자바 인터페이스 특징

1. 인터페이스는 interface 예약어로 정의한다.
2. 인터페이스는 하나 이상의 부모로부터 다중 상속이 가능하다.
3. 인터페이스에는 public static final로 인식되는 클래스(정적)상수만 올수 있다.
4. 인터페이스는 public abstract 로 인식되는 추상메서드가 올수 있다.
5. 인터페이스는 자식클래스에서 implements 키워드로 구현한다. 상속받은 자식클래스에서 부모 인터페이스의 추상메서드를 반드시 오버라이딩을 해야 한다. 그래야 자식클래스 객체 생성이 가능하다.
6. 인터페이스는 객체 생성을 못한다.

인터페이스 정의

```
접근지정자 interface 인터페이스 이름 {

        public staic final 타입(자료형) 정적상수이름;
        public abstract 리턴타입 추상메서드이름();

}
```

인터페이스에 오는 모든 변수는 **public static final**로 인식되는 **정적상수만** 올수 있다. public static final 키워드(예약어)는 생략 가능하다. 추상메서드는 {}가 없고 실행문장이 없다. 그러므로 호출이 불가능하다. 인터페이스에서 추상메서드를 정의할 때 **public abstract** 키워드로 정의한다. 물론 이 키워드는 생략이 가능하다.

인터페이스 정의 예

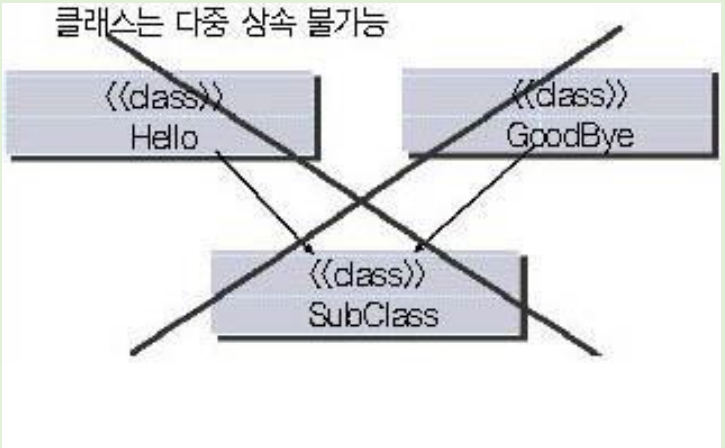
```
interface IHello{//첫번째 인터페이스 사용예
        void sayHello(String name); //public abstract 예약어가 생략됨.
 }
 interface IHello{//두번째 인터페이스 사용예
        public abstract void sayHello(String name);//추상메서드는 {}가 없고,실행문
        //장이 없어서 호출이 불가능하다. public abstract이 생략안됨.
 }
  class Hello implements IHello{//implements 키워드로 구현상속 받는다.
   public void sayHello(String name){
   //인터페이스의 추상 메서드 오버라이딩 : 접근 지정자를 반드시 public으로 해야 함
   //일반 메서드로 오버라이딩을 해서 {}와 실행문장을 넣을 수 있다. 그러면 자손클래스
   //인 Hello 객체 생성이 가능하다.
    System.out.println(name+"씨 안녕하세요!");
  }
}
```

부모 인터페이스 정의와 이를 구현한 자손 클래스 소스 예1)

```
interface IHello{ //interface 키워드로 IHello 인터페이스 정의
  void sayHello(String name); //public abstract가 생략된 추상메서드
}

class Hello implements IHello{
  public void sayHello(String name){
     System.out.println(name+"씨 안녕하세요!");
  }
}

public class InterEx01{
  public static void main(String[] args) {
    Hello obj= new Hello(); //자손클래스 Hello에서 부모 인터페이스의 모든 추상 메서드를 반드시
    //오버라이딩을 해야 new 키워드로 자손클래스 객체 생성이 가능하다.
    obj.sayHello( "홍길동" );
  }
}
```

