method cannot be instantiated. If any class inherits abstract class, that class should provide definition for all abstract method of superclass or the subclass must be declared as an abstract class.

Abstract class

```
Abstract class demo
{
//abstract method
Abstract void m1();
}
```

Concrete class

```
Class demo
{
// concrete method
Void m1()
{
-----
-----
}
}
```

Abstract class	**Concrete class**
• Static members	• Static members
• Non-static members	• Non-static members
• blocks	• blocks
• Constructors	• Constructors
• Concrete methods	• Concrete methods
• Abstract methods	

Example for abstract class cannot be instantiated.

```
Abstract class demo
{
Static int x=10;
Int y=20
}
Class sample extends demo
{
Public static void main(String args[])
```

```
{
System.out.println(demo.x);
Sample s=new sample();
System.out.println(s.y);
}
}
```

For abstract class method should be overridden in sub class or change subclass as an abstract class.

```
Abstract class demo
{
Abstract void m1();
}

Class sample extends demo
{
Void m1()
{
System.out.println("M1 is running");
}
Public static void main(String args[])
{
Sample s=new sample();
s.m1();
}
}
```

Note: An abstract method is created only when we don't know the implementation for the method.

Inside abstract class, we can declare both concrete and abstract method, hence abstract class is not a pure abstract body. Using abstract class we cannot achieve pure abstraction.

1. Write a program to override an abstract method Displayeage() in subclasses

```
package com.Abstract2;

abstract public class Animal
{
abstract public void DisplayAge();
}

package com.Abstract2;

public class Dog extends Animal
{
	public void DisplayAge()
	{
		System.out.println("The age of dog is 15 years");
	}
}
package com.Abstract2;

public class Snake extends Animal
```

```
{
	public void DisplayAge()
	{
		System.out.println("The age of Snake is 10 years");
	}
}

package com.Abstract2;

public class Abstract2Main {

	public static void main(String[] args)
	{
		Snake s=new Snake();
		Dog d=new Dog();
		s.DisplayAge();
		d.DisplayAge();
	}

}
```

2. Write a program with product as an abstract class and laptop, mobile as a subclass

```
package com.Abstract1;

public abstract class Product
{
abstract void ProductDetails();
}
package com.Abstract1;

public class Mobile extends Product
{
public void ProductDetails()
{
	String Mobile_Brand="Samsung";
	String Mobile_Name="S8";
	double Mobile_Version=7.9;

	System.out.println("Modile details: ");
	System.out.println("Mobile Brand:" +Mobile_Brand);
	System.out.println("Mobile Name :" +Mobile_Name);
	System.out.println("Mobile Version:" +Mobile_Version);


}
}
package com.Abstract1;

public class Laptop extends Product
{
public void ProductDetails()
{
	String Laptop_Brand="Hp";
	String Laptop_Name="Yoga";
	double Laptop_Version=7.6;
```

```
        System.out.println("Laptop details: ");
        System.out.println("Laptop Brand:" +Laptop_Brand);
        System.out.println("Laptop Name :" +Laptop_Name);
        System.out.println("Laptop Version:" +Laptop_Version);
}
}
package com.Abstract1;

public class Abstract1_Main
{
        public static void main(String args[])
        {
                Laptop l=new Laptop();
                l.ProductDetails();
                Mobile m=new Mobile();
                m.ProductDetails();
        }

}
```

Interface

Interface is a definition block used to declare abstract methods. Inside interface we can declare only static variables and it must be final. Inside interface we cannot declare blocks and constructors. Inside interface we can declare abstract method without using abstract keyword. The default access specifiers for interface member are public.

From JDK 1.8 onwards we can declare static concrete methods inside an interface.

We cannot declare non-static concrete methods inside an interface. Interface cannot be instantiated.

Interface can be inherited, a class can inherit properties of an interface using implements keyword. A class which is inheriting interface is called as an implementing class.

One class can inherit multiple interfaces. Interface can inherit another interface using extends keyword. One interface can inherit multiple interfaces. Using interface we can achieve multiple inheritance.

