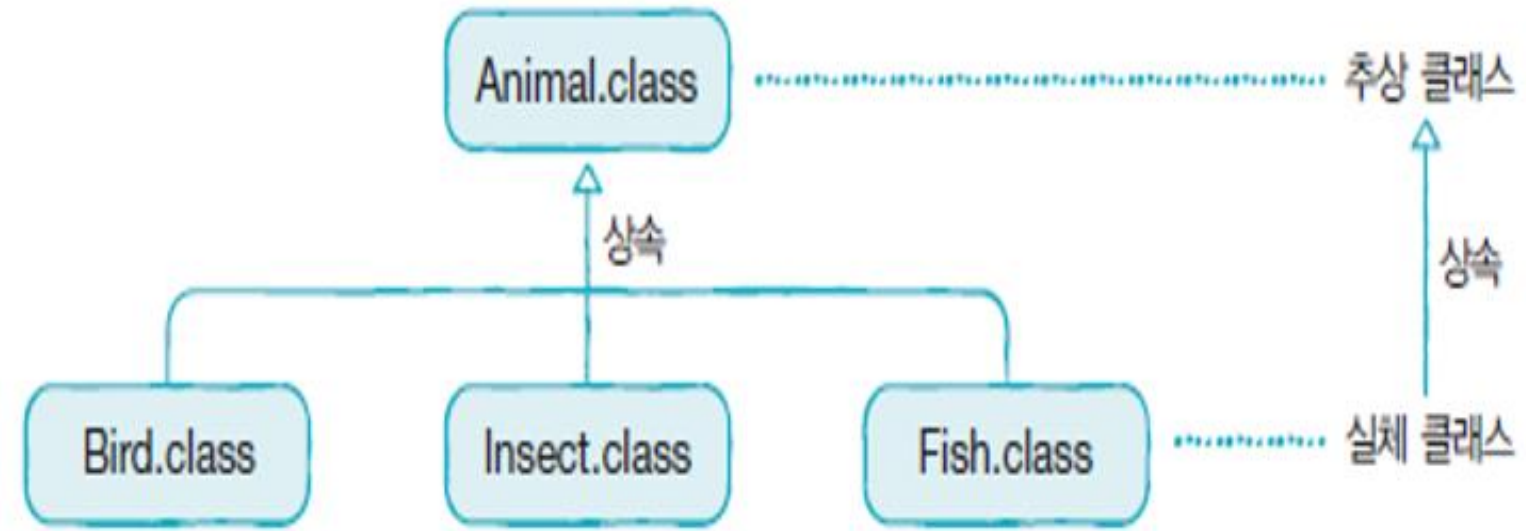
추상클래스와 인터페이스

안 화 수

추상 클래스

❖ 추상 클래스

- 실체 클래스들의 공통적인 특성(필드, 메소드)을 추출하여 선언한 클래스
- 추상 클래스와 실체 클래스는 부모, 자식 클래스로서 상속 관계를 가진다.



추상 클래스

추상 클래스

1. 추상 클래스는 자체적으로 객체를 생성할 수 없다.

```
public abstract class AbsClass{
}
```

2. 추상 클래스를 구성하는 요소는 필드, 생성자, 메소드, 추상 메소드가 있다.

```
public abstract class AbsClass{
    int a=10;                       // 필드
    abstract void Method01();       // 추상메소드
    void Method02(){                // 일반 메소드
    }
}
```

3. 추상 클래스를 상속 받을때는 extends를 이용해서 상속을 받는다.
4. 추상클래스를 상속받은 일반 클래스는 추상클래스 안에 들어있는 추상 메소드를 반드시 Method Overriding 해야 한다.
5. 추상 클래스도 단일 상속만 가능하다.(클래스의 다중상속을 허용하지 않는다.)

추상 클래스

- 추상 클래스 예1

추상 클래스도 extends로 상속 받아야 한다.

```
abstract class AbstractClass{
    abstract void Method01();
    void Method02(){
       System.out.println("Method02:   추상클래스에서 구현");
     }
}
class SubClass extends AbstractClass{
    void Method01(){
        System.out.println("Method01: 서브클래스에서 구현된 추상메소드");
    }
}
public class AbstractTest01{
    public static void main(String args[]){
       SubClass obj =  new SubClass();
       obj.Method01();
       obj.Method02();
     }
}
```

추상 클래스

- 추상 클래스 예2 : 추상 클래스는 다중상속이 허용되지 않는다.

```
abstract class Hello{
   public abstract void sayHello(String name);
}
abstract class GoodBye{
   public abstract void sayGoodBye(String name);
}
class SubClass extends GoodBye, Hello {   //클래스의 다중상속 허용안됨
   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 AbstractTest02{
    public static void main(String[] args) {
      SubClass test= new SubClass();
      test.sayHello("홍길동");
      test.sayGoodBye("홍길동");
    }
}
```

추상 클래스

- 추상 클래스 예3 : 추상 클래스가 추상 클래스를 상속 받을때도 extends로 상속 받아야 한다.

```
abstract class AbstractClass{
    abstract void Method01();
    void Method02(){
       System.out.println("Method02:   추상클래스에서 구현");
     }
}
abstract class MiddleClass extends AbstractClass{
    public void Method03(){
      System.out.println("Method03: 추상클래스에서 구현");
    }
}
class SubClass extends MiddleClass{
    void Method01(){     // 메소드 오버라이딩
      System.out.println("Method01: 서브클래스에서 구현된 추상메소드");
    }
}
public class AbstractTest03{
    public static void main(String args[]){
       SubClass obj =  new SubClass();
       obj.Method01();
       obj.Method02();
       obj.Method03();
     }
}
```

