제 7 장 확장 클래스

- 확장 클래스 – 다른 클래스로부터 상속 받기
- 메소드 재정의(overriding)
- 종단 메소드
- 추상 클래스와 추상 메소드

• 확장 클래스

• 기존의 클래스로부터 상속받아 작성된 클래스를 확장 또는 하위 클래스라 하며, 상속하는 기존의 클래스를 상위 클래스라 한다.

• 상속을 이용하여 상위 및 하위 클래스가 공통적으로 가지는 멤버변수와 메소드들은 상위 클래스에서 선언하고, 하위 클래스는 상위 클래스의 멤버변수 및 메소드들을 상속 받아 체계적으로 클래스를 설계할 수 있다.

• 확장 클래스의 작성

• 기존의 클래스를 상속 받아 확장 클래스를 작성하기 위한 형식은 다음과 같다.

```
[public|<abstract||final>] class 클래스이름 extends 클래스이름
{
    // 멤버변수 선언
    // 생성자
    // 메소드 정의
 }
```

설명 : 상속 받고자 하는 상위 클래스의 이름을 extends 지정어 다음에 한칸 이상 뛰어서 표현하며, 다른 사항은 일반 클래스의 작성형식과 같다.

예제 1

```
class SuperClass
{
   String name;
   String address;
   void methodA(){
      .....
   }
}
class SubClass extends SuperClass
{
   String nickname;
   void methodB(){
      .....
   }
}
```

SubClass는 확장 클래스로서 상위클래스인 SuperClass에서 정의한 멤버변수 및 메소드를 상속받는다.

• **예제 설명**

SubClass 클래스는 SuperClass 클래스로부터 상속된 확장 클래스이다. 따라서 SubClass는 자신의 클래스에서 정의한 멤버변수 및 메소드에 SuperClass에서 정의한 멤버변수 및 메소드를 추가한 클래스가 된다.

• 예제 1의 SubClass를 상속을 이용하지 않고 작성한 클래스는 예제 2와 같다.

SubClass(2)

```
public class SubClass
{
   String name;
   String address;
   String nickname;
   void methodA(){
      .....
   }
   void methodB(){
      .....
   }
}
```

상속을 받지않는 경우는 상위 클래스의 멤버변수 및 메소드를 다시 정의해야 한다.

- 멤버변수 및 메소드의 상속

예제 3

```
class Circle
{
   final double PI = 3.14; // 종단 변수(상수값)
   double radius;
   double area;
   public void calculatingArea(){
      area = PI * radius * radius;
      System.out.println("원기둥의 밑면적은 "+area+"이다.”);
   }
}
```

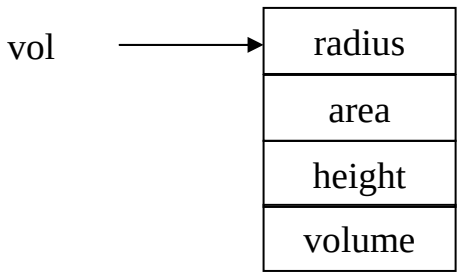
Cylinder 클래스는 Circle 클래스에서 정위한 3개의 객체변수 및 하나의 객체 메소드를 포함한다.

```
class Cylinder extends Circle
{
   double height;
   double volume;
   public void calculatingVolume(){
      volume=area * height;
      System.out.println("원기둥의 부피는 "+volume+"이다.”);
   }
}
```

ExecuteCylinder.java(3)

```
public class ExecuteCylinder
{
   public static void main(String args[]){
      Cylinder vol = new Cylinder();
      vol.radius = 10;
      vol.height = 5;
      vol.calculatingArea();
      vol.calculatingVolume();
   }
}
```

생성된 vol 객체는 그림과 같이 radius, area, height 및 volume 객체변수를 할당받는다.



