

4부 상속

- 18장 추상 클래스와 final

최문환



18장 추상 클래스와 final

1. 추상 클래스
2. 추상 메서드
3. 추상 클래스로 다형성 제공
4. final

1. 추상 클래스

```
abstract class 클래스이름 {  
  
}
```

추상클래스는 자생력이
없으므로 객체 생성을 하지
못함

<예제> 객체 생성이 불가능한 추상 클래스

```
001: abstract class AbstractClass{ //추상 클래스  
002:}  
003: class AbstractTest00{  
004:     public static void main(String[] args){  
005:         AbstractClass obj= new AbstractClass();  
006:     }  
007: }
```

2. 추상 메서드

```
abstract class 클래스이름 {  
    //일반 속성과 메서드 기술  
    abstract void 추상메서드이름( ); // 추상 메서드 선언  
}
```

```
001:class AbstractClass{//추상 클래스가 아닌 클래스에서  
002:    abstract void Method01( );//추상 메서드를 가질 경우 컴파일 에러  
003:}
```

<예제> 추상 클래스의 상속을 받는 서브 클래스 설계

```
001:abstract class AbstractClass{//추상 클래스
002: abstract void Method01( );//추상 메서드 : 실질적인 구현은 없다.
003: void Method02( ){//추상 클래스에서구현한메서드는 서브클래스에서 상속받아사용됨
004:   System.out.println("Method02 : 추상 클래스에서 구현");
005: }
006:}
007:class SubClass extends AbstractClass{//추상 클래스를 상속받은 서브 클래스에서
008:   void Method01( ){   //추상메서드를 반드시 구현해야 한다. (오버라이딩해야함)
009:     System.out.println("Method01 : 서브 클래스에서 구현된 추상 메서드");
010:   }
011:}
012:class AbstractTest01{
013: public static void main(String[] args){
014:   SubClass obj=new SubClass( ); //추상클래스의 상속을 받은 서브클래스로 객체생성
015:   obj.Method01( ); //추상메서드를 서브클래스에서 오버라이딩되어 사용가능하게 됨
016:   obj.Method02( ); //추상 클래스에서 구현한 메서드를 상속받아 사용함
017: }
018:}
```

