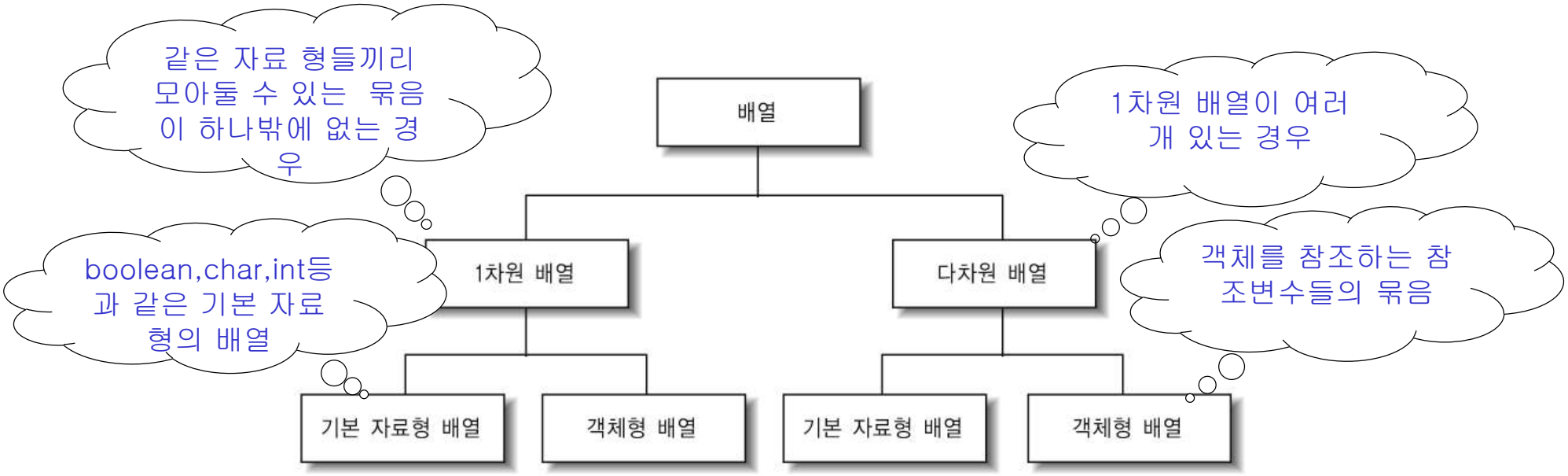
chapter 04.

배열과 클래스

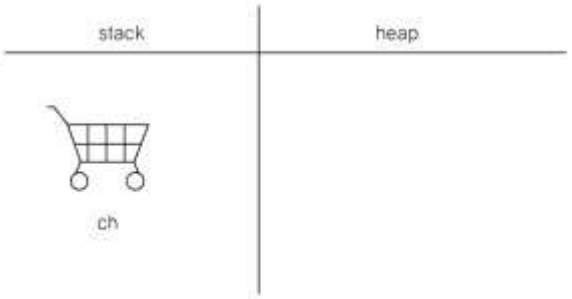
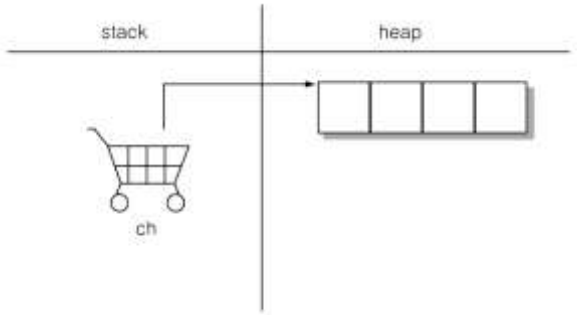
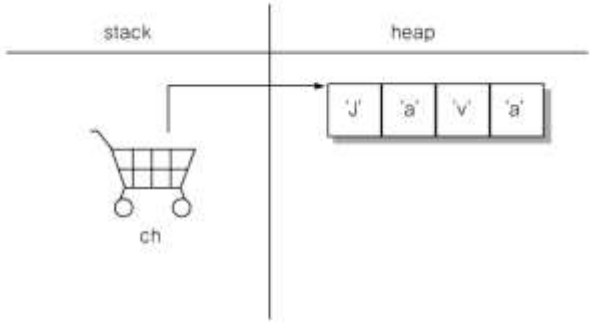
❍배열

- 배열은 같은 자료 형들끼리 모아두는 하나의 묶음이다.
- 자바에서 하나의 배열은 하나의 객체로 인식된다.
- 배열의 종류



[그림 4-55] 배열의 종류와 구분

❍ 배열의 단계적 작업

1) 배열 선언 char[] ch; 또는 char ch[];	2) 배열 생성 ch = new char[4];	3) 배열 초기화 ch[0]='J'; ch[1]='a'; ch[2]='v'; ch[3]='a';
 [그림 4-57] 배열 생성 1단계(선언)	 [그림 4-58] 배열 생성 2단계(생성)	 [그림 4-59] 배열 생성 3단계(초기화)

❑ 3) 배열3 (222p)

- **1차원 배열 예제**

[예제4-38] ArrayEx1.java

```
01  class ArrayEx1{
02   public static void main(String[] args){
03         char[] ch; //배열 선언
04         ch = new char[4]; //배열 생성
05
06         //배열 초기화
07         ch[0] = 'J';
08         ch[1] = 'a';
09         ch[2] = 'v';
10         ch[3] = 'a';
11
12         //배열 내용 출력
13         for(int i = 0 ; i < ch.length ; i++)
14                   System.out.println("ch["+i+"]:"+ch[i]);
15   }
16  }
```

실행결과

```
---------- Java Run ----------
ch[0]:J
ch[1]:a
ch[2]:v
ch[3]:a
Normal Termination
출력 완료 (0초 경과).
```

❍ 객체형 배열

- 객체형 배열은 객체를 가리킬 수 있는 참조 값(주소)들의 묶음이다. 실제 값이 저장된 기본 자료 형과는 다르게 객체의 reference들의 집합인 것이다.

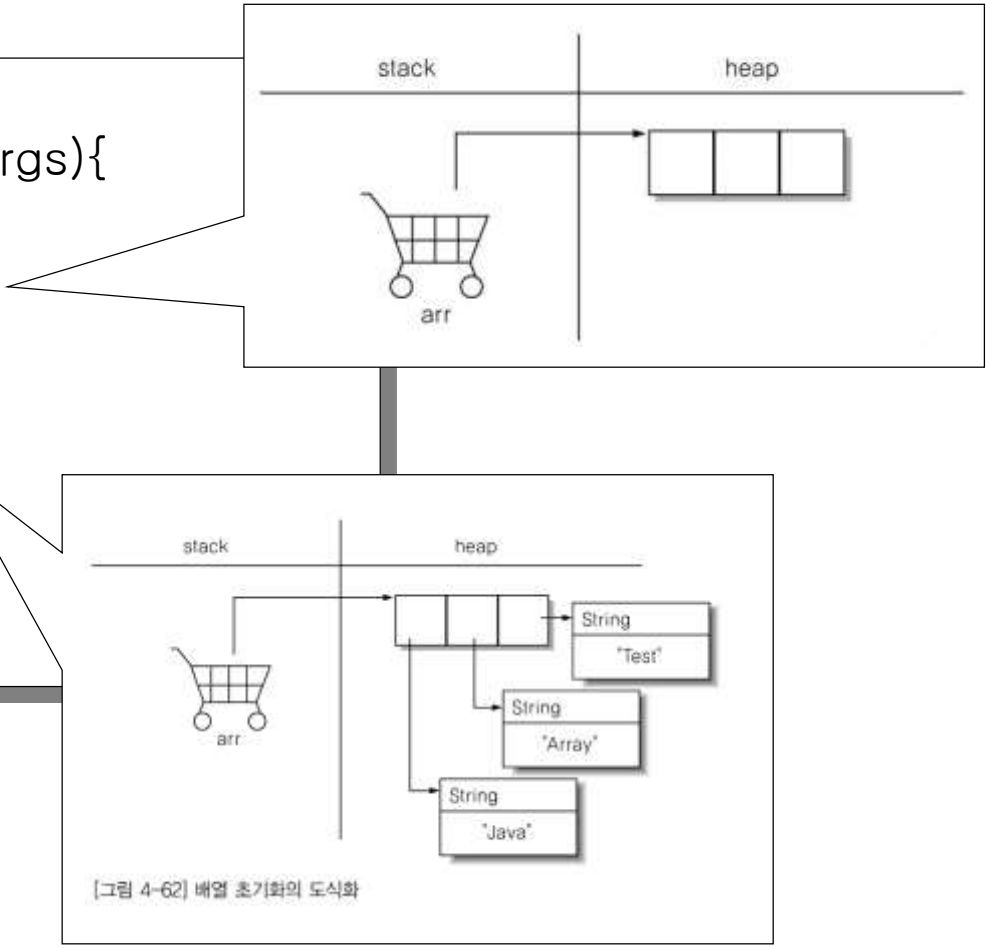
- 객체형 배열은 집집마다 우편함을 한곳에 모아둔 것과 같다. 각 우편함들은 나름대로 특정한 가정이라는 객체(Object)의 주소를 대신하는 것을 의미하며 이들의 묶음(집합)이 곧 reference배열 또는 객체형 배열이라 한다