• **예제 설명**

Circle 클래스를 상속받은 Cylinder 클래스는 Circle 클래스에서 정의한 calculatingArea() 메소드, radius 및 area 변수 그리고 자신의 클래스에서 선언한 calculatingVolume() 메소드, height 및 volume 변수를 포함한다. 따라서 vol 객체를 통하여 접근을 하였으며, 프로그램의 실행결과는 다음과 같다.

원기둥의 밑면적은 314.0이다.
원기둥의 부피는 1570.0이다.

*** 상위 클래스에서 private 접근 한정자로 선언된 모든 메소드 및 멤버변수는 확장 클래스로 상속이 불가능하다**

ExecuteName.java(4)

```
class Name
{
   String name = "홍길동";
}
class AnotherName extends Name
{
   String name = "임꺽정"; // 상위클래스의 멤버변수 이름과 같다.
}
class ExecuteName
{
   public static void main(String args[]){
      AnotherName sub = new AnotherName();
      System.out.println("확장 클래스의 객체변수 name의 값은 "+sub.name);
      Name sup = new Name();
      System.out.println("상위 클래스의 객체변수 name의 값은 "+sup.name);
   }
}
```

하위 클래스에서도 정의되었다.

하위 클래스의 name 값이 출력된다.

- **예제 설명**

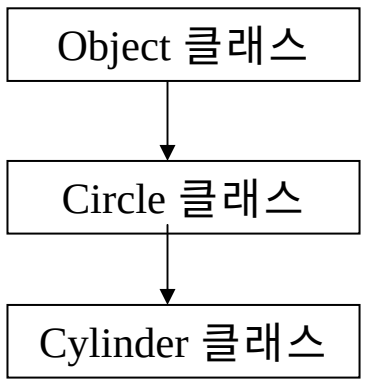
sub 객체는 2개의 name 문자열 객체를 포함하나, sub.name은 하위 클래스에서 정의한 객체의 값을 출력하므로 프로그램의 실행결과는 다음과 같다.

확장 클래스의 객체변수 name의 값은 홍길동
상위 클래스의 객체변수 name의 값은 임꺽정

• 클래스의 계층

• 자바에서 상위 클래스를 지정하지 않은 모든 클래스는 묵시적으로 Object 클래스를 상위 클래스로 가지며, Object 클래스는 자바가 지원하는 라이브러리 클래스이다.

• 예를 들면, 예제 3의 Cylinder 클래스는 Circle 클래스로부터 상속 받은 클래스이고, 상위 클래스가 명시적으로 지정되지 않은 Circle 클래스의 상위 클래스는 묵시적으로 Object 클래스가 되며, 클래스의 계층은 아래의 그림과 같다.



• 따라서 자바에서 클래스의 구조는 Object 클래스를 최상위 클래스로 가지는 트리 구조를 가진다.

• super 예약어

• super 예약어는 하위 클래스의 멤버변수 및 메소드에 의하여 가려진 상위 클래스의 멤버변수 및 메소드에 접근할 때 또는 상위 클래스의 생성자를 호출하는 경우에 사용한다.

• 상위 클래스의 멤버변수 및 메소드의 접근

• 하위 클래스에 의하여 가려진 상위 클래스의 멤버변수 및 메소드에 접근할 때 예약어 super를 사용하며, 접근 형식은 다음과 같다.

super.객체변수
super.메소드이름(매개변수)

설명 :

상위 클래스의 객체변수 및 메소드의 접근은 super 지정어 및 연산자 .를 사용하여 접근한다.(예제 5참조)

예제 5

```
class ListString
{
   String name = "홍길동";
   public ListString(){};
   public void list(){
      System.out.println("상위 클래스의 출력문입니다.");
   }
}
class SubListString extends ListString
{
   String name = "임꺽정"; // 상위클래스의 멤버변수이름과 같다.
   public SubListString(){};
   public void list(){
      System.out.println("하위 클래스의 출력문입니다.");
   }
   public void write(){
      System.out.println("확장 클래스의 객체변수 name의 값은 "+this.name);
      System.out.println("상위 클래스의 객체변수 name의 값은 "+super.name);
      this.list();
      super.list();
   }
}
```

name 객체는 하위 클래스에서도 정의되었다.

list() 메소드는 하위 클래스에서도 정의되었다.(메소드 오버라이딩)

this은 호출한 객체를, super은 상위 클래스를 의미한다.

this는 호출한 객체를, super은 상위 클래스를 의미한다.