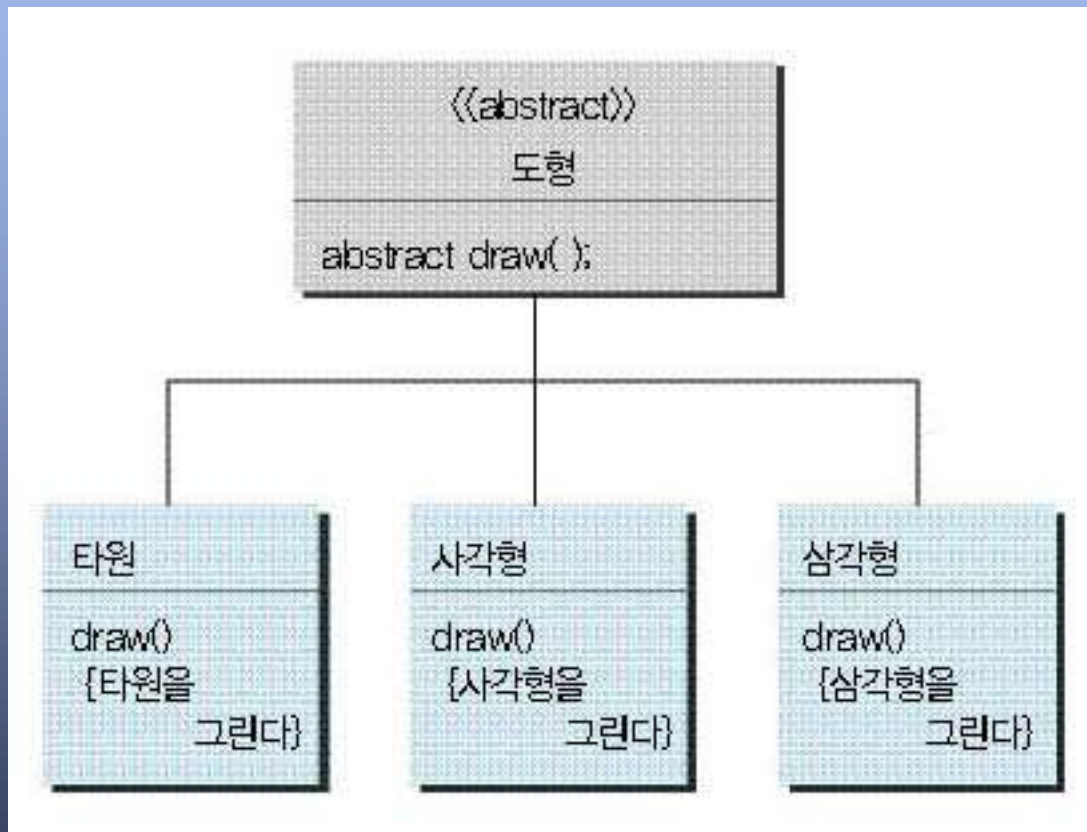
<예제> 추상메서드를 오버라이딩해야만 객체 생성가능

```
001:abstract class AbstractClass{ //추상 클래스
002: abstract void Method01( );//추상 메서드 : 실질적인 구현은 없다.
003: void Method02( ){           //추상 클래스에서 구현된 메서드는 서브 클래스로 상속됨
004:   System.out.println("Method02 : 추상 클래스에서 구현");
005: }
006:}
007:abstract class MiddleClass extends AbstractClass{//추상 클래스를 상속받은 서브 클래스가
008: //추상 메서드를 오버라이딩하지 않으려면 이 서브 클래스도 추상 클래스가 되어야 한다.
009: void Method03( ){           //추상 클래스에서 구현된 메서드는 서브 클래스로 상속됨
010:   System.out.println("Method03 : 추상 클래스에서 구현");
011: }
012:}
013:class SubClass extends MiddleClass{ //하지만 객체 생성하기 위해서는 서브 클래스에서 결국
014: void Method01( ){           //추상메서드를 반드시 구현해야 한다. (오버라이딩해야함)
015:   System.out.println("Method01 : 서브 클래스에서 구현된 추상 메서드");
016: }
017:}
```

<예제> 추상메서드를 오버라이딩해야만 객체 생성가능

```
018: class AbstractTestB{
019:     public static void main(String[] args){
020:         SubClass obj=new SubClass( );
021:         obj.Method01( );    //서브 클래스에서 추상 메서드를 오버라이딩하였기에 호출 가능
022:         obj.Method02( );    //추상 클래스에서 구현된 메서드를 상속 받아 호출
023:         obj.Method03( );    //추상 클래스에서 구현된 메서드를 상속 받아 호출
024:     }
025: }
```

3. 추상 클래스로 다형성 제공



3. 추상 클래스로 다형성 제공

- 서브 클래스에서 체계적인 클래스를 설계하도록 하기 위해서 자바에서는 추상 클래스를 제공합니다.
- 추상 클래스에 동일한 인터페이스를 요구하는 추상 메서드를 정의하면 이 추상클래스를 슈퍼 클래스를 가지는 서브 클래스에서 반드시 동일한 이름의 메서드(추상 메서드)를 정의하도록 강제성을 부여합니다.
- 그리고, 구체적인 기능들을 서브 클래스에서 구현하도록 함으로서 다형성을 제공해 줍니다.