❑ 3) 배열5 (223p)

- 객체형 배열 예제

[예제4-39] ObjArrayEx1.java

```
01 class ObjArrayEx1 {
02   public static void main(String[] args){
03         String[] arr;
04         arr = new String[3];
05
06         arr[0] = "Java ";
07         arr[1] = "Array ";
08         arr[2] = "Test";
09   }
10 }
```



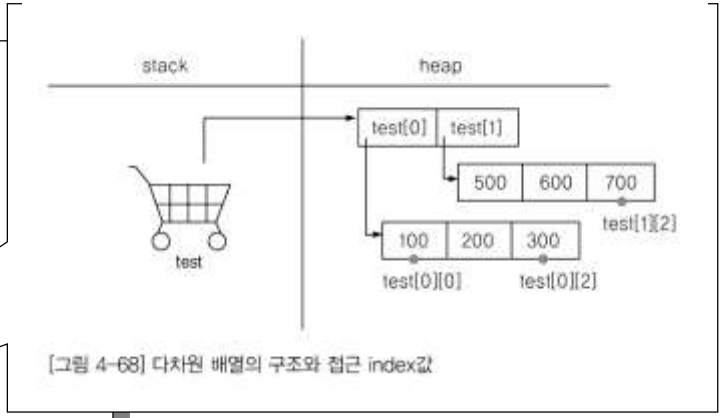
[그림 4-62] 배열 초기화의 도식화

❍다차원 배열

- 1차원 배열이 여러 개 모인 것을 다차원 배열이라 한다.

[예제4-45] ObjArrayEx4.java

```
01 class ObjArrayEx4 {
02   public static void main(String[] args){
03         int[][] test; // 다차원 배열 선언
04         test = new int[2][3];
05         test[0][0] = 100;
06         test[0][1] = 200;
07         test[0][2] = 300;
08         //---------------- 1행 초기화 끝
09         test[1][0] = 500;
10         test[1][1] = 600;
11         test[1][2] = 700;
12         //---------------- 2행 초기화 끝
13   }
14 }
```



[그림 4-68] 다차원 배열의 구조와 접근 index값

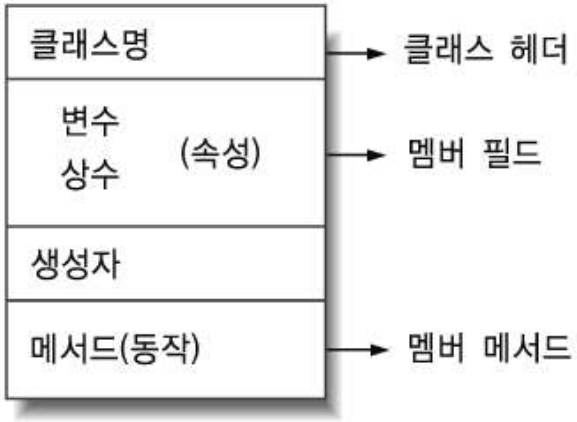
❍ 클래스란?

한 마디로 건물을 지을 때 제일 먼저 필요로 하는 설계도면과 같다. 현관의 위치와 복도의 크기 등의 속성들을 정확하게 Design된 설계도면을 가지고 건물을 짓게 된다. 완성된 건물을 자바에서는 곧 객체(Object)라 한다.
이렇게 건물이 지어지면 그 건물의 위치를 가리키는 주소(reference)가 있게 마련이고 그 주소를 통해 건물로 세금통지서 또는 각종 배달 서비스 즉, 연락(요청)을 취할 수 있다.

[그림 4-1] 클래스의 재사용

❍ 클래스의 구조와 정의

- 자바 프로그램은 클래스들이 모여서 이루어진다고 생각해도 무리는 아니다. 그래서 자바를 두고 “완전한 객체 지향적 언어”라는 호평이 생긴 것이다. 이런 클래스들을 많이 이해하고 생성하여 얻어지는 객체(Object)를 사용하는 것이 자바이므로 객체(Object)를 생성하기 위해서는 먼저 클래스의 구조를 알고 작성 할 수 있어야 한다.



[그림 4-2] 클래스의 구조

❍ 클래스 헤더

- 클래스 헤더는 클래스를 선언하는 부분을 의미한다. 여기에는 class라는 예약어를 중심으로 오른쪽은 클래스 명이며 왼쪽은 접근 제한(Access Control/Access Modifier)과 클래스의 형태 및 클래스 종류를 나타내기도 한다.

```
[접근제한] [클래스 종류] class 클래스명
```

:: 이 외에도 더 많은 설명이 있어야 하겠지만 처음 접하는 관점에서 우선 여기까지만 조금씩 정리하면서 넘어 가자!

- [접근제한]
  - 뒤에서 자세히 공부하게 되지만 그래도 의미 정도는 알아보자. 접근 제한은 말 그대로 현재 클래스를 접근하여 생성하고 사용하는데 있어 제한을 두겠다는 의미에서 정의하는 것이다. 클래스에서 쓰이는 접근 제한은 public을 정의하는 방법과 아예 정의하지 않는 방법 두 가지가 있다.
- [클래스종류]
  - 이는 최종(final)클래스 또는 추상(abstract)클래스와 같은 클래스 종류를 의미하며 현재 클래스가 어떤 클래스인지를 알리는 수식어의 일종이다. 이 부분이 생략되게 되면 일반 클래스임을 의미 하게 된다.
- [클래스명]
  - 클래스의 이름을 의미하며 여기에는 몇 가지 약속이 있다 했다. 이는 제3장 자바 기본문법의 식별자에서 이미 공부한 바가 있으니 기억이 나지 않으면 지금 바로 참조하기 바란다.

```
01      class Ex1{
02
03      }
```

❍ 멤버 필드

- 변수와 상수(속성) 즉, 데이터라고도 하는데 이것은 객체가 만들어질 때에 그 객체의 특징적인 속성을 담아두는 것이다. 여기서 필드의 형태가 static이냐? instance냐?에 따라 필드개념이 달라진다. 이는 멤버변수 부분과 현재 절의 static부분에서 다루도록 하겠다.

- [상수]
  - 상수라는 것은 고정된 값을 의미하며 프로그램이 종료 될 때가지 절대로 변하지 않는 값(속성)인 것이다. 제5장의 final부분에서 자세히 다루고 있다.!
- [변수]
  - 변수는 상수와는 반대로 프로그램이 종료 될 동안 값이 변경될 수 있는 값(속성)을 의미한다.

```
01      class Ex1{
02          int data;
03      }
```

❍ 멤버 메서드

- 메서드는 특정한 일을 수행하는 행위, 다시 말해 동작을 의미하는 것이다.
- 멤버필드 들의 값을 가지고 작업을 수행할 수도 있으며 메서드도 static 메서드(클래스 메서드)와 instance메서드 라는 2종류가 있다. 간단히 설명하자면 static메서드(클래스 메서드)는 메서드를 가지는 객체를 생성하지 않아도 사용할 수 있지만 instance메서드는 객체를 생성해야만 사용이 가능한 것이다. 현재 절의 마지막 부분인 static을 공부할 때 자세히 알아보도록 하고 여기서는 클래스의 구조를 알아보는 것이 목적이므로 클래스의 구조에 중심을 맞춰 공부하도록 하자!