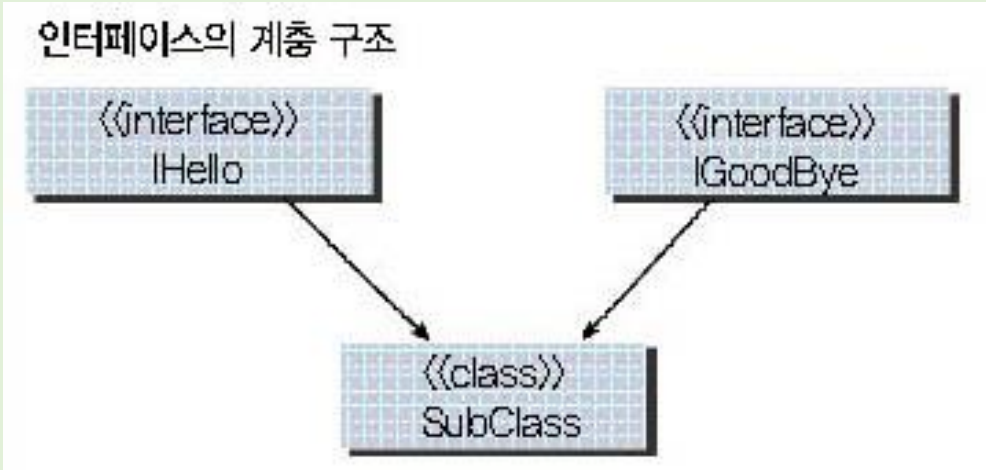
클래스는 단일상속만 가능하다를 확인하는 소스 예2)

```
abstract class Hello{ //abstract class 키워드로 추상클래스 정의
 public abstract void sayHello(String name); //추상클래스에서 추상메서드를 정의할때는 abstract키
 //워드 생략못함.
}
abstract class GoodBye{
 public abstract void sayGoodBye(String name);
}

class SubClass extends GoodBye, Hello { //추상클래스는 하나의 부모로 부터 단일상속만 가능하고, 다
//중 상속은 불가능하다.
 public void sayHello(String name){
     System.out.println(name+"씨 안녕하세요!");
 }
 public void sayGoodBye(String name){
   System.out.println(name+"씨 안녕히 가세요!");
 }
}
```



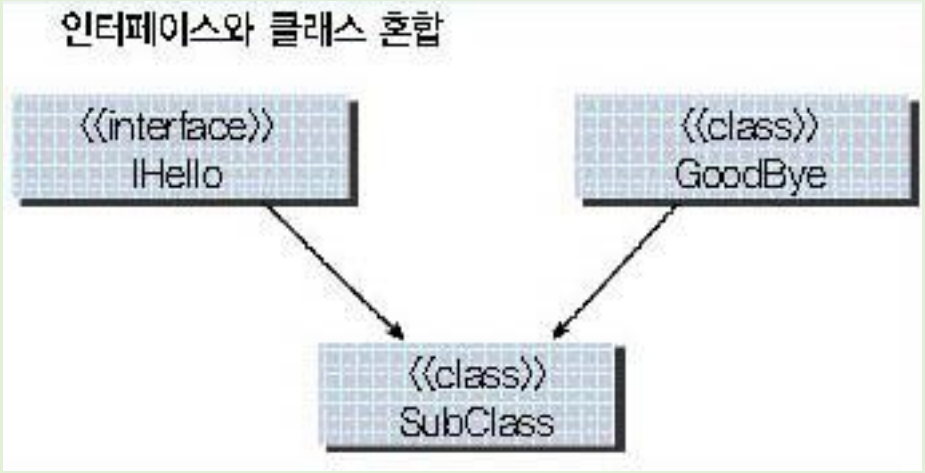
인터페이스를 이용한 다중 상속 소스 예3)

```
interface IHello{
  public abstract void sayHello(String name);
}
interface IGoodBye{
  public abstract void sayGoodBye(String name);
}
class SubClass implements IHello, IGoodBye{//두 부모 인터페이스로부터 다중상속을 받는 클래스 설계
 public void sayHello(String name){
    System.out.println(name+"씨 안녕하세요!");
 }
 public void sayGoodBye(String name){
   System.out.println(name+"씨 안녕히 가세요!");
 }
}
public class InterEx03{
 public static void main(String[] args) {
   SubClass test= new SubClass();
   test.sayHello( "홍길동" );
   test.sayGoodBye( "이순신" );
 }
}
```



