소설같은자바 두번째이야기

3. The Class

자바는 클래스로 시작해서 클래스로 끝나는 객체지향 언어다.
클래스를 음미하면서 밥을 먹고, 클래스를 생각하면서 잠을 자고, 클래스를 느끼면서 아침을 맞이하라!

3장 The Class

3.1 Overview

3.1.1 소개

클래스의 개념을 파악하기 위한 기초적인 작업을 1장과 2장에서 다루어 보았습니다. 이 장에서는 클래스(Class) 자체에 초점을 맞추어 학습하도록 하겠습니다.

클래스는 그 자체가 객체지향(OOP) 개념을 그대로 내포하고 있기 때문에 개념적인 면이 상당부분 포함되어 있습니다. 그렇기 때문에 개념적인 이해 없이 프로그램만 한다면, 오히려 클래스의 디자인적인 차원에서 많은 어려움을 경험하게 될 것입니다. 이미 이러한 것을 경험하신 분들이 이 책을 본다면 만족스러운 결과를 얻을 수 있을 것입니다. 이 장에서 소개되는 내용들은 다음과 같습니다.

▣ 3장의 내용
◈ 자바의 설치, 컴파일, 실행
◈ 프로그램의 기초
◈ 접근제어
◈ 메서드란?
◈ 메서드의 매개변수
◈ 메서드의 클래스 삽입
◈ 메서드와 접근제어
◈ private의 진정한 의미
◈ 메모리 관점에서 객체의 생성

클래스는 데이터 타입 생성기입니다. 보통 여러분이 클래스를 만든다면, 그 클래스를 우리는 사용자 정의 데이터 타입(User Definition Data Type)이라고 합니다. 이것을 자바에서는 클래스(Class)라 칭하지만 정확한 표현은 사용자 정의 데이터 타입입니다. 이 사용자 정의 데이터 타입은 C++, Visual C++, C#에 이르기까지 다양 분야에 적용 가능합니다. 제대로 알아둔다면 다른 언어를 접할 때에도 상당한 도움될 것입니다. 자! 그렇다면 이제부터 클래스를 탐험해 보도록 하겠습니다.

3.2 자바 SDK 설치와 실행

3.2.1 자바 SDK

텍스트 형식으로 프로그램을 작성하고, 작성된 프로그램을 컴파일한 후 실행하기 위해서는 제일 먼저 컴파일러를 구비해야 합니다. 자바의 컴파일러를 포함한 개발도구를 보통 JDK(Java Development Kit)이라고 하며 J2SDK라고도 합니다. JDK는 3가지 종류를 배포하고 있으며 그 종류는 다음과 같습니다.

▣ JDK의 종류
- ◈ J2SE (Java 2 Standard Edition)
- ◈ J2EE (Java 2 Enterprise Edition)
- ◈ J2ME (Java 2 Micro Edition)

위의 3가지 개발 환경은 탑재된 라이브러리와 기능이 약간씩 다릅니다. 하지만 J2EE를 하든 J2ME를 하든 기본적으로 J2SE에서 출발하게 됩니다. J2SE 환경에 어느 정도 익숙해지면 여러분의 관심사에 맞게 다른 환경으로 옮겨가시면 됩니다. J2EE는 기업 환경에 맞추어져 기업 솔루션을 개발하기 위한 플랫폼이기 때문에 꽤 각광받는 환경이기도 합니다. 그리고 J2ME는 모바일 환경에 맞추어져 있으니, 모바일쪽에 관심이 있으시면 한번 시도해 보시기 바랍니다.

▣ J2SE의 중요성
- ◈ J2ME를 하든 J2EE를 하든 J2SE에서 출발한다.

어떤 환경에서 어떤 작업을 하든 자바의 기본을 알기 위해서는 J2SE 환경을 구축해야 합니다. 그렇다면 J2SDK(J2SE Development Kit)를 설치해야겠죠. 우선 설치하기 전에 자바 공식 홈페이지(http://java.sun.com)에서 J2SDK를 다운로드 받으시기 바랍니다. 자바 홈페이지의 초기화면에 다운로드 바로가기 링크가 있으니 찾아보시기 바랍니다. 계속적으로 초기화면이 업데이트되지만 J2SDK는 가장 중요한 파일이기 때문에 초기화면에서 쉽게 다운로드 링크를 찾을 수 있을 것입니다. 그리고 Java Doc 또한 다운로드 받아 두시기 바랍니다. J2SDK와 같은 위치에서 다운로드 받을 수 있습니다.