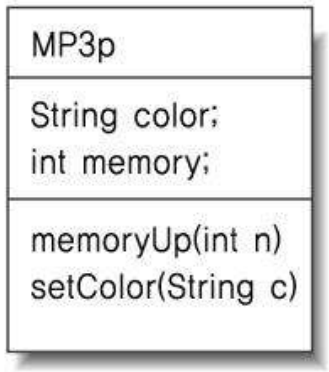
```
01      class Ex1{
02          int data;
03          public void setData(int n){
04              data = n;
05          }
06      }
```

❍ 클래스 정의

- 우리가 얘기하는 클래스란? 초등학생 들이 어떤 사물을 인식하고 그것을 그림으로 표현하는 것처럼 프로그래머들은 그 사물을 자바라는 프로그램 언어에 도입하여 추상적으로 사물의 속성과 움직임을 표현한 것이다. 그럼 다음 [조건]을 보고 우리가 일상 생활에 있는 사물 중 MP3 Player를 클래스로 간단히 정의해 보도록 하자!

[조건]

우선 클래스명은 MP3p라 정하고 속성부분은 색상을 기억하는 color와 메모리 용량을 기억하는 memory로 정하자! 동작부분에서는 memory용량을 upgrade해 주는 memoryUp이라는 동작과 color를 설정하는 setColor라는 동작을 정의해 보자!




```
01      class MP3p{
02              String color;
03              int memory;
04              public void memoryUp(int n){
05                      memory += n;
06              }
07              public void setColor(String c){
08                      color = c;
09              }
10      }
```

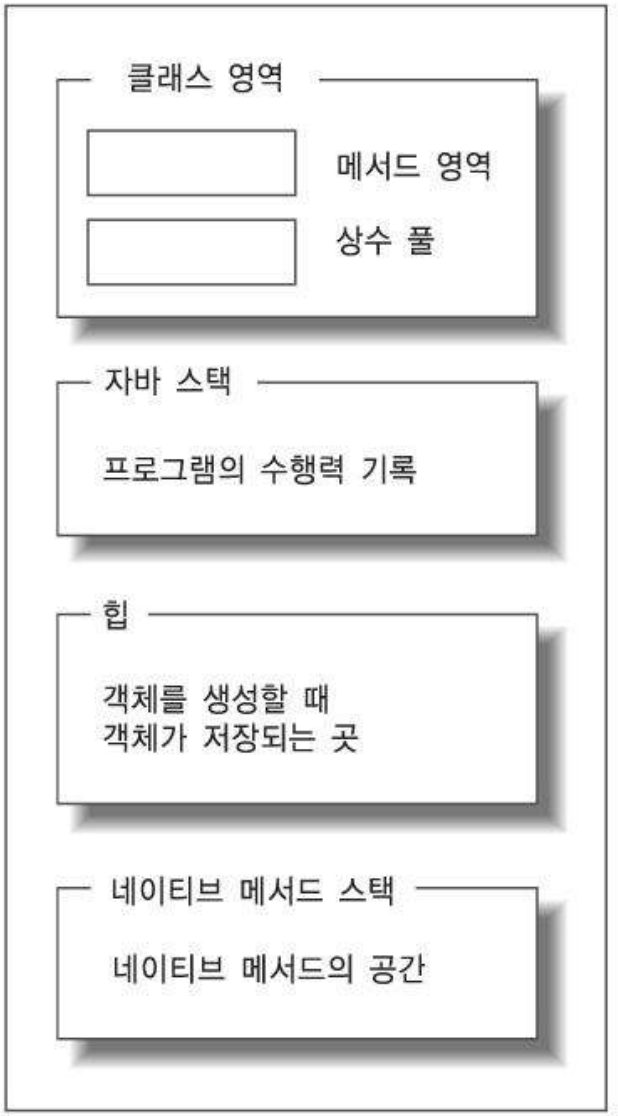
※ 저장 시에는
반드시 클래스 명과 동일!

MP3p.java

❍ 간단히 알아보는 메모리 구조

- 앞서 제1장의 29p를 참조해 보면 메모리 영역은 여러 가지로 나눌 수 있다. static영역은 뒤에서 공부하므로 여기서는 stack과 heap에 관해서만 얘기하도록 하자.

- stack은 offset값만으로 바로 참조할 수 있는 변수와 같이 가벼운 것을 저장하는 곳이며 heap은 사실 내부에 참조영역(registry)을 따로 가지므로 객체와 같은 무거운 것들을 저장하는 공간이라 할 수 있다.

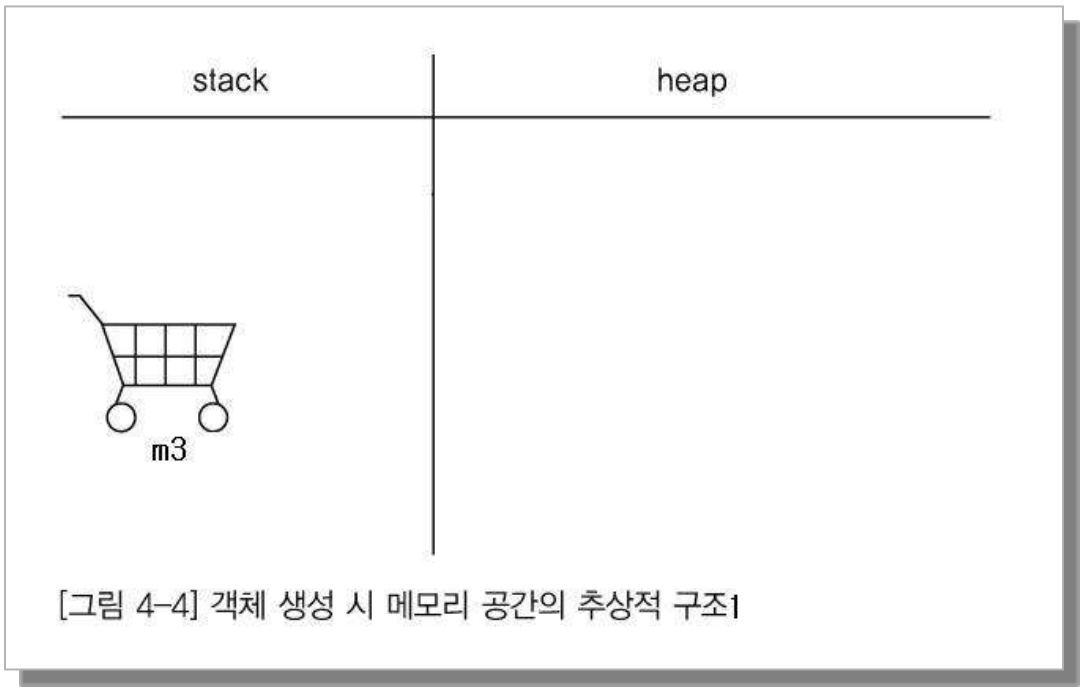


❍ 객체 생성과 멤버 접근법

- 정수형 또는 실수형 그리고 문자형등과 같은 자료형을 '기본 자료형'이라 하며 자바 내부에서 제공되는 클래스 또는 MP3p클래스처럼 프로그래머에 의해 만들어진 사용자 정의 클래스 등을 자료형으로 하는 '참조(객체) 자료형'이 있다는 것을 제3장에서 이미 공부한 적이 있다. 이런 참조 자료형을 가지는 변수(reference) 다시 말해서 객체를 생성하고 사용하는 법을 알아 보도록 하자.

