

chapter 05.

클래스 II

○ 클래스의 관계

[표 5-1] 클래스 관계의 종류

구분	클래스 관계
[has a]	특정 객체 내에서 다른 객체를 가지고 있는 것을 의미한다.
[is a]	특정 객체가 다른 객체에게 자신의 능력을 포함시켜주는 상속 관계를 의미한다.

○ 상속의 개념과 중요성

■ 클래스의 상속

상속이란? 부모가 보유하고 있는 재산 중 일부를 자식이 물려받는 것을 의미한다. 자바에서는 이런 클래스들간의 다중 상속을 지원하지 않으므로 객체의 명확성을 높였다.

[표 5-2] 상속관계의 용어

용어	설명
Base Class(기본 클래스) Super Class(슈퍼 클래스) Parent Class(부모 클래스)	왼쪽의 용어가 모두 같은 뜻을 의미하며 이는 상속을 주기 위해 준비된 특정 클래스를 의미한다.
Derivation Class(유도 클래스) Sub Class(하위 클래스) Child Class(자식 클래스)	왼쪽의 용어가 모두 같은 뜻을 의미하며 이것은 특정 클래스로부터 상속을 받아 새롭게 정의되는 클래스를 의미한다.

□ 1) 클래스의 상속2

○ 클래스 상속의 정의 법

- 자바에서 얘기하는 상속이라는 것은 특정 클래스가 가지는 일부 속성과 기능을 다른 새로운 클래스에게 제공하기 위해 맺는 클래스들간의 관계를 말한다.
이는 super클래스를 새로운 sub클래스에서 [extends]라는 예약어를 사용하여 서로 관계를 맺은 상태이다.

```
class [sub클래스명] extends [super클래스명] {  
    ...;  
}
```

□ 1) 클래스의 상속3

○ 클래스 상속의 중요성

클래스 상속은 객체의 재사용이라는 장점뿐만 아니라 코드의 간결성을 제공해 주는 객체지향적 언어의 장점과 중요한 특징이 된다. 그러므로 잘 정의된 super클래스가 있다면 sub클래스의 작성이 간편해지고 무엇보다 개발 시간이 단축된다는데 상속의 중요성과 장점을 들 수 있다.

[예제5-1] CellPhone.java

```
01 class CellPhone{
02
03     String model; // 모델명
04     String number; // 전화번호
05     int chord; // 화음
06
07     public void setNumber(String n){
08         number = n;
09     }
10     public String getModel(){
11         return model;
12     }
13     public int getChord(){
14         return chord;
15     }
```

```
16         public String getNumber(){
17             return number;
18         }
19     }
```

□ 1) 클래스의 상속4

[예제5-2] D_caPhone.java

```
01 class D_caPhone extends CellPhone{
02
03     String pixel; //화소 수
04     public D_caPhone (String model,String num, int chord, String pixel){
05         this.model = model;
06         number = num;
07         this.chord = chord;
08         this.pixel = pixel;
09     }
10 }
```

[예제5-3] MP3Phone.java


```
01 class MP3Phone extends CellPhone{
02
03     int size; // 저장 용량
04     public MP3Phone (String model,String num, int chord, int size){
05         this.model = model;
06         number = num;
07         this.chord = chord;
08         this.size = size;
09     }
10 }
```

□ 1) 클래스의 상속5

[예제5-4] CellPhoneTest.java

```
01 class CellPhoneTest{
02
03     public static void main(String[] args){
04         D_caPhone dca = new D_caPhone(
05             "IN-7600","011-9XXX-9XXXX",60,"400만");
06         MP3Phone mp = new MP3Phone(
07             "KN-600","011-9XXX-9XXXX",60,256);
08
09         System.out.println(dca.getModel()+" "+
10             dca.getChord()+" "+dca.getNumber());
11     }
12 }
```