▣ 윈도우 환경 설치항목
- ◈ Java 2 Standard Edition(exe 설치파일)
- ◈ Java 2 Standard Edition Documentation(zip 압축파일)

▣ Java 2 Standard Edition Development Kit(J2SDK) 설치
- ◈ 셋업 형태의 실행파일이기 때문에 간단히 설치할 수 있습니다. 버전은 1.1.x부터 1.2.x, 1.3.x, 1.4.x, 1.5.x까지 나와 있으니 가장 최근 버전을 다운받아 설치하시면 됩니다. 물론 특수한 기능이 아닌 경우에는 1.2.x 버전 정도를 설치하셔도 기본적인 자바의 기능은 사용할 수 있습니다. 하지만 최근 버전을 설치하는 것이 좋겠죠. 그리고 설치 디렉토리는 반드시 기억해 두십시오. 자바를 설치한 후 환경설정을 해야 하니까요.

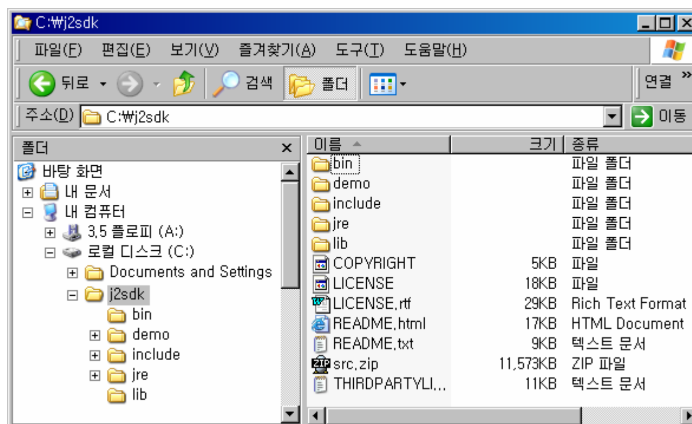
■ Java Documentation API(Java Doc API) 설치

◈ Java Doc는 여러분들이 사용할 클래스 라이브러리의 도움말을 HTML 형태로 제공하고 있습니다. 클래스의 설명서와 같은 역할을 하기 때문에 반드시 다운로드 받아 두시기 바랍니다. 클래스에 대해서 궁금한 점이 있다면 Java Doc에서 해당 클래스의 도움말을 볼 수 있습니다. Java Doc 없이 프로그램을 하는 것은 거의 불가능합니다. 압축파일 형식으로 되어 있기 때문에 특정 디렉토리에 압축만 풀면 됩니다.

3.2.2 자바의 환경설정

자바를 설치하셨다면 설치 디렉토리를 확인해 보시기 바랍니다. 만약 설치 디렉토리가 C:\j2sdk라면 다음과 같은 디렉토리가 만들어 질 것입니다.

그림 3-1 자바 설치 디렉토리



■ 주의

◈ 설치할 때 설치 디렉토리를 설정하지 않았다면 설치 디렉토리가 다를 수 있습니다.

실제 컴파일러의 역할을 하는 것은 javac.exe이며, 자바 프로그램의 실행을 위한 명령 해석기 역할을 하는 것은 java.exe입니다.

■ 자바 컴파일러와 자바 명령 해석기

◈ 자바 컴파일러 : javac.exe

◈ 자바 명령 해석기 : java.exe

이 두 개의 파일은 C:\j2sdk\bin 디렉토리에 위치하고 있습니다. 일단 자바가 설치되었다면 여러분들은 환경설정을 위해서 다음과 같은 두 가지 작업을 해 주어야 합니다.

■ 자바의 환경설정

◈ 자바의 bin 디렉토리 경로(Path) 지정

◈ 현재 작업하는 디렉토리의 클래스 패스(Class Path) 지정

java.exe와 javac.exe가 실행되는지 테스트해 보시기 바랍니다. 콘솔창에 명령을 입력한 후 Enter를 누르시면 됩니다. java.exe는 윈도우 환경에 등록되어 자동으로 실행되지만, javac.exe는 bin 디렉토리로 가서 실행하거나 아니면 경로(Path)를 잡아 주어야 실행됩니다. javac.exe의 경로(Path)를 잡아주기 위해서는 Windows 2000, Windows XP 계열에서는 다음과 같이 [내 컴퓨터] [속성] [고급]의 환경설정 부분에서 경로를 등록시켜 주면 됩니다.

