제 4, 6, 7 장 클래스 개요

- 선택항목(option) 표현방법
- 클래스 작성
- 객체의 생성
- 멤버 변수(member variable)
- 생성자(constructor)
- 메소드
- 메소드의 반환값
- 메소드에 값 전달
- 메소드 중복(overloading)
- 정적 초기화자
- 내부 클래스(Inner Class)

• 선택항목(option) 표현방법

• 본책에서 클래스, 메소드 및 멤버변수에 사용되는 선택항목을 표현하는 방법을 먼저 예제 1를 가지고 설명한다.

예제 1

```
[public|<abstract||final>]
[<public||protected||private>|<abstract||final>|static|synchronized|native]
```

• **예제 설명**

모든 선택항목은 [와]사이에 표현한다. 그리고 여러 항목중 하나만 선택할 수 있는 경우는 ||로 분리하여 <와 >사이에 표현한다. 마지막으로 하나 이상을 선택할 수 있는 경우는 각 항목을 |로 분리한다.

예를 들면, 예제의 첫 번째 줄에서 public abstract, public final, public 또는 선택항목이 하나도 선택되지 않은 경우는 올바른 표현이다. 만일 public abstract final의 선택은 틀린 경우이다.

두 번째 줄에서 public abstract, private static synchronized, protected native등은 올바른 경우이며, public private final static은 틀린 경우이다.

• 클래스 작성

• 선택항목을 포함하지 않은 기본적인 클래스의 작성 방법을 배웠으나, 선택항목을 포함한 클래스의 작성 형식은 다음과 같다.

클래스 작성형식

```
[public|<final||abstract>] class 클래스이름
{
   //  멤버변수 선언 부분
   // 생성자 메소드 (클래스 이름과 동일한 메소드로 객체를 생성한다.)
   // 메소드 선언 부분
 }
```

• 설명

class는 클래스를 정의하는 지정어이고, { 와 }사이에 멤버변수, 생성자 메소드 및 메소드를 포함한다. 클래스 이름은 관례적으로 첫 문자를 대문자로 표현한다.

public은 패키지내의 클래스들 간의 접근의 유무를 나타내는 접근 한정자이고, <final||abstract>은 클래스의 특성을 결정하는 선택항목이다.

• 패키지란

자바에서 패키지는 유사한 기능을 가진 클래스들의 집합을 의미하며, 각각의 패키지에 포함된 클래스들은 특정한 디렉토리에 저장되어 있다.

아래 그림은 자바의 기본적인 패키지 및 디렉토리를 보여준다.

패키지 이름	클래스들을 저장한 디렉토리
java.lang	c:\jdk1.x.x\jre\lib\java\lang
java.util	c:\jdk1.x.x\jre\lib\java\util
java.io	c:\jdk1.x.x\jre\lib\java\io
java.net	c:\jdk1.x.x\jre\lib\java\net
java.awt	c:\jdk1.x.x\jre\lib\java\awt
java.awt.event	c:\jdk1.x.x\jre\lib\java\awt.event

위와 같이 클래스들이 서로 다른 디렉토리에 저장되어 있을 때, A라는 디렉토리에서 작성하는 클래스가 B라는 디렉토리에 저장된 클래스를 불러서 사용할 수 있느냐 없느냐는 클래스의 접근 한정자(**public**)에 따라 결정된다.

• 접근 한정자 public이 사용된 경우

public 접근 한정자를 사용하여 생성된 클래스는 모든 패키지 즉, 다른 디렉토리에 있는 클래스가 접근하여 사용할 수 있음을 의미한다

Triangle.java(2)

```
public class Triangle
{
   double width; // 삼각형의 밑변
   double height // 삼각형의 높이
   double area // 삼각형의 면적

   public void calculatingArea(){
      area = width * height / 2;
      System.out.println("삼각형의 면적은 ?area+"이다.”);
   }
}
```

멤버변수

메소드(기능을 구현한다.)

• 예제 설명

Triaggle.class 파일은 접근 한정자가 public이므로, 다른 디렉토리에서 임포트하여 객체를 생성하여 사용할 수 있다.

- default (접근한정자가 없는 경우)

  예제 2에서 public 선택항목을 사용하지 않는 경우이며, 이러한 클래스는 같은 디렉토리(패키지)에 있는 클래스들 만이 접근하여 사용할 수 있음을 의미한다.

<final||abstract> 선택항목은 다음과 같이 클래스의 특성을 결정한다.

- final

  final 선택항목을 가진 클래스를 종단 클래스라 하며, 종단 클래스는 어떤 클래스의 상위 클래스가 될 수 없다.

- abstract

  abstract 선택항목을 가진 클래스를 추상 클래스라 하고, 추상 클래스는 객체를 생성할 수 없는 클래스이다.

- default (final 및 abstract 선택항목이 없는 경우)

  final 및 abstract가 지정되지 않은 클래스는 어떤 클래스의 상위 클래스가 될 수 있고, 객체를 생성할수 있는 일반적인 클래스를 의미한다.

• 객체의 생성

앞에서 작성한 클래스의 변수를 할당받고, 메소드를 실행하기 위해서는 클래스로부터 객체를 생성해야 하며, 클래스로부터 객체를 선언하는 형식은 다음과 같다.

- 첫 번째 객체 선언 형식

클래스이름 객체이름; // 객체 선언

객체이름 = new 생성자메소드; // 객체의 생성(클래스의 기억공간 할당)

설명 : 첫 번째 줄은 클래스에 해당하는 객체를 선언하고, 두 번째 줄은 객체를 생성한다.
(클래스에서 선언한 변수를 할당 받음.)

- 두 번째 객체 선언 형식

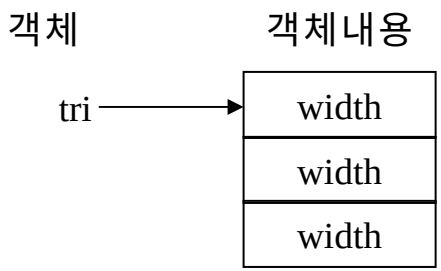
클래스이름 객체이름 = new 생성자메소드; // 객체의 선언 및 생성

설명 : 클래스에 해당하는 객체을 선언과 동시에 객체를 생성한다.