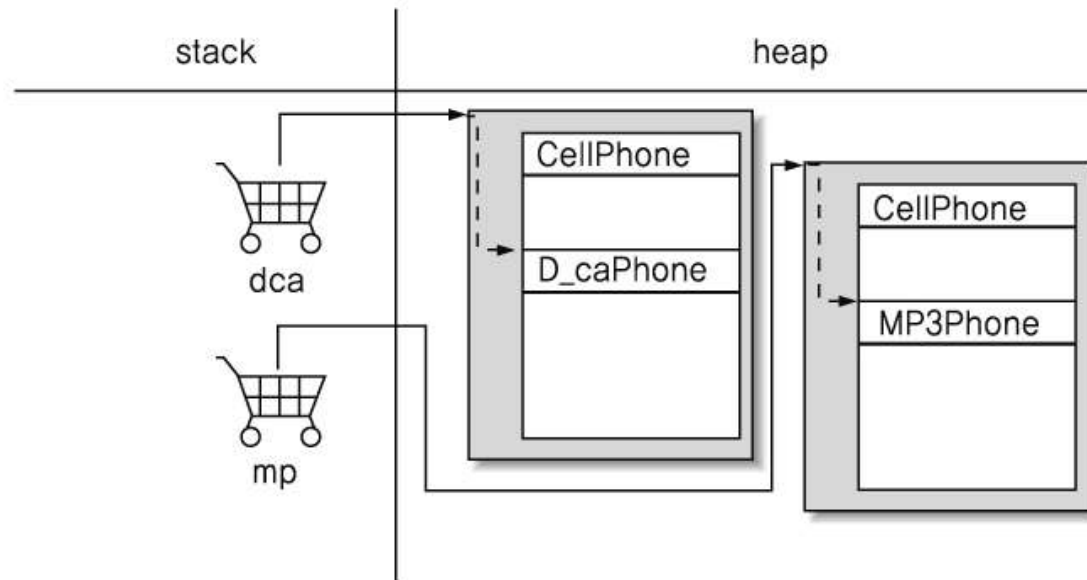
실행결과



```
----- Java Run -----
IN-7600, 60, 011-9XXX-9XXXX
Normal Termination
출력 완료 (1초 경과).
```

□ 1) 클래스의 상속6

- 앞의 예제로 상속관계에 있어 정보의 공유와 상속을 받는 sub클래스의 작업이 매우 유익함을 알 수 있다. 다음 그림을 살펴보고 상속관계의 객체 생성시 구조를 이해해보도록 하자!



[그림 5-3] 상속관계에서 객체를 생성할 때의 메모리 구조

○오버라이딩

- 오버라이딩은 [메서드 재정의]라고도 불리며 이는 서로 상속관계로 이루어진 객체들간의 관계에서 비롯된다.

super클래스가 가지는 메서드를 sub클래스에서 똑 같은 것을 새롭게 만들게 되면 더 이상 super클래스의 이름이 같은 메서드를 호출할 수 없게 된다. 이를 Overriding이라 하고 또는 멤버 은폐라고도 한다.

[표 5-3] 오버라이딩과 오버로딩의 차이

오버라이딩(재정의)	구분	오버로딩(다중정의)
상속관계	적용	특정 클래스
super 클래스의 메서드보다 sub 클래스의 메서드 접근제한이 동일하거나 더 넓어야 한다. 예를 들어, protected라면 protected/public이다.	접근제한	상관없다.
기본적으로 같아야 한다.	리턴형	상관없다.
super 클래스의 메서드명과 sub 클래스의 메서드명이 같아야 한다.	메서드명	반드시 같아야 한다.
반드시 같아야 한다.	인자(타입, 개수)	반드시 달라야 한다.

□ 1) 클래스의 상속8

■ 오버라이딩 예제

[예제5-5]OverridingEx.java

```
01 class Parent{
02
03     String msg = "Parent클래스";
04     public String getMsg(){
05         return msg;
06     }
07 }
08
09 class Child extends Parent{
10
11     String msg = "Child클래스";
12     public String getMsg(){ //메서드 Overriding
13         return msg;
14     }
15 }
16
```

```
17 public class OverridingEx {
18     public static void main(String[] args){
19
20         Child cd = new Child();
21         System.out.println("cd : "+cd.getMsg());
22
23         Parent pt = new Child();
24         System.out.println("pt : "+pt.getMsg());
25     }
26 }
```

실행결과

```
----- Java Run -----
cd : Child클래스
pt : Child클래스
Normal Termination
출력 완료 (0초 경과).
```

○super 와 super ()

- super는 앞서 배운 this와 함께 객체를 참조할 수 있는 reference변수이다. this는 특정 객체 내에서 자기 자신의 객체를 참조할 수 있는 유일한 reference변수이다. 그리고 super라는 reference변수는 현재 객체의 바로 상위인 super클래스(부모클래스)를 참조할 수 있는 것이다.
- super()는 바로 super클래스의 생성자를 의미하는 것이다. 인자가 있다면 인자의 형태와 일치하는 생성자를 의미한다.