그림 3-2 시스템 등록정보

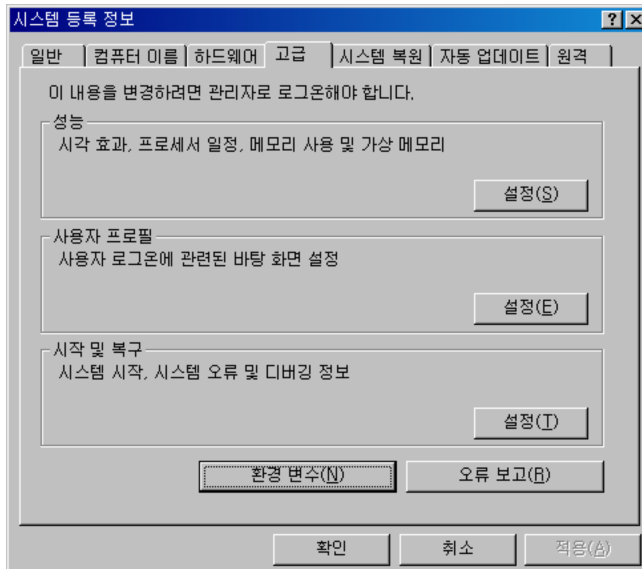
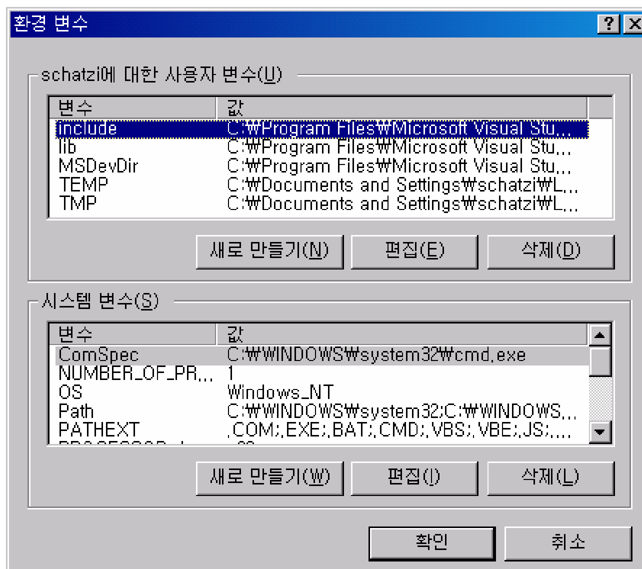


그림 3-3 환경변수 설정



환경변수 창에서 [새로 만들기]를 클릭하시면 새 사용자 변수를 등록할 수 있습니다. 자바의

Path는 다음과 같이 새 사용자변수를 등록하시면 됩니다.

그림 3-4 자바의 Path 추가하기

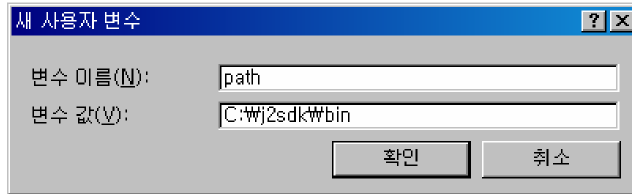
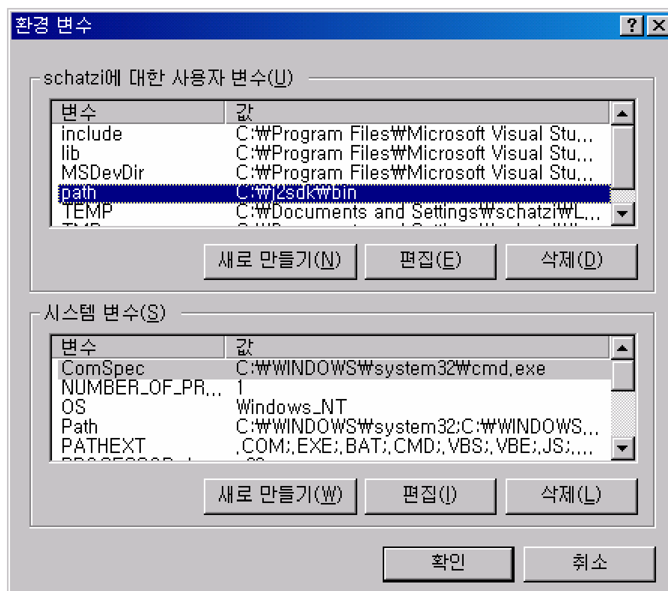
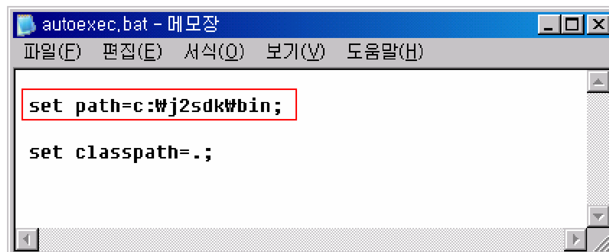


그림 3-5 추가된 자바 Path



만약 운영체제가 Windows 98이나 Windows ME 계열이라면 다음과 같이 autoexec.bat 파일에 경로(Path) 정보를 추가하시면 됩니다.