ExecuteTriangle.java(3)

```
public class ExecuteTriangle
{
   public static void main(String args[]){
      Triangle tri = new Triangle();
      tri.width=100;
      tri.height=10;
      tri.calculatingArea();
   }
}
```

생성된 tri 객체는 Triangle 클래스에서 선언한 객체변수를 할당받는다.



• **예제 설명**

네 번째 줄은 예제 2의 Triangle.class 클래스로부터 tri 객체를 생성하고(3개의 변수 할당), 다음 두줄은 할당된 변수에 값을 대입하고, 마지막 명령은 calculatingArea(); 메소드를 실행하므로 프로그램의 실행결과는 다음과 같다.

삼각형의 면적은 500.0이다.

• 자바 프로그램의 작성 요약

1. 객체를 나타내는 클래스를 작성한다.

2. 라이브러리 클래스 또는 1)에서 작성한 클래스로부터 객체를 생성한다.

(***생성된 객체는 클래스에서 정의한 변수를 할당받는다.**)

3. [객체이름.멤버변수]의 형식으로 객체에 할당된 멤버변수에 데이터를 쓰거나 읽고, [객체이름.메소드]의 형식으로 객체를 생성시킨 클래스의 메소드를 실행시킨다.

• 멤버 변수(member variable)

클래스의 메소드 밖에서 선언된 변수를 멤버변수라 하며, 일반적으로 멤버변수는 첫 문자를 소문자로 시작한다.

멤버변수는 선택항목으로 접근 한정자 및 소유 제한자 등을 가지며, 선택항목의 사용 유무에 따라 객체변수, 클래스 변수 또는 종단변수 등으로 구분된다.

• 객체변수

static 선택항목을 갖지 않은 변수를 객체변수라 하며, 객체가 가질수 있는 특성 및 활동을 나타낸다. 즉, 객체가 할당된 변수를 소유하며, 선언형식은 다음과 같다.

[<public||private||protected>|transient|volatile] 자료형 변수이름;

설명 : 소유 제한자 static 선택항목이 없으므로 선언된 변수는 객체변수이며, 자료형은 int, char 등과 같은 기본 자료형을 나타내는 지정어, 배열, 문자열 등과 같은 객체를 나타낸다.

<public||private||protected> 선택항목은 접근 한정자로서 선언된 변수의 접근 유무를 나타내며, 다음과 같이 4가지 종류가 있다.

- **public**

  변수를 선언한 클래스가 접근 가능하면, 다른 디렉토리(패키지)에 있는 모든 클래스에서 접근하여 읽고 쓸 수 있다는 것을 의미한다.

- **private**

  변수를 선언한 클래스에서만 접근하여 사용할 수 있으며, 하위 클래스에도 상속되지 않는다.

- **protected**

  변수를 선언한 클래스와 같은 패키지에 있는 클래스 또는 그 변수를 선언한 클래스의 하위 클래스에서만 접근할 수 있다.

- **default(접근한정자가 없는 경우)**

  변수를 선언한 클래스와 같은 패키지에 있는 클래스에서만 접근할 수 있다.

• 접근 한정자에 따른 패키지사이의 접근 유무는 다음과 같다.

클래스	public	protected	private	default
같은 클래스	가능	가능	가능	가능
같은 패키지안의 모든 클래스	가능	가능	불가능	가능
패키지 밖의 다른 클래스	가능	불가능	불가능	불가능
같은 패키지안의 서브 클래스	가능	가능	불가능	가능
다른 패키지에 있는 서브 클래스	가능	가능	불가능	불가능

• transient는 수명 제한자로서 한번만 사용하고 버리는 객체변수를 의미하고, 쓰레드에서 사용하는 volatile은 휘발성 변수를 의미한다.

• **객체변수의 접근 형식**

객체이름.접근하고자 하는 객체변수

다시 생각하자, 객체변수의 소유는 객체라는 것을.....

ExecuteCircle.java(4)

```
class Circle
{
   public double pi; // 객체변수
   public double radius; // 객체변수
   public double area; // 객체변수
   public void calculatingCircle(){
      area = pi * radius * radius;
      System.out.println("원의 면적은 "+area+"이다.”);
   }
}
public class ExecuteCircle
{
   public static void main(String args[]){
      Circle circle = new Circle(); // circle 객체 생성
      circle.pi=3.14;
      circle.radius=10;
      circle.calculatingCircle();
   }
}
```

pi, radius 및 area는 객체변수

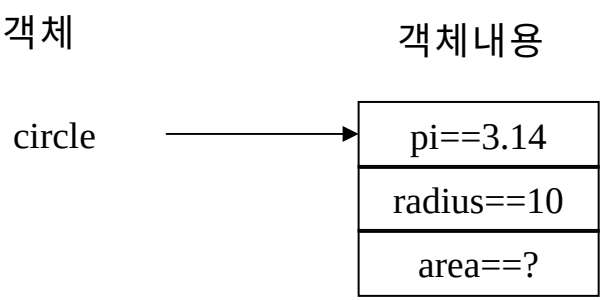
circle 객체에 3개의 객체변수가 할당된다.

메소드를 실행할때 사용되는 변수는 circle 객체에 할당된 값이 사용된다.

- **예제 설명**

circle 객체에 할당된 객체변수의 기억공간 할당형태 및 프로그램 실행결과는 다음과 같다.

원의 면적은 314.0이다



Person.java(5)

```
class Person
{
   private String name;
   private int age;
   public Person(){}; //생성자 메소드
   public void PrintPerson(){
      System.out.print("나의 이름은"+name);
      System.out.println(", 나는 "+age+" 살이다");
   }
   public static void main(String args[]){
      Person p = new Person();
      p.name="홍 길동";
      p.age=41;
      p.PrintPerson();
   }
}
```

Person 클래스에서만 접근됨.