ExecuteSubListString.java(5)

```
public class ExecuteSubListString
{
   public static void main(String args[]){
      SubListString sub = new SubListString();
      sub.write();
   }
}
```

sub 객체는 상위 및 하위 클래스에서 정의한 name 및 list() 메소드를 포함한다.

write() 메소드에서 this 및 super 지정어를 사용하여 해당하는 변수 및 메소드에 접근한다

• **예제 설명**

SubListString 클래스의 출력문에서 사용된 this.name 및 this.list()의 문에서 this 대신에 객체이름 sub를 사용하면 컴파일 에러가 발생한다. 이유는 객체는 그 객체를 생성한 클래스에서만 사용가능하기 때문이며, 프로그램의 실행결과는 다음과 같다.

확장 클래스의 객체변수 name의 값은 임꺽정
상위 클래스의 객체변수 name의 값은 홍길동
하위 클래스의 출력문입니다.
상위 클래스의 출력문입니다.

• 상위 클래스의 생성자 호출

• 확장 클래스로부터 객체를 생성할 때 자동적으로 먼저 상위 클래스의 묵시적인 생성자(매개변수가 없는 생성자)를 실행하고 자신의 생성자를 실행한다.

예제 6

```
class Constructor
{
   int sup;
   public Constructor(){
      System.out.println("상위 클래스의 생성자입니다.");
   }
   public Constructor(int x){
      System.out.println("상위 클래스의 생성자입니다."+x);
   }
}
class SubConstructor extends Constructor
{
   int sub;
   public SubConstructor(){
      System.out.println("하위 클래스의 생성자입니다.");
   }
}
```

Constructor 클래스의 하위 클래스로부터 객체가 생성될 때 자동으로 실행되는 생성자

ExecuteSubConstructor.java(6)

```
public class ExecuteSubConstructor
{
   public static void main(String args[]){
      SubConstructor sub1 = new SubConstructor();
   }
}
```

SubConstructor의 상위 클래스의 묵시적인 Constructor() 생성자를 먼저 실행하고, SubConstructor() 생성자를 실행한다.

• **예제 설명**

네 번째 줄의 sub1객체는 먼저 SubConstructor 클래스의 상위 클래스인 Constructor 클래스의 묵시적인 생성자(첫 번째 생성자)로부터 객체가 생성되고, 그 후에 SubConstructor() 생성자로부터 객체를 생성한다. 따라서 프로그램의 실행결과는 다음과 같다.

상위 클래스의 생성자입니다.
하위 클래스의 생성자입니다.

*** 이렇기 때문에 상위 클래스의 멤버 변수 및 메소드를 이용할 수 있는 것이다.**

• 묵시적인 생성자가 아닌 상위 클래스의 생성자 호출

• 상위 클래스의 묵시적 생성자가 아닌 다른 생성자의 호출을 원한다면 예약어 super를 사용하며, 사용형식은 다음과 같다.

super(매개변수);

설명 : 매개변수를 포함한 상위 클래스의 생성자 호출은 매개변수를 지정한 super 지정어를 사용하며, super 문장은 반드시 생성자의 첫 번째 줄에 위치해야 한다.

예제 6

```
class Circle7
{
   static final double PI = 3.14;
   double radius;
   double area;
   public Circle7(double r){
      radius = r;
   }
   public void calculatingArea(){
      area = PI * radius * radius;
      System.out.println("원기둥의 밑면적은 “+area+"이다.”);
   }
}
```

반지름을 가진 객체를 생성한다.