- 1) 객체 선언

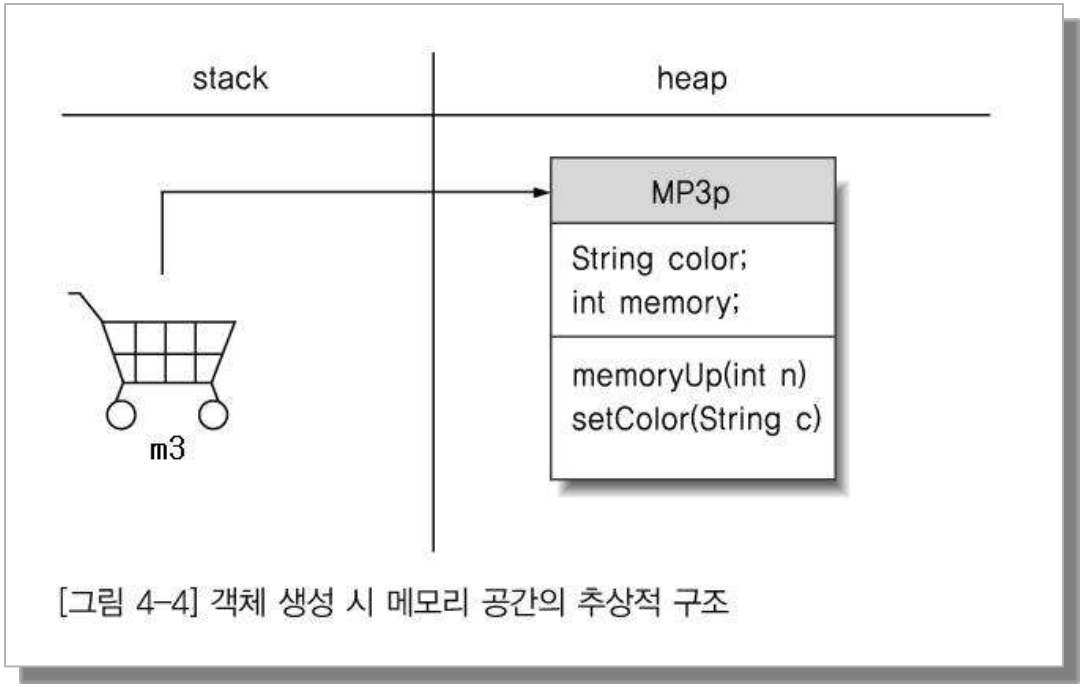
```
MP3p m3;
```



[그림 4-4] 객체 생성 시 메모리 공간의 추상적 구조1

- 1) 객체 생성

```
m3 = new MP3p();
```



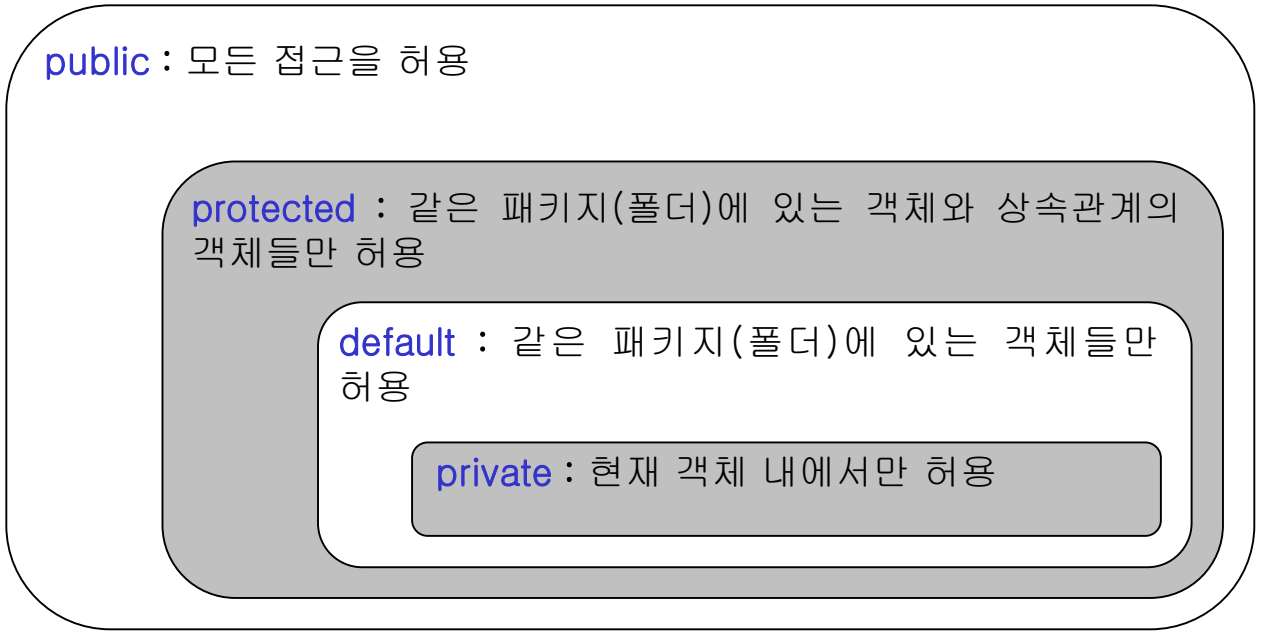
[그림 4-4] 객체 생성 시 메모리 공간의 추상적 구조

- new라는 연산자를 통해 무조건 메모리상에 공간을 할당 받고 MP3p클래스의 생성자로 인하여 객체를 생성한 후 생성된 객체를 참조할 수 있는 reference를 m3에 담아준다.(이것이 객체를 구분할 수 있는 주소개념이다.) 결국 m3를 통해 MP3p객체에게 서비스 요청을 하게 되는 것이다.

- 생성자에 관해서는 뒤에서 따로 공부하는 부분이 있으니 여기서는 객체를 생성할 때는 생성자라는 것에 의해 만들어지는 것이다. 라는 것만 알고 가자!

- 접근 제한자

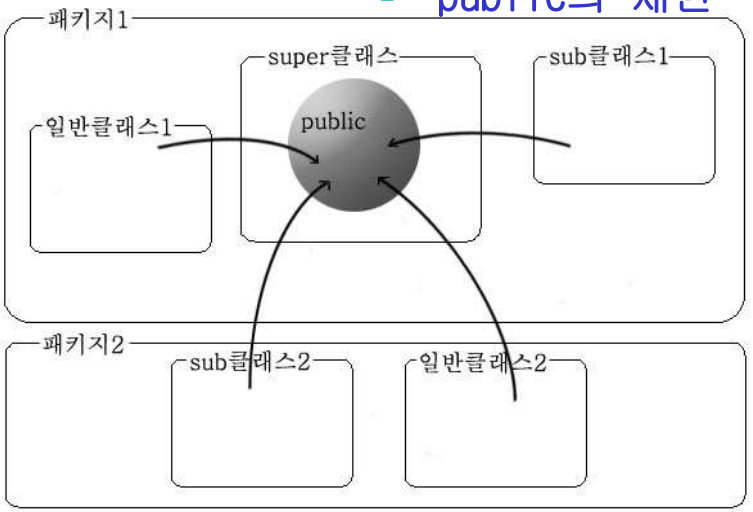
  멤버들은 객체 자신들만의 속성이자 특징이므로 대외적으로 공개되는 것이 결코 좋은 것은 아니다. 그런 이유로 프로그래머가 객체의 멤버들에게 접근 제한을 걸 수가 있는데 자바에서는 이를 접근 제한자라 한다.



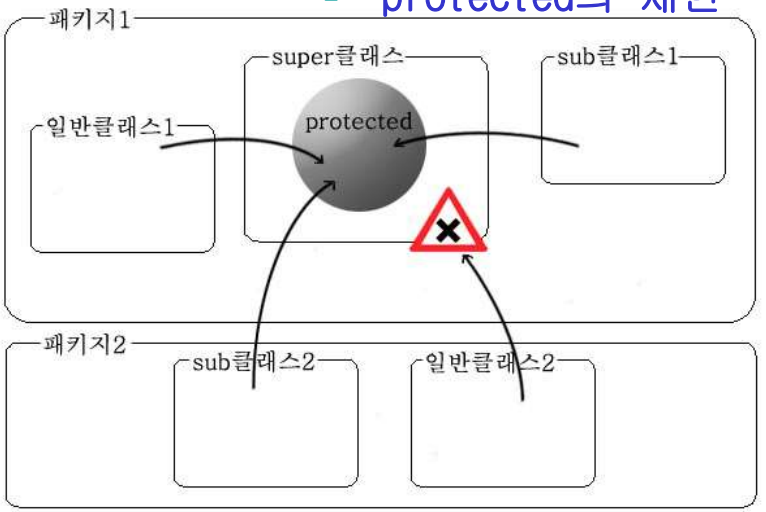
:: 같은 패키지라는 말의 뜻을 간단히 설명 하자면, 같은 폴더에 있는 객체를 의미한다고 생각하면 된다. 6장에서 자세히 알아보도록 하자!