부모 인터페이스와 클래스를 동시에 상속받는 소스 예4)

```
interface IHello{ //interface 키워드로 IHello 부모인터페이스 정의
  public abstract void sayHello(String name); //public abstract 이 생략가능. sayHello() 추상메서드 정의
}
abstract class GoodBye{ //abstract class 키워드로 GoodBye 부모 추상클래스 정의
  public abstract void sayGoodBye(String name); //abstract 생략 불가능. {}가 없고 실행문장이 없는 추상메서드
  //sayGoodBye()정의
}
class SubClass extends GoodBye implements IHello{ //extends 부모추상클래스(클래스) implements 부모 인터페이스
  public void sayHello(String name){
   System.out.println(name+"씨 안녕하세요!");
 }
public void sayGoodBye(String name){
   System.out.println(name+"씨 안녕히 가세요!");
 }
}
public class InterEx04{
  public static void main(String[] args) {
   SubClass test= new SubClass();
   test.sayHello( "강감찬" );
   test.sayGoodBye( "을지문덕" );
 }
}
```



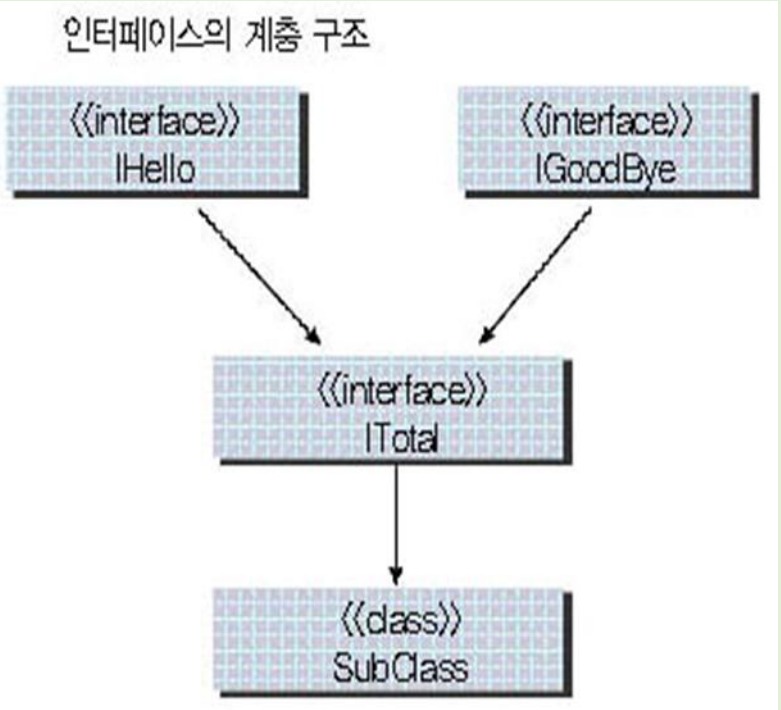
인터페이스 상속

인터페이스 선언시 필요에 따라 다른 인터페이스로부터 상속받을 수 있습니다. 인터페이스들의 상속에도 **extends 예약어**를 사용합니다.

```
interface 인터페이스_이름 extends 인터페이스_이름[, 인터페이스_이름...]{ //하나 이상의 부모 인터페
//이스로 부터 다중상속을 받을 수 있다.
   정적 상수 선언
   추상_메서드 선언
}
```

인터페이스의 다중 상속 소스 예5)

```
interface IHello{
 public abstract void sayHello(String name);
}
interface IGoodBye{
  public abstract void sayGoodBye(String name);
 }
interface ITotal extends IHello, IGoodBye{ //부모 인터페이스 다중상속
  public abstract void greeting(String name);
}

class SubClass implements ITotal{ //자손클래스에서 부모인터페이스 구현상속
  public void sayHello(String name){
    System.out.println(name+"씨 안녕하세요!");
   }
 public void sayGoodBye(String name){
    System.out.println(name+"씨 안녕가세요!");
 }
 public void greeting(String name){
    System.out.println(name + ", 안녕!");
 } //부모 인터페이스의 모든 추상메서드를 반드시 오버라이딩을 해야한다. 그래야만 자손 클래스 SubClass로 객체생
//성 가능하다.
}
```



인터페이스의 다중 상속 소스 예5)

```
public class InterEx05{
 public static void main(String[] args) {
   SubClass test= new SubClass();
   test.sayHello( “홍길동” );
   test.sayGoodBye( “이순신” );
   test.greeting( “강감찬” );
 }
}
```