같은 클래스이므로 접근이 가능

• **예제 설명**

세 번째 및 네 번째 줄 명령은 문자열 객체 name 및 정수변수 age를 선언하고, 2개의 객체 변수는 접근 한정자가 private이므로 이 변수를 선언한 Person 클래스에서만 접근가능하며, 프로그램의 실행결과는 다음과 같다.

나의 이름은 홍 길동, 나이는 41 살이다.

ExecutePerson.java(6)

```
class Person
{
   private String name;
   private int age;
   public Person(){}; //생성자 메소드
   public void PrintPerson(){
      System.out.print("나의 이름은"+name);
      System.out.println(", 나는 "+age+" 살이다");
   }
}
public class ExecutePerson
{
   public static void main(String args[]){
      Person p = new Person();
      p.name="홍 길동";
      p.age=41;
      p.PrintPerson();
   }
}
```

Person 클래스에서만 접근됨.

다른 클래스이므로 접근이 불가능

• **예제 설명**

ExecutePerson.java 소스 파일을 컴파일하면 ExecutePerson 클래스의 p.name 및 p.age 에서 [Undefined variable name or age] 에러 메시지를 발생한다. 즉, private 접근 한정자를 가지는 name 및 age 변수는 Person 클래스이외의 다른 클래스에서는 접근이 불가능하다.

• 클래스 변수

다수의 객체가 공유하여 사용하는 변수를 클래스 변수라 하며, static 예약어를 사용하여 정의한다. 즉, 클래스가 변수를 소유하며, 선언형식은 다음과 같다.

[<public||private||protected>static[final]] 자료형 변수이름;

설명 : static으로 지정되어 선언된 모든 변수는 클래스 변수이고, 선택항목으로 접근 한정자 및 기호상수를 정의하는 final을 선택할 수 있다.

- 접근 한정자 <public||private||protected> 선택항목은 객체변수에서 설명된 기능과 같다.
- final이 사용되면 클래스 종단 변수로서 기호상수를 의미한다.(6.6.3절에서 설명)
- 클래스 변수는 일반적으로 선언과 함께 초기값을 준다.

- **객체변수는 생성된 각각의 객체들이 소유하고, 클래스 변수는 클래스가 소유한다. 이러한 이유 때문에 static 제한자를 소유 제한자라 한다.**
- **즉, 클래스 변수는 생성되는 각 객체에 할당되지 않고 클래스내에 기억공간이 할당되어, 생성된 모든 객체가 공유하며, 접근형식은 다음과 같다.**

클래스이름.접근하고자 하는 클래스 변수

ExecuteSerialNumber.java(7)

```
class SerialNumber
{
   public int serialNo;
   public static int noOfobject=0;

   public SerialNumber(){ // 클래스 변수 noOfobject를 1씩 증가시키는 생성자
      SerialNumber.noOfobject++;
   }
}
public class ExecuteSerialNumber
{
   public static void main(String args[]){
      SerialNumber circle1 = new SerialNumber(); // circle1 객체생성
      circle1.serialNo = SerialNumber.noOfobject;
      SerialNumber circle2 = new SerialNumber();
      circle2.serialNo = SerialNumber.noOfobject;
      SerialNumber circle3 = new SerialNumber();
      circle3.serialNo = SerialNumber.noOfobject;
      System.out.println("생성된 객체의 수는 "+SerialNumber.noOfobject+"이다");
   }
}
```

serialNo은 객체변수

noOfobject는 클래스 변수로서 생성된 모든 객체가 공유

circle1 객체는 serialNo
객체변수만을 할당받는다.

클래스이름을 통하여 접근

- **예제 설명**

SerialNumber() 생성자 메소드는 객체를 생성할때 noOfobject 클래스변수의 값을 1씩 증가시키므로, 프로그램의 실행결과는 다음과 같다. **생성된 객체의 수는 3이다.**

• 종단 변수

• 선택항목 final를 사용하여 선언한 변수를 종단변수라 하며, 프로그램에서 변하지 않는 상수값을 선언할 때 이용한다.

• 관례적으로 종단변수는 대문자를 사용하며, 대부분 클래스 변수로 선언된다.

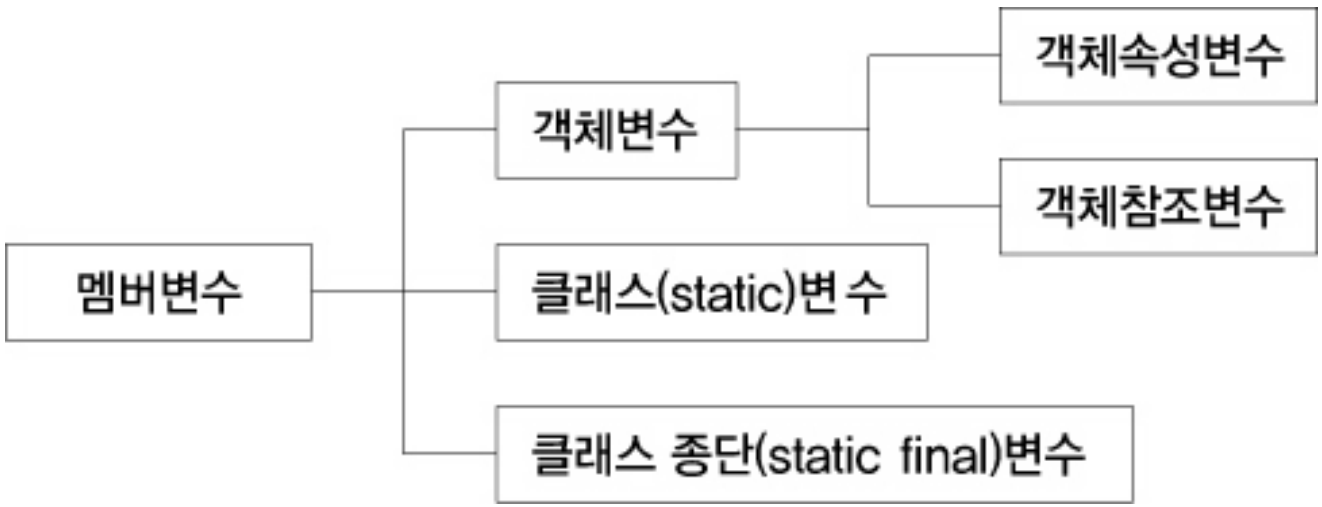
ExecuteCircleStatic.java(8)

```
class CircleStatic
{
   public static final double PI=3.14;
   public double radius;
   public double area;
   public CircleStatic(); // 생성자 메소드
   public void calculatingCircle()
      area = PI * radius * radius;
      System.out.println("원의 면적은 "+area+"이다.”);
   }
}
public class ExecuteCircleStatic
{
   public static void main(String args[]){
      CircleStatic circle = new CircleStatic(); // circle 객체생성
      circle.radius=100;
      circle.calculatingCircle();
   }
}
```