Syntax to declare an interface

```
Interface interface_name
{
---
----
}
```

Interface demo
{

- Static final variables
- Abstract methods
- No blocks, contructors
- Public is default access specifier

}

```
Public interface sample
{
Static int x=20;
Int y=30; // default its static
Abstract void m1();
Void m1();// default  its abstract
/*
Sample()
{
//error constructor is not allowed
}
{
// error non-static block is not allowed
}
Static
{
// error static block is not allowed
}
```

1. Implementing a interface

```
public interface A
{
static final int x=10;
public abstract void m1();
}
package com.Interface1;
public class InterfaceMain1 implements A
{
	public void m1()
	{
		System.out.println("M1 is running");
	}

public static void main(String args[])
{
	// A a=new A(); cannot instantiate an interface
	InterfaceMain1 d=new InterfaceMain1();
	d.m1();
	System.out.println(A.x);
}
}
```

Syntax for implementation:

Public class class_name implenets interface_name1, interface_name2,… interface_namen

2. One class can inherit properties of multiple interfaces

```
package com.Interface2;
public interface A
{
	static int x=20;
	void m1();
}
package com.Interface2;
```

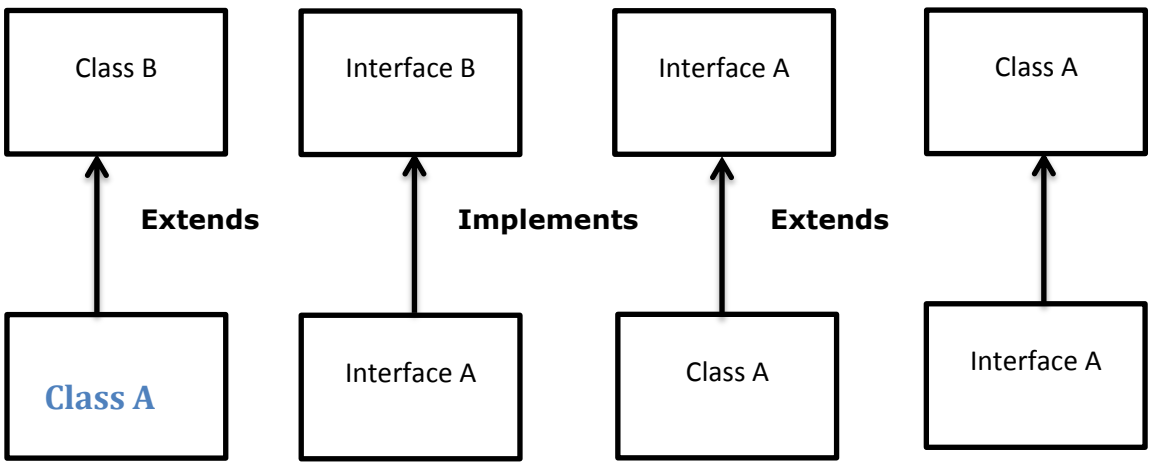
```
public interface B
{
void m2();
}
package com.Interface2;
public class InterfaceMain2 implements A,B
{
	public void m1()
	{
		System.out.println("M1 is running");
	}
	public void m2()
	{
		System.out.println("M2 is running");
	}
public static void main(String args[])
{
	InterfaceMain2 i=new InterfaceMain2();
	i.m1();
	i.m2();
}
}
```

3. One interface can extend properties of another interface using extends keyword.

```
public class InterfaceMain3 implements A
{

	public void m1()
	{
		System.out.println("m1 is running");
	}
	public void m2()
	{
		System.out.println("m2 is running");
	}

	public static void main(String[] args)
	{
		InterfaceMain3 i=new InterfaceMain3();
		i.m1();
		i.m2();
	}
public interface B
{
void m2();
}
public interface A extends B
{
	static int x=20;
	void m1();
}
```

3. One interface can inherit multiple interfaces, an interface can inherit another interface using extends keyword

```
public class InterfaceMain4 implements A
{
public void m1()
{
        System.out.println("m1 is running");
}
public void m2()
{
        System.out.println("m2 is running");
}
public void m3()
{
        System.out.println("m3 is running");
}

public static void main (String args[])
{
        InterfaceMain4 i=new InterfaceMain4();
        i.m1();
        i.m3();
        i.m2();
}
}
public interface A extends B,C
{
        static int x=20;
        void m1();
}
public interface B
{
void m2();
}
public interface C
{
public void m3();
}
```



A class consist of concrete class, hence an interface cannot extend a class. In an interface since all variables are final they all should be initialised at declaration time.

Extends and implements keyword should be used in following order:

First extends, then implements

Write difference between Abstarct class and interface

Abstract class	**Interface**
1. Declared with abstract keyword	1. Declared with interface keyword
2. Can contain blocks, constructor, concrete method and abstact method	2. Can contain only abstract method and static concrete method.
3. variables need not be static final.	3. Variables are always static final.
4. Inherited using extends keyword	4. Inherited using implements keyword.
5. Default access specifier is package	5. Default access specifier is public