□ 1) 클래스의 상속10

■ super() 예제

[예제5-6]SuperEx.java

```
01 class Parent{
02
03     public Parent(int var){
04         System.out.println("Parent 클래스");
05     }
06 }
07 class SuperEx extends Parent {
08     public SuperEx() {
09         super(1);
10         System.out.println("SuperEx 클래스");
11     }
12
13     public static void main(String[] args) {
14         SuperEx se = new SuperEx();
15     }
16 }
```

실행결과

```
----- Java Run -----
Parent 클래스
SuperEx 클래스
Normal Termination
출력 완료 (0초 경과).
```

□ 2) final 예약어 (260p)

- final은 예약어이며 이것은 더 이상의 확장이 불가능함을 알리는 종단(상수)과 같은 것을 의미한다.
 - 변수에 final을 적용 시 상수를 의미한다.

[표 5-4] 변수에 final을 적용할 때의 사용 예

구성	사용 예
final [자료형] [변수명];	final int VAR = 100;

- 메서드에 final을 적용 시 오버라이딩으로의 확장이 불가능하다.

[표 5-5] 메서드에 final을 적용할 때의 사용 예

구성	사용 예
[접근제한] final [반환형] [메서드명]() { }	public final void method() { }

- 클래스에 final을 적용 시 더 이상의 상속 확장이 불가능하다.

[표 5-6] 클래스에 final을 적용할 때의 사용 예

구성	사용 예
[접근제한] final class [클래스명] { }	public final class MeEx { }

○ 추상화의 이해와 선언법

- 추상화라는 것은 구체적인 개념으로부터 공통된 부분들만 추려내어 일반화 할 수 있도록 하는 것을 의미한다. 다시 말해서 일반적으로 사용할 수 있는 단계가 아닌 아직 **미완성(未完成)적 개념**인 것이다. 그럼 자바에서 얘기하는 추상(abstract)화 작업을 하기 위해서는 먼저 추상 메서드를 이해해야 한다.

[표 5-7] 추상 메서드의 구성과 예문

구성	사용 예
[접근제한] abstract void [메서드명]();	public abstract void absTest();

위의 내용은 추상 메서드의 구성이다. 구성과 사용 예를 살펴보면 메서드를 정의하면서 brace({})를 생략하여 실상 메서드가 하는 일(body)이 없이 semicolon(;)으로 문장의 끝을 나타내었다. 그리고 abstract라는 예약어를 통해 현 메서드가 추상 메서드임을 명시하였다.

□ 3) 추상화2

- 앞서 배운 추상 메서드를 하나라도 가지게 되는 클래스가 바로 추상 클래스가 된다. 그리고 이런 추상 클래스 또한 다음과 같이 추상 클래스임을 명시해야 한다.

[표 5-8] 추상 클래스의 구성과 예문

구성	사용 예
[접근제한] abstract class [클래스명]{ }	public abstract class AbsEx{ }

[예제5-10] AbsEx1.java

```
01 abstract class AbsEx1{
02     int a = 100; //변수
03     final String str = "abstract test"; //상수
04     public String getStr(){ //일반 메서드
05         return str;
06     }
07
08     // 추상 메서드는 몸체(body)가 없다.
09     abstract public int getA();
10 }
```

□ 3) 추상화3

○ 추상 클래스의 상속 관계

- 추상 클래스들간에도 상속이 가능하다. 일반 클래스들간의 상속과 유사하지만 추상 클래스들간의 상속에서는 상속 받은 추상 메서드들을 꼭 재정의할 필요는 없다. 그냥 상속만 받아두고 있다가 언젠가 일반 클래스와 상속관계가 이루어 질 때가 있을 것이다. 이때 재정의 하지 못했던 상속 받은 추상 메서드들을 모두 일반 클래스 내에서 재정의해도 되기 때문이다.

그럼 앞서 작성한 예제 AbsEx1을 상속 받는 추상 클래스를 작성하여 다른 예제를 작성해 보자!