인터페이스에 오는 정적상수 소스 예6)

```
//인터페이스(IColor)므로 다중 상속 가능
interface IColor{
  int RED=1;                          //정적상수(public static final 로 인식)
  public static final int GREEN=2;    //정적상수
  public static final int BLUE=3;     //정적상수 , public staic final 생략가능
  void setColor(int c);               //추상메서드 (public abstract로 인식하고 생략가능)
  public abstract int getColor();     //추상메서드
}
//클래스(AbsColor)이므로 다중 상속 불가능하고 단일 상속만 가능
abstract class AbsColor implements IColor{
  int color=GREEN;                        //일반변수도 가질 수 있디.
  public void setColor(int c){            //구현된 메서드도 가질 수 있다.
    color=c;
  }
}
class SubClass extends AbsColor{
  public int getColor(){ //추상 메서드 오버라이딩
   return color;
  }
}
```

인터페이스에 오는 정적상수 소스 예6)

```
public class InterEx06{
 public static void main(String[] args) {
    SubClass test= new SubClass();
    test.setColor(IColor.RED);
    System.out.println(test.getColor());
   }
}
```

public abstract 추상 메서드만 가질 수 있다(jdk1.7)
Jdk1.8 이후 부터는 static 메서드와 default 메서드를 가질 수 있다.

원래는 인터페이스에 자바7버전 까지는 추상메서드만 선언할 수 있었다. JDK8 이후부터는 디폴트 메서드와 static 메서드도 추가할 수 있게 되었다. static 메서드는 인스턴스와 관계가 없는 독립적인 메서드이기 때문에 예전부터 인터페이스에 추가하지 못할 이유가 없었다.
인터페이스의 static 메서드 역시 접근 제어자는 항상 public이며, 생략할 수 있다.

디폴트 메서드

부모 인터페이스에 추상메서드를 추가한다는 것은 이 인터페이스를 구현 상속한 모든 자손클래스에서 부모에 새롭게 추가된 추상메서드를 강제 오버라이딩을 해야 한다. 이러한 불편함을 해소하기 위해서 추가된 것이 default 메서드이다. 부모 인터페이스에 default메서드를 추가하면 이를 구현한 자손에서 굳이 오버라이딩을 하지 않아도 된다.
default 메서드는 접근 제어자가 publi이며 생략가능하다. 추상메서드와 달리 {}가 있고 실행문장이 있다.

주의할 사항

1. 여러 부모 인터페이스에 동일한 이름의 default 메서드가 있는 경우에는 이 부모 인터페이스를 다중 상속한 자손 클래스에서 부모의 default 메서드를 오버라이딩을 해야 한다.

2. 부모 인터페이스의 default 메서드와 부모 조상클래스의 일반 메서드명이 동일한 경우 부모 클래스 일반 메서드가 상속되고 부모 인터페이스의 default 메서드는 무시된다.

default 메서드가 추가된 이유에 관한 소스예 7)

```
interface ParentInterface {
   void abstractMethod();//public abstract가 생략된 추상메서드

   public default void defaultMethod() {//public 접근 제어자 생략가능,디폴트 메서드 정의,{}가 있고 실행문장
    System.out.println("default 메서드 in ParentInterface");
   }
}


class ChildClass implements ParentInterface {
   @Override
   public void abstractMethod() {
     System.out.println("부모 인터페이스 추상메서드 오버라이딩");
   }

  // defaultMethod()를 오버라이딩하지 않아도 됩니다. 선택적 오버라이딩
  //@Override
  //public void defaultMethod() {
   //System.out.println("default 메서드는 선택적 오버라이딩");
 //}
}
```

default 메서드가 추가된 이유에 관한 소스예 7)

```
public class InterEx07 {
	public static void main(String[] args) {
		ChildClass child = new ChildClass();
		child.abstractMethod(); //오버라이딩 한 추상메서드 호출
		child.defaultMethod();  //디폴트 메서드 호출
	}
}
```