ExecuteCylinder7.java(7)

```
class Cylinder7 extends Circle7
{
   double height;
   double volume;
   public Cylinder7(double r, double h){
      super(r);
      height = h;
   }
   public void calculatingVolume(){
      volume=area * height;
      System.out.println("원기둥의 부피는 ?volume+"이다.?;
   }
}
public class ExecuteCylinder7
{
   public static void main(String args[]){
      Cylinder7 vol = new Cylinder7(10, 5);
      vol.calculatingArea();
      vol.calculatingVolume();
   }
}
```

상위 클래스인 Circle7의 Circle7(double r) 생성자를 실행

2개의 생성자를 실행하므로 밑변의 반지름 및 높이를 가진 원기둥을 생성한다.

• 예제 실행결과

원기둥의 밑면적은 314.0이다.
원기둥의 부피는 1570.0이다.

• 메소드 재정의(overriding)

• 상위 클래스에서 정의한 메소드와 이름, 매개변수의 자료형 및 개수가 같으나 수행문이 다른 메소드를 하위 클래스에서 정의하는 것을 메소드 재정의라 한다.

• 이렇게 상위 클래스의 메소드를 하위 클래스에서 재정의할 때, 상위 클래스의 메소드는 하위 클래스의 메소드에 의하여 가려진다.

예제 8

```
class Overriding
{
   int vara, result1;
   public void assign(int a){
      vara = a;
      System.out.println("상위 클래스의 assign() 메소드입니다.");
   }
   public void sum(int a, int b, int c){
      result1 = a+b+c;
   }
}
class SubOverriding extends Overriding
{
   int varb;
   double result2;
   public void assign(int b){
      varb = b;
      System.out.println("하위 클래스에서 재정의된 assign() 메소드입니다.");
   }
   public void sum(double a, double b){
      result2 = a+b;
   }
}
```

assign(int b) 메소드는 상위 클래스의 assign(int a) 메소드를 재정의 한다.

2개의 sum() 메소드는 메소드 중복으로서 서로다른 메소드로 취급된다.

ExecuteSubOverriding.java(8)

```
public class ExecuteSubOverriding
{
   public static void main(String args[]){
      SubOverriding s1 = new SubOverriding();
      s1.assign(8);
      s1.sum(10, 20, 40);
      s1.sum(20.4, 34.5);
      System.out.println("세수의 덧셈값은 "+s1.result1+" 두수의 덧셈값은 "+s1.result2);
   }
}
```

재정의된 확장 클래스의 메소드가 실행된다.

• **예제 설명**

s1.assign(8)은 재정의된 하위 클래스의 assign(int b) 메소드를 실행하고, s1.sum(10, 20, 40); 문은 상위 클래스의 sum() 메소드를 실행하고, s1.sum(20.4, 34.5); 문은 확장 클래스의 sum() 메소드를 실행하므로 프로그램의 실행결과는 다음과 같다.

하위 클래스에서 재정의된 assign() 메소드입니다.
세수의 덧셈값은 70 두수의 덧셈값은 54.9

*** static 또는 final로 선언된 클래스 및 클래스 종단 메소드는 하위 클래스에서 재정의 할 수 없다.**

• 종단 메소드

• final 선택항목이 지정된 메소드를 종단 메소드라 하며, 종단 메소드는 하위 클래스에서 재정의하여 사용할 수 없다.

예제 9

```
class Triangle
{
   int width;
   int height;
   int area;
   final void calculatingTriangle(int w, int h){
      width=w;
      height=h;
      area = width * height / 2;
   }
}
class SubTriangle extends Triangle
{
   .......
   void calculatingTriangle(int a, int b){
      // 다른 기능을 하는 수행문
   }
}
```

종단 메소드이므로 하위 클래스에서 재정의할 수 없다.

종단 메소드를 재정의하므로 컴파일시 에러가 발생한다.

• 추상 클래스와 추상 메소드

• 수행문 없이({와 }가 없음) 정의된 메소드를 추상 메소드라 한다.