<예제> 추상 클래스를 이용한 클래스 설계

```
001:abstract class ShapeClass{//추상 클래스
002: abstract void draw( ); //추상 메서드 : 실질적인 구현은 없다.
003:}
004:class Circ extends ShapeClass{ //추상 클래스를 상속받은 서브 클래스에서
005: void draw( ){ //추상메서드를 반드시 구현해야 한다.
006: System.out.println("원을 그린다");
007: }
008:}
009:
010:class Rect extends ShapeClass{ //추상 클래스를 상속받은 서브 클래스에서
011: void draw( ){ //추상메서드를 반드시 구현해야 한다.
012: System.out.println("사각형을 그린다");
013: }
014:}
015:
```

<예제> 추상 클래스를 이용한 클래스 설계

```
016: class Tria extends ShapeClass{ //추상 클래스를 상속받은 서브 클래스에서
017: void draw( ){                  //추상메서드를 반드시 구현해야 한다.
018:   System.out.println("삼각형을 그린다");
019: }
020:}
021:
022: public class AbstractTest02 {
023:   public static void main(String[] args) {
024:     Circ c = new Circ( );
025:     Rect r = new Rect( );
026:     Tria t = new Tria( );
027:
028:     c.draw( ); //메서드는 draw로 동일하지만 출력되는 내용이 다르다.
029:     r.draw( );
030:     t.draw( );
031:   }
032:}
```

3.1 추상 클래스와 업 캐스팅

```
001:public class AbstractTest03 {  
002: public static void main(String[] args) {  
003:     ShapeClass ref; //추상 클래스로 레퍼런스 변수 선언함  
004:     //동일한 레퍼런스 변수로 동일한 함수를 호출하지만 실행결과는 다름  
005:     ref = new Circ( ); //업 캐스팅  
006:     ref.draw( );       //Circ 클래스의 draw 메서드 호출  
007:  
008:     ref = new Rect( ); //업 캐스팅  
009:     ref.draw( );       //Rect 클래스의 draw 메서드 호출  
010:  
011:     ref = new Tria( ); //업 캐스팅  
012:     ref.draw( );       //Tria클래스의 draw 메서드 호출  
013:
```

3.1 추상 클래스와 업 캐스팅

```
014: System.out.println("-----");
015: //추상 클래스로 배열을 선언하여 3개의 서로 다른 객체를 가리키도록 한다.
016: ShapeClass[] arr= new ShapeClass[3];
017: arr[0] = new Circ( ); //업 캐스팅
018: arr[1] = new Rect( ); //업 캐스팅
019: arr[2] = new Tria( ); //업 캐스팅
020:
021: for(int i=0; i<3; i++)
022:     arr[i].draw( );
023: }
024: }
```

<예제> 추상 클래스는 다형성 제공

```
001:public class AbstractTest04 {  
002:  static void polymorphism(ShapeClass ref){  
003:    ref.draw( );  
004:  }  
005:  public static void main(String[] args) {  
006:    Circ c = new Circ( );  
007:    Rect r = new Rect( );  
008:    Tria t = new Tria( );  
009:    polymorphism(c); //polymorphism(c);  
010:    polymorphism(r);  
011:    polymorphism(t);  
012:  }  
013:}
```

추상 클래스 정리

- abstract 클래스는 추상 클래스이다.
- 추상 메서드와 일반 메서드를 가질 수 있다.
- 상속을 위해서 extends를 사용한다.
- 모든 추상 메서드는 구현하여야 사용할 수 있다.
- 업캐스팅이 가능하다.

4. final

4.1 final 속성 ▪

```
001: class FinalMember {  
002:     final int a=10;  
003:     public void setA(int a){  
004:         this.a=a;  
005:     }  
006: }  
007: public class FinalTest01{  
008:     public static void main(String[] args) {  
009:         FinalMember ft= new FinalMember( );  
010:         ft.setA(100);  
011:         System.out.println(ft.a);  
012:     }  
013: }
```


4.2 final 메서드

<예제> 오버라이딩 불가능한 final 메서드

```
001: class FinalMethod{
002:   String str="Java ";
003:   //public void setStr(String s) {
004:   //final 붙이면 서브 클래스에서 오버라이딩이 불가.
005:   public final void setStr(String s) {
006:     str=s;
007:     System.out.println(str);
008:   }
009: }
010: class FinalEx extends FinalMethod{
011:   int a=10; // final 붙이면 밑에서 a값 대입 불가.
012:   public void setA(int a) {
013:     this.a=a;
014:   }
```