❑ 1) 클래스와 객체의 개념12 (137p)

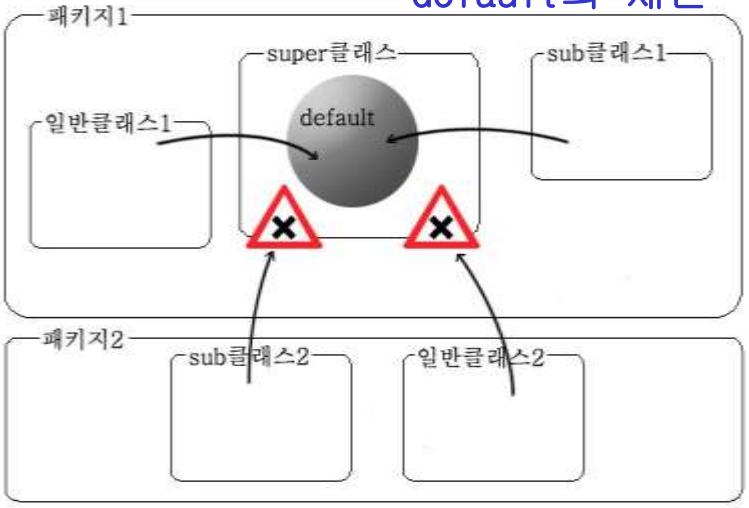
- public의 제한



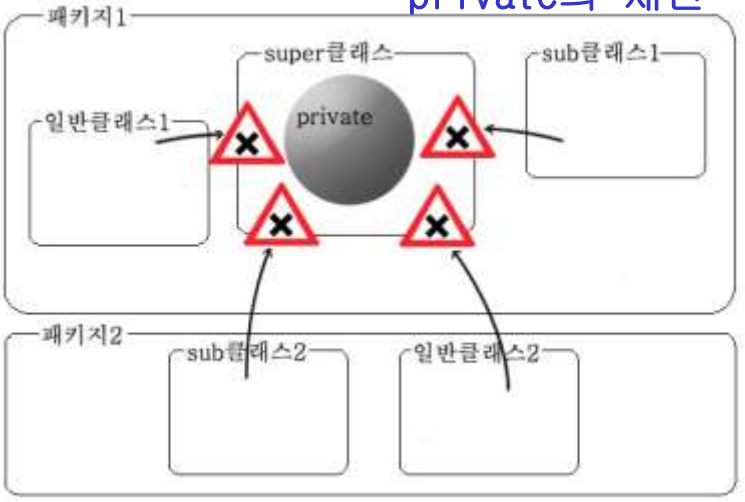
- protected의 제한



- default의 제한



- private의 제한



❍ 멤버 접근법

- 멤버 접근법에 대해서는 앞서 확인해본 것과 같은 m3와 같은 객체의 참조(reference)를 통해 (.)를 사용하여 해당 객체의 메서드나 변수 즉 멤버들을 접근 할 수 있다. (.)는 dot연산자라고도 하며 사용법은 [객체참조.멤버명] 형식으로 사용된다.

```
m3.memoryUp(768);
```

- 여기서 쓰인 m3가 바로 MP3p클래스가 생성되어 그 객체의 참조(reference)를 가지는 변수이다. 즉 생성된 MP3p객체를 m3가 참조하게 되는 것이고 이것을 통해 인자(Arguments)값으로 768을 전달하며 memoryUp()함수를 호출하여 하나의 멤버변수에 누적 시키는 동작을 하게 되는 것이다.

❍ 캡슐화

- 객체에 대한 수정 작업으로 인해 객체 활용도에 문제가 발생한다면 제품 하나에 대한 upgrade가 상당히 부담스러울 것이다. 자바에서는 각 기능을 하나의 모듈(부품)처럼 활용하여 객체간의 이식성이 높고 독립적인 면을 유지하는 장점이 있다.

  전기 압력밥솥이 어떤 원리와 어떤 과정으로 밥을 지어내는지는 알 필요 없이 그냥 사용법만 익혀서 사용한다. 고장이 나서 A/S를 받을 때마다 사용법이 바뀐다면 좋은 회사 또는 좋은 제품(객체)이라 할 수 없다.
  사용법이 바뀌지 않는 이유는 모든 기능이 모듈화 되어 있기 때문이다.

  캡슐화란? 관련이 있는 데이터 그리고 동작들을 하나로 묶어 요약하고 사용자에게는 내부적인 접근을 허용하지 않는 대신에 사용의 편의성을 제공해 주는 것이다.

❍ 멤버변수

- instance변수 (개인이 소유하고 있는 물컵)
  - 객체가 생성될 때 각 객체들마다 따로 따로 생성 되어 고유의 값을 받아 각 객체의 속성으로 자리 잡는 변수가 바로 instance변수이다.

```
1    class MP3p{
2            String color;
3            int memory;
```

- static변수 (약수터에 있는 물 바가지)
  - 여러 개의 객체가 생성될 때 단 하나만 생성 되며 모든 객체들이 공유하는 개념으로 사용되는 변수가 static변수이다.

```
1    class MP3p{
2            String color;
3            int memory;
4            static String maker;
```

❍ 메서드

메서드(멤버함수)란? 객체가 할 수 있는 동작을 정의하는 것이며 메서드 또한 instance메서드와 static메서드로 나뉘어 진다. 의미는 앞의 내용과 같다.

▪ 메서드의 구성

```
[접근제한] [반환형] [메서드명](자료형 인자1, 자료형 인자2,…){

        수행문1;
        수행문2;
        …;
}
```

- 접근제한

  앞서 공부한 부분이며 다시 한번 복습하자면 자바에서 객체나 멤버들에 대한 접근을 제한하는 방법을 의미한다. 여기에는 public, protected, default, private의 종류로 나눌 수 있다.

- 반환형

  메서드(멤버함수)에서 해야 할 일들을 모두 끝내고 마지막으로 메서드(멤버함수) 자신을 불러준 곳으로 반환하는 값의 자료형을 의미하는 것이다. 만약 반환 값이 없다면 void라는 예약어로 대처해야 한다. 즉 생략은 불가능하다는 뜻이다.

- 메서드명

  사용자 정의 이름(User defined name)이며 제3장의 식별자에 대한 규칙을 참조하기 바란다.

- 인자

  Arguments라고도 하며 이것은 메서드(멤버함수)를 호출할 때 필요에 따라 특정 값을 제공해주기 위해 미리 선언하는 것이며 메서드(멤버함수)를 호출할 시에 반드시 인자의 자료형과 수가 일치해야 한다.

- 수행문

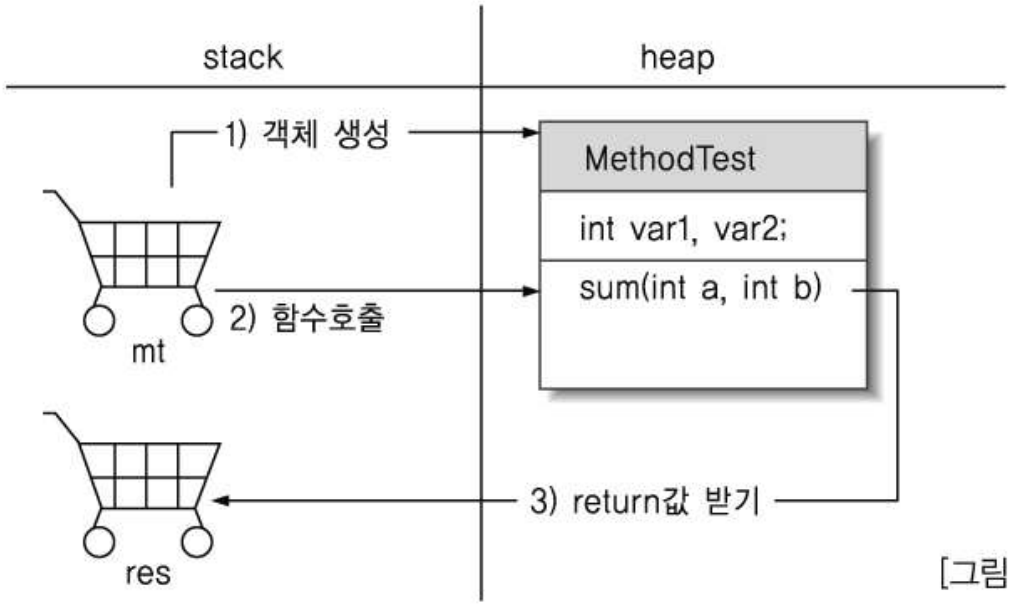
  식 수행 및 제어문 또는 실행문 등을 의미한다.