두 부모 인터페이스에 동일한 이름의 디폴트 메서드가 정의된 경우 관한 소스예 8)

```
interface InterfaceA {
  default void display() {//디폴트 메서드 정의
    System.out.println("Display from InterfaceA");
  }
}


interface InterfaceB {
  default void display() {
    System.out.println("Display from InterfaceB");
  }
}


class ChildClass08 implements InterfaceA, InterfaceB {
  @Override
  public void display() {//두 부모 인터페이스에 동일한 이름의 디폴트 메서드가 정의된 경우 자손클래스에서 오
  //버라이딩을 해야 한다.
  System.out.println("디폴트 메서드 오버라이딩");
 }
}
```

두 부모 인터페이스에 동일한 이름의 디폴트 메서드가 정의된 경우 관한 소스예 8)

```
public class InterEx08 {
public static void main(String[] args) {
  ChildClass08 child = new ChildClass08();
   child.display();
  }
}
```

부모 인터페이스의 default 메서드와 부모 조상클래스의 일반 메서드가 동일한 경우 관한 소스예 9)

```
 class ParentClass09 {
    public void display() {
      System.out.println("부모클래스 일반 메서드");
    }
 }

interface ParentInterface09 {
   default void display() {//public 접근제어자가 생략된 디폴트 메서드
       System.out.println("부모 인터페이스의 디폴트 메서드");
   }
}

class ChildClass09 extends ParentClass09 implements ParentInterface09 {
    /* 부모 인터페이스의 default 메서드와 부모 조상클래스의 일반 메서드명이 동일한 경우
     * 부모 클래스 일반 메서드가 상속되고 부모 인터페이스의 default 메서드는 무시된다.
     */
}

public class InterEx09 {
   public static void main(String[] args) {
     ChildClass09 child = new ChildClass09();
     child.display(); // ParentClass09 부모클래스의 일반메서드가 호출됨
  }
 }
```

봉인된 인터페이스

1. java 15부터는 무분별한 자손 인터페이스 생성을 방지하기 위해 봉인된(sealed) 인터페이스가 도입되었다.

```
//봉인된 인터페이스 정의
 public sealed interface InterfaceA permits InterfaceB {

}
```

다음과 같이 sealed로 InterfaceA를 봉인된 인터페이스로 정의하면 permits 다음에 오는 InterfaceB만 자손인터페이스로 정의할 수 있다. 그 이외는 자손 인터페이스로 만들 수 없다. sealed 키워드로 봉인된 인터페이스를 정의하면 permits 키워드 뒤에 상속 가능한 자손 인터페이스를 지정해야 한다.

2. 봉인된 InterfaceA 인터페이스를 상속하는 InterfaceB는 non-sealed 키워드로 비봉인 자손 인터페이스로 선언하거나, sealed 키워드를 사용해서 또 다른 봉인 인터페이스로 선언해 야 한다.

```
public non-sealed interface InterfaceB extends InterfaceA{ ....}
```

non-sealed 는 봉인을 해제하겠다는 뜻이다. 따라서 interfaceB는 다른 자손 인터페이스를 만들 수 있다.

```
 public interface interfaceC extends InterfaceB{ ... }
```

봉인된 인터페이스 소스예 10)

```
//sealed 키워드로 봉인된 인터페이스 InterfaceA 정의 => InterfaceB만 자손 인터페이스로 정의가능
public sealed interface InterfaceA permits InterfaceB{

  void methodA();//추상메서드 정의, public abstrac이 생략됨.

}

//non-sealed 키워드로 비봉인 인터페이스 정의, 다른 인터페이스 자손을 만들수 있다.
public non-sealed interface InterfaceB extends InterfaceA {

  void method();

}

// InterfaceB를 상속하는 또 다른 자손 인터페이스 정의
public interface InterfaceC extends InterfaceB {

  void methodC();
}
```

같은 패키지안에 InterfaceA.java, InterfaceB.java, InterfaceC.java를 생성한다.

봉인된 인터페이스 소스예 10)

```
public class ImplClass implements InterfaceC {//implements 키워드로 부모인터페이스 구현상속한다.

  @Override
  public void methodB() {
    System.out.println("methodB() 실행");
  }

  @Override
  public void methodA() {
   System.out.println("methodA() 실행");
  }

  @Override
  public void methodC() {
   System.out.println("methodC() 실행");
  }//부모 인터페이스의 모든 추상메서드를 반드시 오버라이딩을 해야 한다.
 }
```

동일 패키지에 InterfaceC.java를 구현상속한 ImplClass.java를 생성한다.