클래스 종단변수(PI)

PI는 3.14로 사용된다.

• 아래의 그림은 지금까지 선택항목에 따라 구분된 변수를 보여 주며, 객체속성변수는 자바에서 지원하는 8개의 기본 자료형 변수를 의미한다.



• 변수의 유효범위

• 변수의 유효범위는 선언된 변수가 사용될 수 있는 영역을 의미한다.

• 변수는 클래스의 메소드밖에서 선언된 멤버변수 및 메소드내에서 선언된 메소드 변수가 있다.

• 멤버변수의 유효범위는 클래스 전체에서 사용 가능하며, 메소드 변수는 지역변수로서 선언된 메소드내에서만 사용 가능하다.

VariableScope.java(9)

```
class VariableScope
{
   static int serial;
   String name;
   …
   public void display(String mes){
      int x;
      …
   }
}
```

메소드 변수 mes 및 x는 display 메소드에서만 사용 가능하다.

멤버변수 serial 및 name은 VariableScope 모든 영역에서 사용가능하다.

• 생성자(constructor)

• 생성자는 객체가 생성될 때 객체의 초기화 과정을 기술하는 특수한 메소드이다.

• 생성자는 일반 메소드와 같이 연산자 .를 이용하여 명시적으로 호출되지 않고, 객체를 생성할 때 new 연산자에 의하여 자동으로 실행되며, 생성자의 형식은 다음과 같다.

```
[<public||private||protected>] 클래스이름(매개변수){
        ..... // 수행문
}
```

형식설명

다른 항목은 일반 메소드와 사용형태가 같으나, 생성자는 일반 메소드와 달리 반환하는 자료형이 없고, 이름은 반드시 클래스 이름과 같아야 한다는 차이점이 있다.

• 생성자도 매개변수 및 수행문을 포함할 수 있다.

• 이는 클래스로부터 객체를 생성할 때, 생성자 메소드에 매개변수를 전달하고 생성자의 수행문을 실행하여 **특정한 객체를 생성할 수 있다는** 의미이다.

CircleConstructor.java(10)

```
public class CircleConstructor
{
   public static final double PI=3.14;
   public double radius;
   public double area;
   public CircleConstructor(double a){ // 하나의 정수변수를 가지는 생성자
      radius = a;
   }
   public void calculatingCircle(){
      area = PI * radius * radius;
      System.out.println("원의 면적은 "+area+"이다.”);
   }
}
class ExecuteCircleConstructor
{
   public static void main(String args[]){
      CircleConstructor circle = new CircleConstructor(100);
      circle.calculatingCircle();
   }
}
```

PI : 클래스 종단변수
radius, area : 객체변수

a로 전달받은 값을 반지름(radius)으로 가진 객체를 생성

생성자에 실 매개변수
100을 전달한다.

• **예제 설명**

실행 클래스의 네 번째 줄은 radius 객체변수의 값이 100인 circle 객체를 생성한다. 즉, 객체 radius는 생성될 때 반지름의 값을 가진 원을 나타내며, 실행결과는 다음과 같다.

원의 면적은 314.0이다.

ExecutePerson.java(11)

```
class Person
{
   public String name;
   public int age;
   public Person(String x, int y){
      name = x;
      age = y;
   } //생성자 메소드
   public void printPerson(){
      System.out.print("나의 이름은"+name);
      System.out.println(", 나는 "+age+" 살이다");
   }
}
public class ExecutePerson
{
   public static void main(String args[]){
      Person p1 = new Person("홍 길동", 41);
      Person p2 = new Person("임 꺽정", 41);
      p1.printPerson();
      p2.printPerson();
   }
}
```

name, age : 객체변수

이름 및 나이를 가진 객체를 생성한다.

생성자에 실 매개변수
이름 및 나이를 전달한다.

- **예제 설명 :** 2 사람의 객체를 생성하며, 프로그램의 실행결과는 다음과 같다.

나의 이름은 홍 길동, 나이는 41 살이다.
나의 이름은 임 꺽정, 나이는 41 살이다.

• 생성자 오버로딩(overloading)

• 하나의 클래스에 매개변수의 자료형과 개수가 서로 다른 다수의 생성자를 포함하여 다양한 객체를 생성하는 것을 생성자 오버로딩이라 한다.

Car.java(12)

```
public class Car
{
   String standard = "기본형"; // 문자열 객채변수
   String option1; // 문자열 객체변수
   String option2; // 문자열 객체변수
   public Car(){ // 기본형차를 생성하는 매개변수가 없는 생성자
      System.out.println(standard+" 자동차이다");
   }
   public Car(String opt1){ // 기본형+opt1 선택품목을 가진 차를 생성함.
      option1 = opt1;
      System.out.println(standard+"+"+option1+" 자동차이다");
   }
   public Car(String opt1, String opt2){ // 기본형+opt1+opt2 선택품목을 가진 자동차.
      option1 = opt1;
      option2 = opt2;
      System.out.println(standard+"+"+option1+"+"+option2+" 자동차이다");
   }
   public void drawCar(String opt1, String opt2){ // 자동차를 그리는 메소드
    }
}
```

3개의 생성자는 매개변수의 자료형과 개수가 서로 다르다.