[예제4-1]MethodEx.java

```
01  class MethodEx {
02
03      int var1,var2;  // 멤버 변수들
04
05      public int sum(int a, int b){  // 메서드(멤버 함수)
06              return a+b;
07      }
08      public static void main(String[] args){
09              MethodEx me = new MethodEx();
10              int res = me.sum(1000, -10);
11              System.out.println("res="+res);
12      }
13  }
```

실행결과

```
---------- Java Run ----------
res=990
Normal Termination
출력 완료 (0초 경과).
```



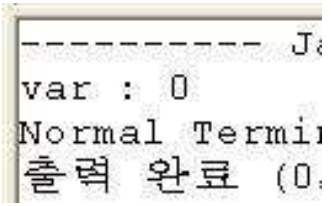
[그림 4-13] 객체 생성과 메서드(멤버함수) 사용

:: 위 예제에서 초보자들이 궁금해 하는 것 중의 하나가
우선 "프로그램의 시작이 어디인가?"인데 그것은 바로 8행부터이다.

[예제4-2]MethodEx2.java

```
01  class MethodEx2{
02
03      int var;
04
05      public void setInt(int var){
06          var = var;
07      }
08
09      public int getInt(){
10          return var;
11      }
12
13      public static void main(String[] args){
14          MethodEx2 me2 = new MethodEx2();
15          me2.setInt(1000);
16          System.out.println("var : "+ me2.getInt());
17      }
18  }
```

실행결과



1000이 출력되지 않는 이유가 무엇일까? 문제는 setInt(int var)라는 메서드(멤버함수)에 있으며 자바에서는 특정 영역의 우선권은 *멤버변수가 아닌 지역변수*가 가지게 되어 있다.

인자가 정의되면서 var이라는 지역변수가 선언되었다. 지역 변수란 현재 영역(함수)을 벗어나면 소멸되는 메모리 공간을 의미한다.

```
05  public void setInt(int var){
06        var = var;
07  }
```

지역변수의 이름이 멤버변수와 같을 경우에는 지역변수가 우선권을 가지므로 여기서는 멤버변수에 값 대입이 아닌 지역변수 자신에게 자신의 값을 대입한 것이다.

❍인자 전달 방식

- 값 호출 (Call by value)
  - 이는 앞선 예제 MethodEx2.java와 같이 메서드를 호출 시 기본 자료형의 값을 인자로 전달하는 방식을 의미한다.

- 참조 호출 (Call by reference)
  - 메서드 호출 시 전달하려는 인자를 참조(객체) 자료형을 사용할 경우를 의미한다. 여기에는 기본 자료형이 아닌 일반 객체 또는 배열들이 여기에 속한다.

- Varargs(Variable Arguments)
  - JDK5.0에서 새롭게 추가된 기능이며 이는 메서드 정의 시 통일된 인자의 자료형에 '…'라고 명시하므로 이를 통해 메서드를 수행하는데 필요한 인자의 수를 유연하게 구현할 수 있다.
    (내부적으로 배열화 작업을 자동적으로 해 주기 때문!)

▪ 값 호출 (Call by value)에 대한 예제

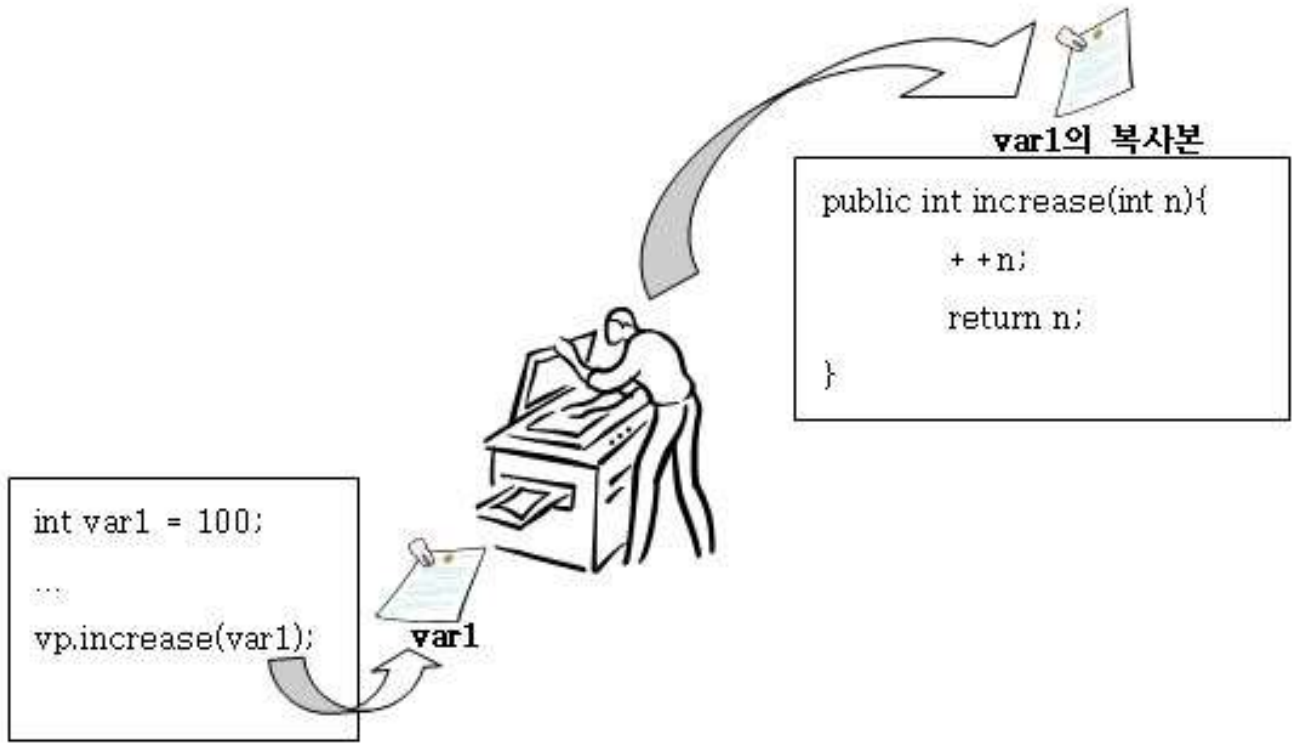
[예제4-3]ValueParameter.java

```
01  class ValueParameter{
02
03          public int increase(int n){
04                  ++n;
05                  return n;
06          }
07          public static void main(String[] args){
08                  int var1 = 100;
09                  ValueParameter vp = new ValueParameter();
10                  int var2 = vp.increase(var1);
11                  System.out.println("var1 : "+ var1 + ", var2 : " + var2);
12          }
13  }
```

실행결과

```
---------- Java Run ----------
var1 : 100, var2 : 101
Normal Termination
출력 완료 (0초 경과).
```

- 결과 내용을 보게 되면 인자로 전달된 int var1(실인자)의 값은 변함이 없음을 알 수 있다. 그러므로 int var1(실인자)의 값이 다음 그림과 같이 복사본이라고 할 수 있는 int n(형식인자)에 대입되고 그것을 증가 시키면 int var1(실인자)는 영향을 전혀 받지 않게 된다. 즉 int n(형식인자)만 증가하게 되는 것이다.



▪ 참조 호출 (Call by reference)에 대한 예제

[예제4-4]ReferenceParameter.java

```
01  class ReferenceParameter{
02
03          public void increase(int[] n){
04                  for(int i = 0 ; i < n.length ; i++)
05                          n[i]++;
06          }
07          public static void main(String[] args){
08                  int[] ref1 = {100,800,1000};
09                  ReferenceParameter rp = new ReferenceParameter();
10                  rp.increase(ref1);
11
12                  for(int i = 0 ; i < ref1.length ; i++)
13                          System.out.println("ref1["+i+"] : "+ ref1[i]);
14          }
15  }
```

실행결과

```
---------- Java Run ----------
ref1[0] : 101
ref1[1] : 801
ref1[2] : 1001
Normal Termination
출력 완료 (0초 경과).
```

- Varargs를 제외하고 자바에서 얘기하는 인자 전달방식에 있어서는 모두가 전달하는 인자가 복사되어 전달된다.
  (Call by value)에서는 값을 복사하여 전달하였으므로 호출 시의 실인자는 별도의 값으로 인식되어 영향을 받지 않으며
  (Call by reference)에서는 reference(주소)가 복사되어 전달 되었으므로 하나의 객체를 참조하는 변수가 2개가 되어 어느 한 곳에서 수정을 하게 되면 같은 객체를 참조하는 다른 쪽에서도 영향을 받게 된다.