[예제5-11] AbsEx2.java

```
01 abstract class AbsEx2 extends AbsEx1{
02     public int getA(){ // 부모클래스의 추상 메서드 재 정의
03         return a;
04     }
05     public abstract String getStr();
06 }
```

□ 3) 추상화4

[예제5-12] AbsEx.java

```
01 class AbsEx extends AbsEx2{
02
03     public String getStr(){ //AbsEx2의 추상 메서드 재 정의
04         return str; //str은 AbsEx1의 멤버이다
05     }
06     public static void main(String[] args){
07         AbsEx ae = new AbsEx();
08         System.out.println("ae.getA():"+ae.getA());
09         System.out.println("ae.getStr():"+ae.getStr());
10     }
11 }
```

실행결과

```
----- Java Run -----
ae.getA():100
ae.getStr():abstract test
Normal Termination
출력 완료 (0초 경과).
```


□ 4) 인터페이스1 (272p)

- 인터페이스는 음식점의 메뉴판과 같은 것이다.

메뉴판을 보고 고객이 원하는 음식을 요청하게 되는데 메뉴판 자체가 음식을 주지는 않는다. 실제 음식은 주방이라는 곳에서 나오므로 메뉴판은 고객이 호출할 수 있는 서비스의 목록이라 할 수 있다.

```
[접근제한] interface [인터페이스명] {  
    상수;  
    추상메서드;  
}
```

- 위의 인터페이스의 구조를 보고 알 수 있듯이 인터페이스 내에는 상수 또는 추상 메서드들만 정의가 가능하다. 그리고 사용하기 위해서는 일반 클래스에서 구현(implements)력을 가져야 한다. 다시 말해서 일반 클래스에서 “implements”라는 예약어로 특정 인터페이스를 구현하겠다고 명시하는 것이다. 그렇게 되면 명시한 인터페이스가 가지는 추상 메서드들은 구현 받은 클래스에서 하나도 빠짐없이 Overriding(재정의)해야 한다. 다음 예제는 인터페이스의 기본 구성과 구현을 다룬 예제이다.

□ 4) 인터페이스2

[예제5-18]InterTestEx.java

```
01 interface InterTest {
02     static final int A = 100;
03     abstract int getA(); //abstract예약어는 생략 가능!
04 }
05
06 class InterTestEx implements InterTest
07 {
08     public int getA(){
09         return A;
10     }
11
12     public static void main(String[] args)
13     {
14         InterTestEx it1 = new InterTestEx();
15         System.out.println("getA():"+it1.getA());
16     }
17 }
```

실행결과

```
----- Java Run -----
getA():100
Normal Termination
출력 완료 (0초 경과).
```

□ 4) 인터페이스3

○ 인터페이스간의 상속

- 위에서 공부한 것처럼 인터페이스 내에는 상수 또는 동작부분을 구현하지 않은 추상 메서드들이 정의된다. 그러므로 인터페이스를 구현(**implements**)하겠다고 명시한 일반 클래스에서 원하는 형태로 실제 구현력을 가지게 된다. 그러므로 실제 구현력이 없는 인터페이스들 간의 상속에서는 다중 상속이 제공 된다
인터페이스는 메뉴판과 같이 음식점에서 어떤 음식을 만들 수 있는지를 알려주는 중계자 역할만 할 뿐이다. 음식을 만들어 가져오는 것은 그 메뉴판을 포함(구현)하고 있는 음식점이 반드시 제공해야 할 의무가 있는 것이다.

```
[접근제한] interface [인터페이스명] extends 부모인터페이스명1,부모인터페이스명2,...,부모인터페이스명n {  
    상수;  
    추상메서드;  
}
```

- 인터페이스가 다른 인터페이스로부터 상속을 받았다고 하지만 Overriding을 할 수는 없다. 왜냐하면 앞서 공부했듯이 인터페이스는 body를 가지는 일반 메서드를 포함할 수 없다. 그러므로 상속을 받은 자식 인터페이스를 구현(**implements**)하는 일반 클래스에서 부모 인터페이스와 자식 인터페이스의 추상 메서드들을 모두 Overriding(재정의)해야 한다.