ExecuteCar.java(13)

```
public class ExecuteCar
{
   public static void main(String args[])
      Car car1 = new Car();
      Car car2 = new Car("가죽쉬트");
      Car car3 = new Car("가죽쉬트", "에어백");
   }
}
```

매개변수가 없는 생성자를 호출한다.

매개변수가 하나인 생성자를 호출한다.

매개변수가 두개인 생성자를 호출한다.

• **예제 설명**

객체를 생성할 때 호출되는 생성자는 자동으로 매개변수의 자료형과 수가 일치하는 생성자가 실행된다. 따라서 첫 번째 생성자는 예제 12의 매개변수가 없는 생성자를 실행하고, 두 번째 생성자는 매개변수가 하나인 생성자를 실행하며, 세 번째 생성자는 매개변수가 두개인 생성자를 실행하므로 프로그램의 실행결과는 다음과 같다.

기본형 자동차이다

기본형+가죽쉬트 자동차이다

기본형+가죽쉬트+에어백 자동차이다

• this 예약어

• this 예약어는 생성자나 메소드의 매개변수가 멤버변수와 같은 이름을 사용하는 경우에 사용되어, 이름이 같은 두 변수를 구분할 수 있도록 한다.

Triangle.java(14)

```
public class Triangle
{
   double width; // 삼각형의 밑변
   double height // 삼각형의 높이
   double area // 삼각형의 면적
   public void Triangle(double width, double height){
      width = width;
      height = height;
   }
}
```

생성자 및 메소드의 매개변수의 이름이 같으므로 구분을 할 수 가 없다.

• 예제 설명

일곱 및 여덟 번째 줄의 width = width; height = height;의 의미는 생성자에 전달된 매개변수의 값을 객체변수에 할당하라는 의미이나, 실제로 컴파일러는 매개변수의 값을 매개변수에 다시 할당하는 형식이 되어 아무런 의미를 가지지 못한다.

따라서 예제 15와 같이 this 예약어를 사용하여 2개의 매개변수를 구분한다.

Triangle.java(15)

```
public class Triangle
{
   double width; // 삼각형의 밑변
   double height // 삼각형의 높이
   double area // 삼각형의 면적
   public void Triangle(double width, double height){
      this.width = width;
      this.height = height;
   }
}
```

this.width에서 width는 객체변수를 의미한다.
즉, this는 호출한 객체를 의미한다.

• **예제 설명**

어떤 클래스에서 Triangle tri = new Triangle(10, 20); 수행문을 실행한다고 가정하자. 이 때 10 및 20 정수값이 각각 생성자의 매개변수 width 및 height에 전달되고, 생성자의 첫 번째 수행문 this.width=width;에서 this는 현재의 객체를 나타내므로 호출한 객체 tri를 의미하므로 전달된 width 매개변수의 값 10을 tri 객체가 가르키는 width 객체변수에 할당한다. 다음 줄의 this.height = height;도 같은 의미이다.

• 클래스에 다수의 생성자가 존재하는 경우 this(매개변수); 형식을 사용하여 생성자 메소드를 간단하게 작성할 수 있다.

Car.java(16)

```
public class Car
{
   String standard = "기본형"; // 문자열 변수선언
   String option1; // 문자열 변수 선언
   String option2; // 문자열 변수 선언
   public Car(){
      this("", "");
   }
   public Car(String opt1){
      this(opt1, "");
   }
   public Car(String opt1, String opt2){
      option1 = opt1;
      option2 = opt2;
      System.out.println(standard+"+"+option1+"+"+option2+" 자동차이다");
   }
   public void drawCar(String opt1, String opt2){
      // 자동차를 그리는 메소드
    }
}
```

세 번째 Car 생성자(수행문이 있음)를 호출한다.

세 번째 Car 생성자를 호출한다.

모든 매개변수를 포함한 생성자
생성자의 마지막 위치에 존재해야 함

• 메소드

• 메소드는 대입문 및 제어문 등으로 구성되어 특정한 기능을 수행하는 프로그램이며, 기능면에서 C 언어의 함수와 같다. 일반적으로 메소드는 첫 문자를 소문자로 시작한다.

• 메소드는 선택항목으로 접근 한정자 및 소유 제한자 등을 가지며, 선택항목의 사용 유무에 따라 객체 메소드, 클래스 메소드 또는 종단 메소드등으로 구분된다.(멤버변수와 유사함)

• 객체 메소드

static 선택항목을 갖지 않은 메소드를 객체 메소드라 하며, 객체를 통하여 접근한다. 따라서 호출한 메소드가 실행될 때 호출한 객체에 할당된 객체변수의 값이 사용되며, 객체 메소드를 선언하는 형식은 다음과 같다.

```
[ < public||private||protected>|final|synchronized] 반 환 값 유 형
메소드이름([매개변수들]){
        ..... // 메소드변수 선언 및 수행문
}
```

설명 : static, abstract 선택항목을 갖지 않는 메소드를 의미하며, synchronized 선택항목은 쓰레드에서 사용되는 선택항목으로 11 장에서 설명한다. 접근 한정자 및 final 선택항목은 멤버변수의 내용과 같다.

• 객체변수는 클래스로부터 생성된 객체에 별도로 할당되나, 객체 메소드는 프로그램 코드로서 다수의 객체가 접근하여 사용할 수 있다.
• 즉, 클래스에 존재하는 객체 메소드들은 그 클래스로부터 생성된 모든 객체에 의하여 접근되며, 접근형식은 다음과 같다.