▪ Varargs(Variable Arguments)에 대한 예제

[예제4-5] VarTest.java

```
01  class VarTest{
02
03          public void argTest(String ... n){
04                  for(int i = 0 ; i < n.length ; i++)
05                          System.out.println("n["+i+"]:"+n[i]);
06                  System.out.println("--------------------------");
07          }
08          public static void main(String[] args){
09                  VarTest vt = new VarTest();
10                  vt.argTest("Varargs", "Test");
11
12                  vt.argTest("100", "600", "900", "1000");
13          }
14  }
```

실행결과

```
---------- Java Run ----------
n[0]:Varargs
n[1]:Test
--------------------------
n[0]:100
n[1]:600
n[2]:900
n[3]:1000
--------------------------
Normal Termination
출력 완료 (0초 경과).
```

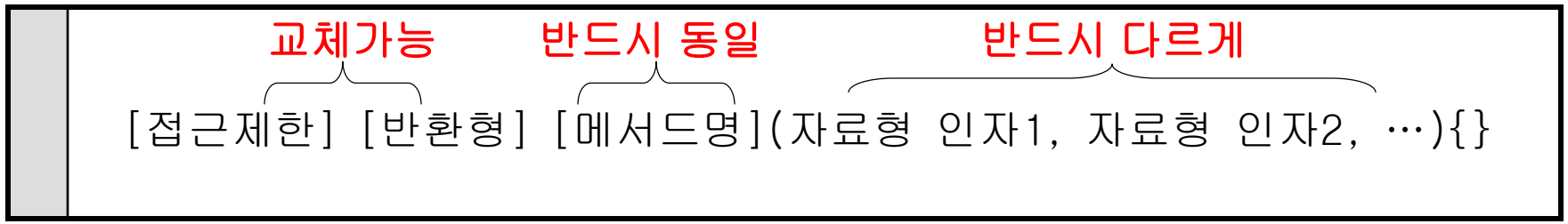
- Varargs의 장점은 앞으로 배우게 되는 '메서드 오버로딩'과 연관이 있겠지만 이에 대한 설명은 간단히 해보자!

  JDK5.0 이전의 버전에서는 특정 메서드를 정의할 때 인자(argument)의 타입과 수를 정해 두고 호출 시 전달되는 인자(argument)의 수가 일치하지 않을 경우에는 메서드를 호출할 수가 없었다.
  이런 문제로 인해 많은 개발자들이 메서드의 이름은 같지만 인자의 수가 다른 여러 개의 메서드를 정의하는 메서드 Overloading법 또는 메서드 정의 시에 배열객체를 인자로 지정하므로 이 문제를 해결 하였지만 매번 호출 시의 배열화 작업이 매우 불편하다.

  이런 부분이 앞의 예제 [VarTest.java]의 Varargs기능으로 해소 되었다.
  앞으로 Generics, 개선된 루프등과 같은 JDK5.0에서 새롭게 변화된 점들을 배워가면서 지금의 Variable Arguments부분을 같이 병행하게 되면 보다 심플한 내용을 보게 될 것이다.

❍메서드 오버로딩

- Overloading(중복정의)이라는 것은 하나의 클래스 내에서 같은 이름을 가지는 메서드(멤버함수)가 여러 개 정의되는 것을 말한다. 이것은 컴파일 시 컴파일러에 의해 각 메서들이 구별되며 기준은 인자가 된다



- 이런 Overloading기법이 필요한 이유는 같은 목적으로 비슷한 동작을 수행하는 메서드(멤버함수)들을 모아 이름을 같게 하면 프로그래머로 하여금 다양한 메서드(멤버함수)들을 같은 이름으로 일관된 작업을 할 수 있다는 편리함이 있다. 그럼 본 교재(153페이지)의 다음 예제들을 살펴보자!

- 비 오버로딩 예제

[예제4-6] OverloadingTest1.java

```
01  class OverloadingTest1{
02
03          public void intLength(int a){
04                  String s = String.valueOf(a);
05                  System.out.println("입력한 값의 길이 : "+s.length());
06          }
07          public void floatLength(float f){
08                  String s = String.valueOf(f);
09                  System.out.println("입력한 값의 길이 : "+s.length());
10          }
11          public void stringLength(String str){
12                  System.out.println("입력한 값의 길이 : "+str.length());
13          }
14          public static void main(String[] args) {
15                  OverloadingTest1 ot1 = new OverloadingTest1();
16
17                  ot1.intLength(1000);
18                  ot1.floatLength(3.14f);
19                  ot1.stringLength("10000");
20          }
21  }
```

- 오버로딩 예제

[예제4-7] OverloadingTest2.java

```
01  class OverloadingTest2{
02
03          public void getLength(int n){
04                  String s = String.valueOf(n);
05                  getLength(s);
06          }
07          void getLength(float n){
08                  String s = String.valueOf(n);
09                  getLength(s);
10          }
11          private int getLength(String str){
12                  System.out.println("입력한 값의 길이 : "+str.length());
13                  return 0;
14          }
15          public static void main(String[] args) {
16                  OverloadingTest2 ot2 = new OverloadingTest2();
17
18                  ot2.getLength(1000);
19                  ot2.getLength(3.14f);
20                  ot2.getLength("10000");
21          }
22  }
```

앞의 예제[OverloadingTest2.java]의 7번과 11번 행에서 알 수 있듯이 Overloading에 있어 [접근제한]과 [반환형]그리고 인자의 이름은 아무런 영향력을 미치지 못한다.

메서드명이 같은 여러 개의 메서드를 구별하는 방법은 메서드에 사용되는 인자의 자료형 또는 인자의 수 아니면 인자들의 배치가 서로 달라야 각 메서드들을 구별할 수 있다는 것을 명심하자!.

❍ 생성자

- 생성자라는 것은 메모리상에 객체가 생성될 때 자동적으로 단 한번 호출되어 객체의 구조를 인식하게 하고 생성되는 객체의 멤버 변수들을 초기화 하는데 목적을 둔 것을 말한다.
- 생성자의 특징

  1) return Type이 전혀 정의되지 않는다.
  2) 생성자의 이름이 클래스 명과 같아야 한다.

- 생성자의 구성

```
[접근제한] [생성자명](자료형 인자1, 자료형 인자2,…){

    수행문1;
    수행문2;
    …;
}
```

- 생성자의 필요성

  특정 클래스가 생성될 때마다 가지는 instance변수가 있어 항상 값을 설정해야 하는 번거로움이 있다면 생성자를 이용하여 1차적으로 초기화 작업을 거치면서 객체를 생성된다면 좀 더 편리할 것이다.

```
1.  class MyClass1{
2.          private String name;
3.          public MyClass1(String n){
4.                  name = n;
5.          }
6.          public void setName(String n){
7.                  name = n;
8.          }
9.  }
```

- 생성자 접근제한의 의미

  생성자의 접근 제한에도 앞서 설명한(135p) 접근 제한자(Access Modifiers)를 똑같이 사용한다. public과 protected 그리고 default와 private의 의미적 차이는 없으며 클래스의 접근 제한과 생성자의 접근 제한은 별도의 제한력을 가진다는 것이 중요하다.

- 생성자 오버로딩

  생성자의 Overloading은 객체를 생성할 수 있는 방법의 수를 제공하는 것과 같으며 앞서 배운 메서드 오버로딩법과 다를 것이 없어 각 생성자의 구분 또한 인자로 구별함을 잊지 말자!