• 하나 이상의 추상 메소드를 포함한 클래스를 추상 클래스라 한다.

• 추상 메소드 및 추상 클래스는 abstract 지정어를 반드시 사용해야 하며, 선언형식은 다음과 같다.

```
[public] abstract class 클래스이름
{
  // 생성자
  // 멤버변수 선언
  [접근한정자] abstract 자료형 추상메소드이름(); // 추상 메소드
  [접근한정자] 자료형 메소드이름().....; // 클래스 및 객체 메소드
 }
```

설명 : 클래스는 추상 메소드를 포함하므로 abstract 선택항목을 사용하여 추상 클래스로 선언하였다.

* 추상 클래스는 구현되지 않은 추상 메소드를 포함하므로 객체를 생성할 수 없다. 따라서 추상 클래스를 상속받은 확장 클래스에서 모든 추상 메소드를 구현한 후에 확장 클래스로부터 객체를 생성한다.

Arithmetic.java(10)

```
abstract class Arithmetic
{
   int result;
   public abstract void calculate(int a, int b);
}
```

Arithmetic 클래스는 하나의 추상 메소드를 포함하므로 추상 클래스이다.

calculate() 추상 메소드는 {와 }가 없고, abstract 선택항목을 사용한다.

• **예제 설명 :** Arithmetic 추상 클래스는 2개의 정수변수를 매개변수로 가지는 calculate() 추상 메소드를 포함한다. 추상 메소드는 마지막에 { 와 }를 포함해서는 않되며, 마지막에 추상 메소드를 의미하는 ;을 반드시 추가해야 한다.

• **예제 11은 4개의 클래스가 Arithmetic 추상 클래스를 상속받아 calculate() 추상 메소드를 사칙 연산을 하도록 구현한 클래스이다.**

예제 11

```
class Add extends Arithmetic
{
  public void calculate(int a, int b){
    result = a+b;
    System.out.println("덧셈 결과는 "+result);
  }
}
class Subtract extends Arithmetic
{
  public void calculate(int a, int b){
    result = a-b;
    System.out.println("뺄셈 결과는 "+result);
  }
}
class Multiply extends Arithmetic
{
  public void calculate(int a, int b){
    result = a * b;
    System.out.println("곱셈 결과는 "+result);
  }
}
class Divide extends Arithmetic
{
  public void calculate(int a, int b){
    result = a / b;
    System.out.println("나눗셈 결과는 "+result);
  }
}
```

Add 클래스는 Arithmetic 클래스로부터 상속받아 calculate() 추상 메소드에 덧셈 기능을 구현하였다.

Subtract 클래스는 Arithmetic 클래스로부터 상속받아 calculate() 추상 메소드에 뺄셈 기능을 구현하였다.

Multiply 및 Divide 클래스도 각각 calculate() 추상 메소드에 곱셈 및 나눗셈을 구현한다. 이와 같이 추상 메소드를 포함한 추상 클래스를 상속받아 서로 다른 기능으로 추상 메소드를 구현할 수 있다.

ExecuteArithmetic.java(11)

```
public class ExecuteArithmetic
{
   public static void main(String args[]){
      Add add = new Add();
      add.calculate(100, 73);
      Subtract sub = new Subtract();
      sub.calculate(100, 73);
      Multiply mul = new Multiply();
      mul.calculate(100, 73);
      Divide div = new Divide();
      div.calculate(100, 73);
   }
}
```

calculate() 메소드는 객체에 따라 해당하는 사칙연산을 수행한다.

• **예제 설명**

add, sub, mul 및 div 객체는 자신을 생성시킨 클래스의 calculate() 메소드를 호출하여 각각의 사칙연산을 실행하며, 프로그램의 실행결과는 다음과 같다.

```
덧셈 결과는 173
뺄셈 결과는 27
곱셈 결과는 7300
나눗셈 결과는 1
```

*** 마지막으로 자바에서 확장 클래스는 하나의 상위 클래스만을 가질수 있다.**