객체이름.접근하고자 하는 메소드 이름

ExecuteStudent.java(17)

```
class Student
{
   protected int first;
   protected int second;
   protected int third;
   private int total;
   public Student(int first, int second, int third){
      this.first = first;
      this.second = second;
      this.third = third;
   }
   public int studentnum(){
      total=first+second+third;
      return total;
   }
}
public class ExecuteStudent
{
   public static void main(String args[]){
      int count; // main() 메소드에서만 사용되는 메소드 변수
      Student st = new Student(37, 40, 42);
      count = st.studentnum();
      System.out.println("객체 st의 총 인원은 "+count);
      System.out.println("객체 st의 first반의 인원은 "+st.first);
      System.out.println("객체 st의 second반의 인원은 "+st.second);
      System.out.println("객체 st의 third반의 인원은 "+st.third);
   }
}
```

같은 패키지 및 하위 클래스에서 사용가능(객체변수)

접근 한정자가 Private이므로 Student 클래스에서만 사용가능

this 예약어를 사용하여 객체변수에 값을 할당한다.

다른 클래스에 total 값을 반환하는 객체 메소드

Student 클래스의 total(Private) 값을 구한다.

Studentp.java(18)

```
class Studentp
{
   private int first;
   private int second;
   private int third;
   private int total;
   public Studentp(int first, int second, int third){
      this.first = first;
      this.second = second;
      this.third = third;
      this.studentnum();
   }
   private void studentnum(){ // 다른 클래스에서는 접근이 안됨.
      total=first+second+third;
   }
   public int gettotal(){
      return total;
   }
   public int getfirst(){
      return first;
   }
   public int getsecond(){
      return second;
   }
   public int getthird(){
      return third;
   }
}
```

접근 한정자가 Private이므로 Student 클래스에서만 사용가능

객체를 생성할 때 total 객체변수에 초기값을 할당한다.

Private로 선언된 4개의 객체변수의 값을 반환한다.

ExecuteStudentp.java(18)

```
public class ExecuteStudentp
{
   public static void main(String args[]){
      int count;
      Studentp st = new Studentp(37, 40, 42);
      count = st.gettotal();
      System.out.println("객체 st의 총 인원은 "+count);
      System.out.println("객체 st의 first반의 인원은 "+st.getfirst());
      System.out.println("객체 st의 second반의 인원은 "+st.getsecond());
      System.out.println("객체 st의 third반의 인원은 "+st.getthird());
   }
}
```

Private로 선언된 객체변수의 값을 구한다.

예제 설명 : Studentp 클래스에서 정의한 학생의 총수를 계산하는 studentnum() 객체 메소드의 접근 한정자는 private이므로 생성자의 네 번째 명령에서 studentnum() 메소드를 호출하여 total 변수의 값을 구하고, 4개의 객체변수는 실행 클래스에서 해당하는 메소드를 호출하여 4개의 객체변수의 값을 구하였다.

• 클래스 메소드

static 선택항목을 명시한 메소드를 클래스 메소드라 하며, 클래스 변수와 마찬가지로 클래스 이름을 통하여 접근할 수 있다. 이러한 이유로 클래스 메소드내에서는 클래스 변수만을 사용할 수 있으며, 클래스 메소드를 선언하는 형식은 다음과 같다.

```
<public||private||protected>] static [final|synchronized] 반환값유형 메소드이름([매개변수들]){
        ..... // 메소드변수 선언 및 수행문
}
```

설명 : static 선택항목을 사용하는 것만 다를 뿐, 다른 항목은 객체 메소드의 내용과 같다.

• 클래스 메소드는 객체의 생성과정 없이 직접 클래스 이름을 통하여 접근하며, 접근형식은 다음과 같다.

클래스 이름.접근하고자 하는 클래스 메소드 이름

ExecuteCircleNumber.java(19)

```
class CircleNumber
{
   public int serialno;
   static int noOfobject=1;
   public CircleNumber(){};
   public static int numberOfCircle(){
      return CircleNumber.noOfobject++;
   }
}

public class ExecuteCircleNumber
{
   public static void main(String args[]){
      CircleNumber circle1 = new CircleNumber();
      circle1.serialno = CircleNumber.numberOfCircle();
      CircleNumber circle2 = new CircleNumber(); // circle2 객체생성
      circle2.serialno = CircleNumber.numberOfCircle();
      CircleNumber circle3 = new CircleNumber(); // circle3 객체생성
      circle3.serialno = CircleNumber.numberOfCircle();
      System.out.println("circle1의 일련번호는 : "+circle1.serialno);
      System.out.println("circle2의 일련번호는 : "+circle2.serialno);
      System.out.println("circle3의 일련번호는 : "+circle3.serialno);
   }
}
```

클래스 변수를 선언하고 초기값을 할당한다.

클래스 메소드로서 noOfobject 클래스 변수값을 1증가시킨다.

클래스 이름을 통하여 클래스 베소드를 호출한다. 반환값을 객체변수에 할당함.

각 객체의 serialno 객체변수 값을 출력한다.

예제 실행결과

```
circle1의 일련번호는 : 1
circle2의 일련번호는 : 2
circle2의 일련번호는 : 3
```

CountInstances.java(20)

```
public class CountInstances
{
   private static int numInstances = 0;
   protected static int getNumInstances(){
      return numInstances;
   }
   public CountInstances(){
      CountInstances.addInstances();
   }
   private static void addInstances(){
      numInstances++; // 클래스 이름 생략가능
   }
   public static void main(String args[]){
      System.out.println("인스턴스의 시작값: "+CountInstances.getNumInstances());
      for(int i=0; i<10; ++i)
         new CountInstances();
      System.out.println("생성된 인스턴스의 수: "+CountInstances.getNumInstances());
   }
}
```

클래스 변수를 선언하고 초기값을 할당한다.

클래스 메소드로서 numInstances 클래스 변수값을 반환

클래스내에서 클래스 메소드를 접근하면 클래스 이름 생략 가능함

생성자에서 클래스 메소드(addInstances())를 호출한다.

클래스 메소드로서 클래스 변수값을 1증가시킨다.

클래스 메소드(getNumInstances())를 호출한다.

예제 실행결과

인스턴스의 시작값: 0
생성된 인스턴스의 수: 10

• 종단 메소드

• 선택항목 final이 객체 메소드에 선언되면 객체 종단 메소드, 클래스 메소드에 선언되면 클래스 종단 메소드라 한다.