❑ 1) 클래스와 객체의 개념34 (159p)

[예제4-9] MyClass2.java

```
01  class MyClass2{
02
03          private String name;
04          private int age;
05          public MyClass2(){
06                      name = “www.sist.co.kr”;
07          }
08          public MyClass2(String n){
09                      name = n;
10          }
11          public MyClass2(int a, String n){
12                      age = a;
13                      name = n;
14          }
15          public MyClass2(String n, int a){
16                      age = a;
17                      name = n;
18          }
19          public String getName(){
20                      return name;
21          }
22          public int getAge(){
23                      return age;
24          }
25  }
```

[예제4-10] MyClass2Test.java

```
01  class MyClass2Test{
02
03    public static void main(String[] args)
04    {
05          MyClass2 mc1 =  new MyClass2();
06          MyClass2 mc2 = new MyClass2("아라치");
07          MyClass2 mc3 =
                        new MyClass2("마루치",46);
08          MyClass2 mc4 =
                        new MyClass2(23,"오자바");
09
10          System.out.println(mc1.getName()+","+
                        mc1.getAge());
11          System.out.println(mc2.getName()+","+
                        mc2.getAge());
12          System.out.println(mc3.getName()+","+
                        mc3.getAge());
13          System.out.println(mc4.getName()+","+
                        mc4.getAge());
14    }
15  }
```

this와 this()

- this : 객체 내부에서 객체 자신을 칭하고 싶을 때나 아니면 지역변수와 멤버변수를 구별해야 할 때도 있을 것이다. 이럴 때 사용하는 객체 자신을 가리킬 수 있는 유일한 reference! 이것이 바로 this이다.

- this() : 이것은 현재 객체의 생성자를 의미하는 것이다. 주의 해야 할 점은 생성자의 첫 행에 정의해야 한다는 것이며 그렇지 않으면 Compile시 오류가 발생한다.

  다시 말하면 이 this()를 이용하여 한 클래스내의 특정 생성자에서 Overloading되어 있는 다른 생성자를 호출할 수 있는 것이다. 그렇게 함으로 해서 생성자에서 코딩 내용이 중복되는 부분을 막을 수 있다.

:: 165p의 예제 참조!!

static예약어

- static 예약어는 메서드(멤버함수)나 멤버변수에 정의 할 수 있으며 지역 변수나 클래스에게는 정의 할 수 없다.

  메서드(멤버함수)나 멤버변수에 static이라는 예약어를 정의하면 static메서 (클래스메서드)와 static변수(클래스변수)라고 불리게 된다. 이유는 멤버변수나 메서드(멤버함수)들은 해당 객체가 생성될 때 객체가 생성된 메모리 공간에 같이 존재 하게 되지만 static으로 선언된 *메서드(멤버함수)나 변수들은 static영역(메소드영역)이라는 곳에 유일하게 만들어 지면서 모든 객체(Object)들이 사용 할 수 있도록 공유개념을 가지기 때문*이다.

- static정의 형식

```
교체가능
[접근제한] static [반환형] 메서드명(자료형 인자1, 자료형 인자2, …){}

교체가능
[접근제한] static [자료형] 변수명;
```

:: *167p~ 174p까지의 예제 참조*

❍ String 클래스

- 문자열 객체
- 문자열 수정 불가능(불변적 특징)
- + 연산자를 이용하여 새로운 문자열 객체 생성

```
String str3 = str1 + str2;
```

- 주요 생성자

[표 4-1] String 클래스 생성 방법

생성법	예
암시적 객체 생성	String s1 = "www.sist.co.kr";
명시적 객체 생성	String s2 = new String("www.sist.co.kr");

- String클래스 예제

[예제4-20] StringEx1.java

```
01  class StringEx1{
02
03          public static void main(String[] args){
04              String s1 = "Twinkle";
05              String s2 = "Twinkle";
06              if(s1 == s2)
07                  System.out.println("s1과 s2는 같다. ");
08              else
09                  System.out.println("s1과 s2는 같지 않다. ");
10
11              String s3 = new String("Little Star");
12              String s4 = new String("Little Star");
13              if(s3 == s4)
14                  System.out.println("s3과 s4는 같다. ");
15              else
16                  System.out.println("s3과 s4는 같지 않다. ");
17          }
18  }
```

암시적 객체 생성

==연사자로 객체를 비교시에는 참조 값(주소) 비교

명시적 객체 생성

:: *객체비교에 있어서 184p의 예제를 참조!*

:: *String활용예제 186p의 예제 참조!*

실행결과

```
---------- Java Run ----------
s1과 s2는 같다.
s3과 s4는 같지 않다.
Normal Termination
출력 완료 (1초 경과).
```

StringBuffer 클래스

- 문자열 버퍼 객체
- 문자열 추가 변경 가능
- append() 메서드를 이용하여 문자(열) 추가

```
str1.append(str2);
```

- 주요 생성자

[표 4-4] StringBuffer 클래스의 주요 생성자

생성자	설명
StringBuffer( )	비어있는 StringBuffer 객체를 생성하고 초기값으로 문자 16자를 기억할 수 있는 용량(buffer의 길이)을 가진다.
StringBuffer(CharSequence seq)	현재 인자로 전달된 특정한 CharSequence와 같은 문자열을 포함한 StringBuffer 객체를 생성한다.
StringBuffer(int capacity)	현재 인자로 들어온 capacity이라는 값으로 새롭게 생성될 StringBuffer 객체의 용량(buffer의 길이)으로 초기화하여 생성한다.
StringBuffer(String str)	현재 인자로 전달된 문자열 str을 초기값으로 하여 StringBuffer 객체를 생성한다. 이때 용량(buffer의 길이)은 str의 길이+16이다.

- StringBuffer클래스 예제

[예제4-27] StringBufEx2.java

```
01  class StringBufEx2{
02
03    public static void main(String[] args){
04
05      StringBuffer sb1 = new StringBuffer("Sun-Ae");
06      StringBuffer sb2 = sb1.append(" & Kyung-Ju");
07
08      String msg = null;
09      if(sb1 == sb2)
10        msg = "sb1와 sb2는 같다.";
11      else
12        msg = "sb1와 sb2는 다르다.";
13      System.out.println(msg);
14      msg = sb1.toString();
15      System.out.println("sb1 : " + msg);
16      msg = sb2.toString();
17      System.out.println("sb2 : " + msg);
18    }
19  }
```

StringBuffer 객체 생성

문자열 추가

문자열이 추가되고 참조 값(주소)는 변함이 없다.

실행결과

```
---------- Java Run ----------
sb1와 sb2는 같다.
sb1 : Sun-Ae & Kyung-Ju
sb2 : Sun-Ae & Kyung-Ju
Normal Termination
출력 완료 (0초 경과).
```

StringTokenizer 클래스

- 문자열 분리 객체
- nextToken() 메서드를 이용하여 문자(열) 분리

```
String token = st1.nextToken();
```

- 주요 생성자

[표 4-6] StringTokenizer 클래스의 주요 생성자

생성자	설명
StringTokenizer(String str)	현재 인자로 전달된 String 객체인 str을 기본 구분문자인 white space, new line, tab 등의 구분문자로 하여 분할할 StringTokenizer 객체를 생성한다.
StringTokenizer(String str, String delim)	현재 인자로 전달된 String 객체인 str을 두 번째 인자인 delim으로 구분문자로 하여 분할할 StringTokenizer 객체를 생성한다.
StringTokenizer (String str, String delim, boolean returnDelims)	현재 인자로 전달된 String 객체인 str을 두 번째 인자인 delim으로 구분문자로 하여 분할할 StringTokenizer 객체가 생성될 때 세 번째 인자인 boolean형에 의해 delim 또한 token 자원으로 사용할 것인지를 결정하게 된다.

❑ 2) 기본 클래스 익히기6

- StringTokenizer클래스 예제

[예제4-29] StringTokenEx1.java

```
01  import java.util.StringTokenizer;
02  class StringTokenEx1{
03
04    StringTokenizer st;
05    public StringTokenEx1(String str){
06        System.out.println("str : " + str);
07        st = new StringTokenizer(str);
08    }
09    public StringTokenEx1(String str, String delim){
10        System.out.println("str : " + str);
11        st = new StringTokenizer(str, delim);
12    }
13
14    public void print(){
15        System.out.println("Token count : " + st.countTokens());
16        while(st.hasMoreTokens()){
17            String token = st.nextToken();
18            System.out.println(token);
19        }
20        System.out.println("-----------------------------");
21    }
```

```
22    public static void main(String[] args) {
23
24        StringTokenEx1 st1 =
              new StringTokenEx1("Happy day");
25        st1.print();
26
27        StringTokenEx1 st2 =
              new StringTokenEx1("2005/08/15","/");
28        st2.print();
29    } //end main
30  } // end class
```

실행결과

```
---------- Java Run ----------
str : Happy day
Token count : 2
Happy
day
-----------------------------
str : 2005/08/15
Token count : 3
2005
08
15
-----------------------------
Normal Termination
출력 완료 (0초 경과).
```