□ 4) 인터페이스4 (275p)

[예제5-19] InterEx2.java

```
01 interface Inter1{
02     public int getA();
03 }
04 //////////////////////////////////////
05 interface Inter2{
06     public int getA();
07 }
08 //////////////////////////////////////
09 interface Inter3 extends Inter1, Inter2{
10     public int getData();
11 }
12 //////////////////////////////////////
13 class InterEx2 implements Inter3{
14     int a = 100;
15     public int getA(){
16         return a;
17     }
18     public int getData(){
19         return a+10;
20     }
```

실행결과

```
----- Java Run -----
100
100
110
InterEx2@9cab16
Normal Termination
출력 완료 (0초 경과).
```

```
21     public static void main(String[] args){
22         InterEx2 it = new InterEx2();
23         Inter1 it1 = it;
24         Inter2 it2 = it;
25         Inter3 it3 = it;
26         System.out.println(it1.getA());
27         System.out.println(it2.getA());
28         System.out.println(it3.getData());
29     }
30 }
```

○ 내부 클래스의 이해와 특징

- 내부 클래스란? 특정 클래스 내에 또 다른 클래스가 정의되는 것을 의미한다.

이런 내부 클래스가 필요한 이유는 지금까지 작업해 왔던 클래스들과는 다르게 독립적이지는 않지만 하나의 멤버처럼 사용할 수 있는 특징이 있다.

- 내부 클래스를 정의 시 주의사항이자 장점
 - 내부 클래스는 외부 클래스의 모든 멤버들을 마치 자신의 멤버처럼 사용할 수 있다.
 - static 내부 클래스는 제외하고는 다른 내부 클래스는 항상 외부 클래스를 통해야 생성이 가능하다.

□ 6) 내부 클래스2

○ 내부 클래스의 종류와 사용 방법

[표 5-10] 내부 클래스의 종류

종류	설명
Member	멤버 변수나 멤버 메서드들과 같이 클래스가 정의된 경우에 사용한다.
Local	특정한 메서드 내에 클래스가 정의된 경우에 사용한다.
Static	static 변수(클래스 변수)와 같이 클래스가 static으로 선언된 경우에 사용한다.
Anonymous	참조할 수 있는 이름이 없는 경우에 사용한다.

□ 6) 내부 클래스3

- Member 내부 클래스

말 그대로 객체를 생성해야만 사용할 수 있는 멤버들과 같은 위치에 정의되는 클래스를 말한다. 즉 내부 클래스를 생성하려면 외부 클래스의 객체를 생성한 후에 생성할 수 있다.

- Member 내부 클래스의 구성

```
class Outer {  
    ...  
    class Inner {  
  
    }  
    ...  
}
```

□ 6) 내부 클래스4 (289p)

■ Member 내부 클래스 예제

[예제5-25] MemberInner.java

실행결과

```
01 class MemberInner{
02
03     int a = 10;
04     private int b = 100;
05     static int c = 200;
06
07     class Inner { //내부 클래스 정의
08         public void printData(){
09             System.out.println("int a : "+a);
10             System.out.println("private int b : "+b); // 주시하자!
11             System.out.println("static int c : "+c);
12         }
13     }
14     public static void main(String[] args){
15
16         // MemberInner outer = new MemberInner();
17         // MemberInner.Inner inner = outer.new Inner();
18         MemberInner.Inner inner = new MemberInner().new Inner();
19         inner.printData();
20     }
21 }
```

```
----- Java Run -----
int a : 10
private int b : 100
static int c : 200
Normal Termination
출력 완료 (0초 경과).
```


□ 6) 내부 클래스5

- Local 내부 클래스

Local 내부 클래스는 특정 메서드 안에서 정의되는 클래스를 말한다. 다시 말해서 특정 메서드 안에서 선언되는 지역변수와 같은 것이다. 메서드가 호출될 때 생성할 수 있으며 메서드의 수행력이 끝나면 지역변수와 같이 자동 소멸된다.