• 종단 메소드를 포함하는 클래스로부터 하위 클래스를 생성할 때, 하위 클래스에서 종단 메소드를 재정의(overriding)하여 사용할 수 없는 메소드를 의미한다.(7장에서 설명)

• 추상 메소드

• abstract 선택항목을 갖는 메소드를 추상 메소드라 하며, 수행문을 가질수 없다.

• 하나 이상의 추상 메소드를 포함한 클래스를 추상 클래스라 하며, 추상 메소드의 선언형식은 다음과 같다.

public abstract 반환값유형 메소드이름([매개변수들]);

설명 : 추상 메소드는 abstract로 선언해야 하며, 접근 한정자는 public만을 사용할 수 있다. 그리고 추상 메소드는 메소드의 수행문을 포함하는 {와 }문이 없으며, 반드시 하위 클래스에서 재정의하여 사용해야 한다.(7.4절에서 자세히 설명)

• native 메소드(시스템 고유함수 사용)

• native 선택항목을 가진 메소드는 자바이외의 플렛폼 의존적인 C, C++등의 언어로 작성된 메소드를 의미한다.

• 메소드의 반환값

• 반환값 유형은 메소드를 실행했을 때 반환하는 결과값의 자료형을 의미하며, 반환값 유형은 선택항목이 아닌 필수항목이다.(생성자 메소드는 없음)

• 반환값 유형을 나타내는 자료형은 int, float, String, 객체이름 및 void등이 있다.

• 정수값을 반환하는 메소드

ExecuteReturnInteger.java(21)

```
class ReturnInteger
{
   int result;
   int value;
   public int calculatingArea(){ // int는 반환되는 자료형을 의미한다.
      result = value * value;
      return result;
   }
}
public class ExecuteReturnInteger
{
   public static void main(String args[]){
      ReturnInteger obj = new ReturnInteger();
      obj.value=10;
      System.out.println("결과값은 "+obj.calculatingArea());
   }
}
```

메소드의 반환값 자료형

호출한 메소드에 정수값을 반환한다.

메소드를 실행하면 정수값을 반환받는다.

예제 설명 및 실행결과 : obj.claculatingArea()은 claculatingArea()에서 반환된 result 값 100이 되므로 화면에 다음의 결과를 출력한다.

결과값은 100

• 문자열 객체를 반환하는 메소드

ExecuteBook.java(22)

```
class Book
{
   String display;
   public String message(){
      display = "자바 배움터 교재";
      return display;
   }
}
public class ExecuteBook
{
   public static void main(String args[]){
      Book obj = new Book();
      System.out.println("결과값은 "+obj.message());
   }
}
```

메소드의 반환값 자료형(문자열)

문자열(객체)를 반환한다.

메소드를 실행하면 문자열을 반환받는다.

예제 설명 및 실행결과 : obj.message()은 message()에서 반환하는 display 객체가 가르키는 문자열인 “자바 배움터 교재”이므로 화면에 다음의 결과를 출력한다.

결과값은 자바 배움터 교재

- void 자료형를 반환하는 메소드

ExecuteBookVoid.java(23)

```
class BookVoid
{
   public void message(){
      System.out.println("결과값은 자바 배움터 교재");
   }
}
public class ExecuteBookVoid
{
   public static void main(String args[]){
      BookVoid obj = new BookVoid();
      obj.message();
   }
}
```

void는 실행결과 어떤 값도 반환하지 않는다.

메소드에서 직접 메시지를 화면에 출력한다.

예제 설명 및 실행결과 : message() 메소드는 결과값을 반환하지 않고 메시지를 화면에 출력하므로 화면에 다음의 결과를 출력한다.

결과값은 자바 배움터 교재

• 메소드에 값 전달

ExecuteArea.java(24)

```
class Area
{
   public int calculatingArea(int width, int length){
      int result;
      result = width * length;
      return result;
   }
}
public class ExecuteArea
{
   public static void main(String args[]){
      int area;
      Area rec = new Area();
      area = rec.calculatingArea(20, 30);
      System.out.println("면적은 "+area);
   }
}
```

형식 매개변수로서 자료형 및 변수가 반드시 선언되어야 하며, 초기값은 호출한 메소드로부터 전달받는다.

호출하는 메소드의 실 매개변수로서 반드시 형식 매개변수의 자료형 및 개수와 일치하는 상수값을 전달해야 한다.

예제 설명 및 실행결과 : 프로그램의 실행결과는 다음과 같다.

면적은 600

ExecuteSwap.java(25)

객체가 매개변수로 사용된다.

```
class Swap
{
   public String x, y;
   public static void change(Swap obj){
      String temp;
      temp = obj.x; obj.x = obj.y; obj.y = temp;
   }
}
public class ExecuteSwap
{
   public static void main(String args[]){
      Swap object = new Swap();
      object.x = "영화 글레디에이터"; object.y = "영화 퍼펙트 스톰";
      System.out.println("바꾸기전: x= "+object.x+", y= "+object.y);
      Swap.change(object); // object 객체를 매개변수로 전달한다.
      System.out.println("바뀐후: x= "+object.x+", y= "+object.y);
   }
}
```

예제 설명 및 실행결과 : 프로그램은 객체를 매개변수로 전달하여 두 문자열을 교환하며, 프로그램의 실행결과는 다음과 같다.

바꾸기전: x= 영화 글레디에이터, y=영화 퍼펙트 스톰
바뀐후: x= 영화 퍼펙트 스톰, y=영화 글레디에이터

• 메소드 중복(overloading)

• 하나의 클래스에 이름은 같으나 매개변수의 자료형과 개수가 서로 다른 다수의 메소드를 사용하는 것을 메소드 오버로딩이라 하며, 이러한 내용을 다형성(polymorphism)이라 한다.

Arithmetic.java(26)

```
class Arithmetic
{
   double result;
   public void sum(int a, int b){
      result = a+b;
      System.out.println("두 정수의 합은 "+result);
   }
   public void sum(int a, int b, int c){
      result = a+b+c;
      System.out.println("세 정수의 합은 "+result);
   }
   public void sum(double a, double b){
      result=a+b;
      System.out.println("두 실수의 합은 "+result);
   }
}
```

3개의 sum() 메소드는 이름은 같으나, 매개변수의 자료형과 개수가 서로 다르므로 별도의 메소드로 간주된다.