4.3 final 클래스

<예제> 더 이상의 상속을 허락하지 않는 클래스 설계

```
001:final class FinalClass{
002: String str="Java ";
003: public void setStr(String s){
004:     str=s;
005:     System.out.println(str);
006: }
007:}
008:class FinalEx extends FinalClass{
009: int a=10;
010: public void setA(int a) {
011:     this.a=a;
012: }
013: public void setStr(String s){
014:     str+=s;
015:     System.out.println(str);
016: }
017:}
018:public class FinalTest03{
019: public static void main(String[] args) {
020:     FinalEx fe= new FinalEx( );
021: }
022:}
```

<문제>

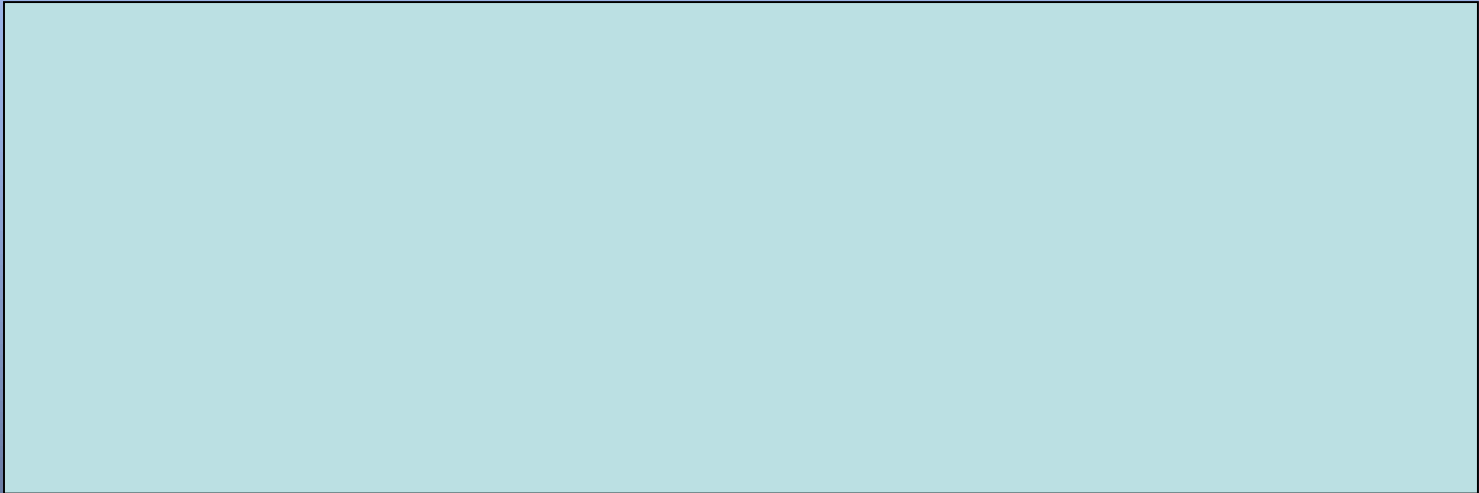
1. 다음 예제에서 에러가 발생하지 않도록 박스를 채우시오.

```
abstract class Abs1{  
    int a=10;  
    String str="Test";  
    public abstract int getA( );  
    public String getStr( ){  
        return str;  
    }  
}
```

```
abstract class Abs2 extends Abs1{  
    int b=100;  
    public abstract int getB( );  
}
```

<문제>

```
class AbsMain extends Abs2{
```



```
}  
public class Ex18_01{  
    public static void main(String[] args) {  
        AbsMain am=new AbsMain( );  
        System.out.println(am.getA( ));  
        System.out.println(am.getB( ));  
    }  
}
```

<문제>

2. 다음 프로그램의 수행 결과를 유추하시오.

```
001: class A {  
002:   public final int method1() {return 0; }  
003: }  
004: class B extends A {  
005:   public int method1() { return 1; }  
006: }  
007: public class Ex18_02{  
008:   public static void main(Strings args[]) {  
009:     B b=new B( );  
010:     System.out.println("x = " + b.method1());  
011:   }  
012: }
```