봉인된 인터페이스 소스예 10)

```
public class SealedExample2 {
public static void main(String[] args) {

 ImplClass impl = new ImplClass();
 InterfaceA ia = impl;//업캐스팅
 ia.methodA();//업캐스팅 이후 오버라이딩 된 메서드 호출

 System.out.println();

 InterfaceB ib=impl;
 ib.methodA();
 ib.methodB();
 System.out.println();

 InterfaceC ic=impl;
 ic.methodA();
 ic.methodB();
 ic.methodC();
}
}
```

추상 클래스와 인터페이스의 차이점

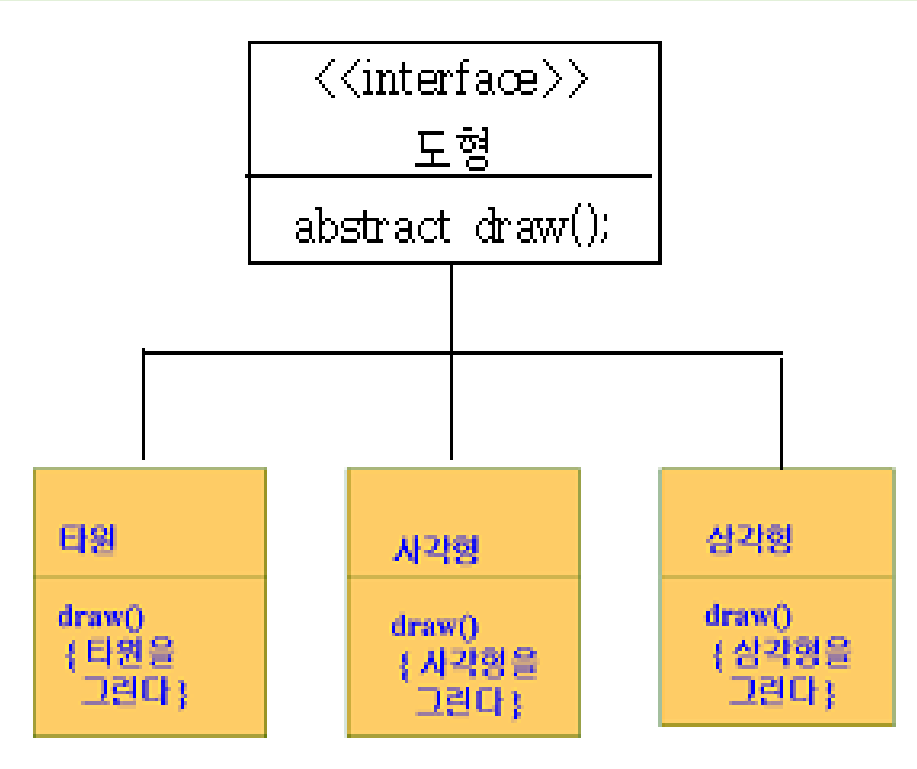
추상 클래스	인터페이스
클래스이다. (abstract class로 선언)	클래스가 아니다.(interface로 선언)
일반 메서드와 추상 메서드를 모두 가질 수 있다.	public abstract 추상 메서드만 가질 수 있다 (jdk1.7). Jdk1.8 이후 부터는 static 메서드와 default 메서드를 가질 수 있다.
확장을 하며 extends로 서브 클래스로 정의	구현을 하며 implements로 서브 클래스 정의
단일 상속만 가능하다.	다중 상속도 가능하다.
변수와 상수를 모두 가질 수 있다.	public static final 형태의 정적상수만 가질 수 있다.
모든 추상 메서드는 객체 생성을 위한 서브 클래스에서 반드시 구현되어야 한다. 업캐스팅이 가능하다.	

<문제>

1. 다음과 같은 인터페이스와 클래스를 설계하시오.

다음은 설계된 클래스로 프로그래밍한 것입니다.

```
public class Ex19_01 {
    public static void main(String[] args) {
     IShapeClass   ref;
     ref = new  Circ();
     ref.draw();

     ref = new  Rect();
     ref.draw();

     ref = new  Tria();
     ref.draw();
  }
}
```



<문제>

2. 다음 예제는 컴파일 에러가 발생합니다. 성공적으로 컴파일하기 위해서 어떤 부분을 수정해야 하는지 적합한 설명을 하시오.

```
public abstract class Test {
  public abstract void methodA();

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