추상 클래스

- 추상 클래스 예4 : 메소드 오버라이딩(다형성)

```
abstract class ShapeClass{
    abstract void draw();
}
class Circ extends ShapeClass{
    void draw(){          // 메소드 오버라이딩
        System.out.println("원을 그린다");
    }
}
class Rect extends ShapeClass{
    void draw(){          // 메소드 오버라이딩
        System.out.println("사각형을 그린다");
    }
}
class Tria extends ShapeClass{
    void draw(){          // 메소드 오버라이딩
        System.out.println("삼각형을 그린다");
    }
}
public class AbstractTest04{
    public static void main(String args[]){
        Circ c =  new Circ();
        Rect r = new Rect();
        Tria t=  new Tria();
        c.draw();
        r.draw();
        t.draw();
    }
}
```

인터페이스

인터페이스(interface)

1. 인터페이스는 상수와 추상 메소드로 구성되어 있다.
   자바8부터 디폴터 메소드, 정적 메소드도 사용 가능함

```
public interface Inter01{
    int a = 10;                 //상수 (public static final생략가능)
    void check();               //추상메소드 (public abstract 생략가능)
}
```

2. 인터페이스를 상속 받을때는 implements로 상속을 받는다.
3. 인터페이스를 상속받은 일반 클래스는 인터페이스 안에 들어있는 추상 메소드를 반드시 Method Overriding 해야 된다.

```
interface A{
    public abstract void check();
}
class S  implements  A{
      public void check(){      //public을 생략할 수 없다.
      }
}
```

인터페이스

인터페이스(interface)

4. 인터페이스는 다중상속을 허용한다.

```
interface A{}
interface B{}
class S implements A, B{  }
```

5. 추상클래스와 인터페이스를 모두 상속을 받는 경우에는 추상클래스를 먼저 상속을 받고, 인터페이스는 그 다음으로 상속 받아야 한다. (상속 받는 순서가 바뀌면 안됨)

```
interface A{}
abstract class B{}
class S extends B implements A{}
```

6. 인터페이스끼리 상속을 받을 때는 extends로 상속 받는다.

```
interface A{}
interface B{}
interface C   extends   A, B { }
```

인터페이스

인터페이스(interface) 예1

인터페이스는 implements로 상속 받는다.

```
interface IHello {
    void sayHello(String name);
}
class Hello implements IHello {
    public void sayHello(String name) {
//  void sayHello(String name){
        System.out.println(name + "씨 안녕하세요!");
    }
}
class InterfaceTest01 {
    public static void main(String[] args) {
        Hello obj = new Hello();
        obj.sayHello("홍길동");
        obj.sayHello("홍길동");
    }
}
```

인터페이스

- 인터페이스(interface) 예2

인터페이스는 다중 상속을 허용한다.

```
interface IHello {
    public abstract void sayHello(String name);
}
interface IGoodBye {
    public abstract void sayGoodBye(String name);
}
class SubClass implements IHello, IGoodBye { //인터페이스의 다중상속
    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 InterfaceTest02 {
    public static void main(String[] args) {
        SubClass test = new SubClass();
        test.sayHello("홍길동");
        test.sayGoodBye("홍길동");
    }
}
```

인터페이스

인터페이스(interface) 예3

클래스를 먼저 상속받고, 인터페이스는 나중에 상속을 받아야 한다.

```
interface IHello {
    public abstract void sayHello(String name);
}
abstract class GoodBye {
    public abstract void sayGoodBye(String name);
}
class SubClass extends GoodBye implements IHello {
    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 InterfaceTest03 {
    public static void main(String[] args) {
        SubClass test = new SubClass();
        test.sayHello("홍길동");
        test.sayGoodBye("홍길동");
    }
}
```

인터페이스

인터페이스(interface) 예4

인터페이스는 다중상속을 허용한다.

```
interface IHello {
    public abstract void sayHello(String name);
}
interface IGoodBye {
    public abstract void sayGoodBye(String name);
}
// 인터페이스의 다중상속
class SubClass implements IHello, IGoodBye {
    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 InterfaceTest04 {
    public static void main(String[] args) {
        SubClass test = new SubClass();
        test.sayHello("홍길동");
        test.sayGoodBye("홍길동");
    }
}
```

인터페이스

인터페이스(interface) 예5 (1/2)

인터페이스가 인터페이스를 상속 받을 때는 extends로 상속을 받는다.

```
interface IHello {
    int a = 10;
    public abstract void sayHello(String name);
}
interface IGoodBye {
    public abstract void sayGoodBye(String name);
}
interface ITotal extends IHello, IGoodBye {
    public abstract void greeting(String name);
}
class SubClass implements ITotal {
    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!");
    }
}
```

인터페이스

- 인터페이스(interface) 예5 (2/2)

```
public class InterfaceTest05 {
    public static void main(String[] args) {
        SubClass test = new SubClass();
        test.sayHello("홍길동");
        test.sayGoodBye("홍길동");
        test.greeting("홍길동");
    }
}
```

인터페이스

- 인터페이스(interface) 예6

```
interface IColor {
    int RED = 1;              // 상수(public static final 생략)
    public static final int GREEN = 2; // 상수
    public static final int BLUE = 3;  // 상수
    void setColor(int c);  // 추상메서드 (public abstract 생략)
    public abstract int getColor();     // 추상메서드
}
abstract class AbsColor implements IColor {
    int color = GREEN;
    public void setColor(int c) {
        color = c;
    }
}
class SubClass extends AbsColor {
    public int getColor() {
        return color;
    }
}
public class InterfaceTest06 {
    public static void main(String[] args) {
        SubClass test = new SubClass();
        test.setColor(IColor.RED);
        System.out.println(test.getColor());
    }
}
```