ExecuteArithmetic.java(26)

```
public class ExecuteArithmetic
{
   public static void main(String args[]){
      Arithmetic hap = new Arithmetic();
      hap.sum(10, 20);
      hap.sum(100, 200, 300);
      hap.sum(27.7, 60.6);
   }
}
```

3개의 메소드는 자동으로 매개변수의 수와 자료형이 같은 sum() 메소드를 실행한다.

예제 설명 및 실행결과

ExecuteArithmetic 실행 클래스의 hap.sum(10, 20);는 sum(int a, int b) 메소드를 실행하고, hap.sum(100, 200, 300);는 sum(int a, int b, int c) 메소드를 실행하며, hap.sum(27.7, 60.6);는 sum(float a, float b) 메소드를 실행하므로 다음의 결과를 화면에 출력한다.

두 정수의 합은 30.0
세 정수의 합은 600.0
두 실수의 합은 88.3

• 정적 초기화자

• 클래스 변수가 배열과 같이 초기화해야 할 값이 많아 변수 선언문에서 초기화하기 힘든 경우는 정적 초기화자 static을 사용한다.

• 즉, static이라는 지정어를 사용하여 정의된 블록은 클래스가 로드될 때 한번만 실행되고, 클래스 변수만을 초기화할 수 있으며, 선언형식은 다음과 같다.

static {정적초기화 문}

설명 :

{와 }사이의 정적 초기화문은 클래스를 로드할 때 클래스 변수를 초기화하는 문이며, 정적 초기화문은 생성자보다도 먼저 실행된다.

StaticInitialization.java(27)

```
class StaticInitialization
{
   static int[] values = new int[10];
   static{
      for(int i=0; i<values.length; i++)
         values[i]=(int)(100.0*Math.random());
   }
   public void listvalues(){
      for(int i=0; i<values.length; i++)
         System.out.print(values[i]+" ");
   }
   public static void main(String[] args){
      StaticInitialization exam = new StaticInitialization();
      exam.listvalues();
   }
}
```

클래스 배열객체로서 초기화를 하지 않았음.

values 배열객체를 static 정적초기화자를 사용하여 초기화한다.(클래스가 로드될때 실행)

values 배열객체의 내용을 출력한다.

예제 설명 및 실행결과

초기화문에서 Math.random() 문은 Math 클래스에서 지원하는 random() 클래스 메소드를 호출하여 난수를 구하는 기능을 한다. 따라서 프로그램은 실행할 때마다 다르지만, 저자가 프로그램을 실행한 결과 다음과 같이 출력하였다.

40 97 88 63 58 48 84 5 32 67

• 내부 클래스(Inner Class)

• 하나의 클래스 내부에 선언된 클래스를 내부 클래스라 하고, 내부 클래스를 포함한 클래스를 외부 클래스라 한다.

• 내부 클래스는 외부 클래스에 포함된 클래스이기 때문에 외부 클래스에서 정의된 모든 멤버 변수 및 메소드를 접근 한정자에 관계없이 접근할 수 있으며, 형식은 다음과 같다.

```
class 클래스이름   // 외부 클래스
{
   ..................
   class 클래스이름   // 내부 클래스
   {
      ..................
   }
   ..................
}
```

• 내부 클래스의 특징

• 외부 클래스에서 내부 클래스의 멤버변수 및 메소드를 사용하기 위해서는 반드시 내부 클래스로부터 객체를 생성해서 접근해야 한다.

• 다른 클래스에서 내부 클래스를 사용하기 위해서는 외부 클래스를 통하여 접근해야 한다. 예를 들면, Inner 클래스는 Outer 클래스의 내부 클래스이고, Another라는 클래스에서 Inner 클래스를 접근하기 위해서는 외부 클래스인 Outer 클래스를 통하여 Inner 클래스의 객체를 다음과 같이 생성해야 한다.

```
Outer.Inner innerobj = new Outer.Inner();
```

• 내부 클래스는 외부 클래스에서 선언된 모든 멤버변수나 메소드를 접근 한정자에 관계없이 자신의 멤버변수나 메소드처럼 사용이 가능하다.

• 내부 클래스는 별도로 컴파일을 하지 않고, 외부 클래스를 컴파일할 때 자동으로 컴파일이 되어 다음과 같은 바이트 코드(확장자 명은 class) 이름을 가진다.

외부 클래스이름$내부 클래스이름.class

예를 들면, Inner1 및 Inner2 내부 클래스를 포함한 Outer 외부 클래스를 컴파일하면 다음과 같이 3개의 바이트 코드가 만들어진다.

Outer.class Outer$Inner1.class Outer$Inner2.class

ArithmeticTwo.java(28)

```
class ArithmeticTwo
{
  float operand1;
  float operand2;
  public ArithmeticTwo(float x, float y){
    operand1 = x;
    operand2 = y;
    Sum sum = new Sum();
    Subtract sub = new Subtract();
    System.out.println("합의 결과는 "+sum.calculate());
    System.out.println("뺄셈의 결과는 "+sub.calculate());
  }
  class Sum
  {
    public float calculate(){
      return operand1+operand2;
    }
  }
  class Subtract
  {
    public float calculate(){
      return operand1-operand2; // 외부 클래스의 객체변수를 객체 생성없이 사용
    }
  }
  public static void main(String[] args){
    ArithmeticTwo gen = new ArithmeticTwo(200, 100);
  }
}
```

2개의 내부 클래스를 포함하는 외부 클래스

내부 클래스의 메소드를 사용하기 위하여 객체를 생성한다.

내부 클래스(컴파일결과는 ArithmeticTwo$Sum.class)

외부 클래스의 객체변수를 객체 생성없이 사용한다.

내부 클래스(컴파일결과는 ArithmeticTwo$Subtract.class)

실행결과

합의 결과는 300

뺄셈의 결과는 100