- Local 내부 클래스의 구성

```
class Outer {  
    ...  
    public void methodA() { // 멤버 메서드  
        class Inner {  
  
        }  
    }  
    ...  
}
```

□ 6) 내부 클래스6 (291p)

■ Local 내부 클래스 예제

[예제5-26] LocalInner.java

```
01 class LocalInner {
02
03     int a = 100; //멤버 변수
04     public void innerTest(int k){
05         int b = 200; // 지역변수
06         final int c = k; //상수
07         class Inner{
08             // Local 내부 클래스는 외부클래스의 멤버 변수와
09             // 상수들만 접근이 가능하다.
10             public void getData(){
11                 System.out.println("int a : "+a);
12                 // System.out.println("int b : "+b); //Local 내부클래스는
13                 // 지역변수를 사용할 수 없다.
14                 System.out.println("final int c : "+c); //상수 사용
15             }
16         } // 내부 클래스의 끝
17         Inner i = new Inner(); //메서드 내에서 Local 내부 클래스 생성
18         i.getData(); //생성된 reference를 통해 메서드 호출
19     } // 메서드의 끝
```

□ 6) 내부 클래스7

```
20     public static void main(String[] args) {  
21         LocalInner outer = new LocalInner();  
22         outer.innerTest(1000);  
23     }  
24 }
```

실행결과

```
----- Java Run -----  
int a : 100  
final int c : 1000  
Normal Termination  
출력 완료 (0초 경과).
```

□ 6) 내부 클래스8

- static 내부 클래스

static 내부 클래스로 어쩔 수 없이 정의하는 경우가 있는데 그것은 바로 내부 클래스 안에 static변수를 가지고 있다면 어쩔 수 없이 해당 내부 클래스는 static으로 선언하여야 한다.

- static 내부 클래스의 구성

```
class Outer {  
    ...  
    static class Inner {  
  
    }  
    ...  
}
```

■ static 내부 클래스 예제

[예제5-27] StaticInner.java

```
01 class StaticInner {
02
03     int a = 10;
04     private int b = 100;
05     static int c = 200;
06
07     static class Inner{
08         // 어쩔 수 없이 내부 클래스를 static으로 선언해야 할 경우가 있다.
09         // 그건 바로 내부 클래스의 멤버들 중 하나라도
10         // static멤버가 있을 때이다.
11         static int d = 1000;
12         public void printData(){
13             // System.out.println("int a : "+a); //오류
14             // System.out.println("private int b : "+b); //오류
15             System.out.println("static int c : "+c);
16         }
17     }
```

□ 6) 내부 클래스10

```
18     public static void main(String[] args) {  
19         //또 다른 독립된 객체에서 static 내부 클래스 생성시  
20         StaticInner.Inner inner = new StaticInner.Inner();  
21         inner.printData();  
22  
23         // StaticInner라는 외부 클래스내에서 생성시  
24         // Inner inner = new Inner();  
25     }  
26 }
```

실행결과

```
----- Java Run -----  
static int c : 200  
Normal Termination  
출력 완료 (0초 경과).
```

- static변수나 메서드들은 객체를 생성하지 않고도 접근이 가능하다고 했었다. 즉 static 내부 클래스는 외부 클래스를 생성하지 않고도 **[외부_클래스명.내부_클래스_생성자()]**로 생성이 가능함을 잊지 말자!

□ 6) 내부 클래스11

- Anonymous(익명) 내부 클래스

익명이란? 이름이 없는 것을 의미한다. 이것을 자바의 프로그램적으로 해석하면 정의된 클래스의 이름이 없다는 것이 된다.

- Anonymous(익명) 내부 클래스의 구성

```
class Outer {  
    ...  
    Inner inner = new Inner(){  
        ...;  
    };  
    public void methodA() { // 멤버 메서드  
        new Inner() {  
            ...;  
        };  
    }  
    ...  
}
```

□ 6) 내부 클래스12 (297p)

■ Anonymous(익명) 내부 클래스 예제

[예제5-29] AnonyInner1.java

```
01 abstract class TestAbst{
02     int data = 10000;
03     public abstract void printData(); // 추상메서드
04 }
05 class AnonyInner1{
06
07     TestAbst inn = new TestAbst (){
08         public void printData(){ // 미완성된 것을 완성한다.
09
10             System.out.println("data : "+data);
11         }
12     };
13
14     public static void main(String[] args){
15         AnonyInner1 ai = new AnonyInner1();
16         ai.inn.printData();
17     }
18 }
```

실행결과

```
----- Java Run -----
data : 10000
Normal Termination
출력 완료 (0초 경과).
```