그림 3-6 Window98 & Window Me에서의 autoexec.bat-자바 Path 지정



위와 같이 C:\j2sdk\bin 디렉토리에 패스(Path)를 잡아 주었다면, 어디에서든 javac.exe를 실행할 수 있습니다. Windows 98, Windows ME는 재부팅을 해주어야 하며, Windows 2000,

Windows XP 계열은 모든 콘솔창을 전부 닫고 새로 콘솔창을 열어서 javac.exe를 입력하면 뭔가 동작하는 것이 보일 것입니다. 다음은 javac.exe의 실행 장면입니다.

그림 3-7 콘솔화면에서 javac.exe 실행

```
C:\>javac.exe
Usage: javac <options> <source files>
where possible options include:
  -g                        Generate all debugging info
  -g:none                   Generate no debugging info
  -g:{lines,vars,source}    Generate only some debugging info
  -nowarn                   Generate no warnings
  -verbose                  Output messages about what the compiler is doing
  -deprecation              Output source locations where deprecated APIs are us
ed
  -classpath <path>         Specify where to find user class files
  -sourcepath <path>        Specify where to find input source files
  -bootclasspath <path>     Override location of bootstrap class files
  -extdirs <dirs>           Override location of installed extensions
  -d <directory>            Specify where to place generated class files
  -encoding <encoding>      Specify character encoding used by source files
  -source <release>         Provide source compatibility with specified release
  -target <release>         Generate class files for specific VM version
  -help                     Print a synopsis of standard options

C:\>_
```

두 번째로 클래스 패스(Class Path)를 지정해 주어야 합니다. 클래스 패스라는 것은 라이브러리의 물리적인 위치를 지정하는 역할을 합니다. 즉 라이브러리(Library)에 대한 경로를 클래스 패스에 추가하지 않으면 라이브러리를 사용할 수 없습니다. 일반적인 자바 라이브러리는 클래스 패스가 이미 지정되어 있기 때문에 클래스 패스를 지정할 필요가 없습니다. 하지만 특정 디렉토리에서 main() 메서드를 포함한 자바 실행파일을 만들었다면 자바 명령 해석기(java.exe)는 현재 디렉토리에 있는 실행파일도 찾지 못합니다. 이러한 문제를 해결하기 위해서 현재 자신이 작업하고 있는 디렉토리를 클래스 패스에 추가해 주어야 합니다. 다음은 현재 작업 디렉토리에 대한 클래스 패스를 지정하는 예입니다.

그림 3-8 클래스 패스 추가하기

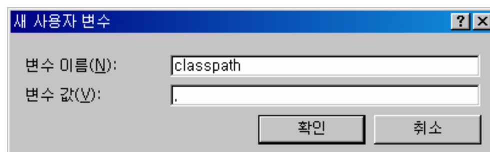


그림 3-9 추가된 클래스 패스

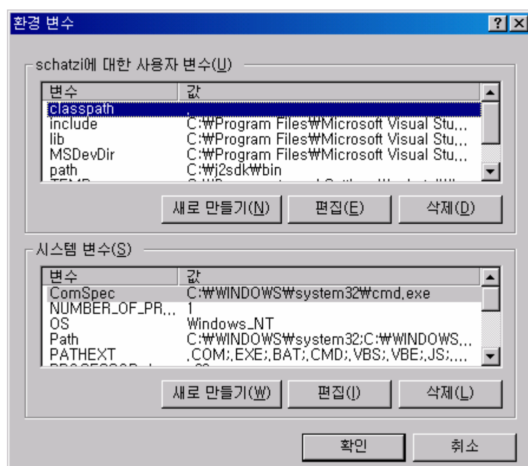
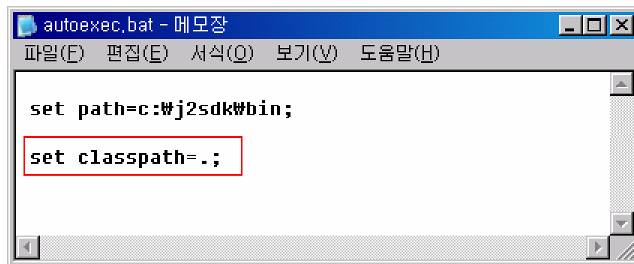


그림 3-10 Window98 & Window Me에서의 autoexec.bat-클래스 패스 지정



현재 작업 디렉토리를 클래스 패스에 추가시켜 주지 않으면 컴파일은 되지만 실행이 되지 않는 현상이 발생합니다. 처음 자바를 컴파일하고 실행하신다면 bin의 패스지정과 현재 디렉토리의 클래스 패스가 추가되었는지 반드시 확인하시기 바랍니다.

3.2.3 컴파일과 실행

자바 파일을 편집하는 편집기는 여러분이 원하는 텍스트 편집기를 사용하시면 됩니다. 상용 자바 편집기도 존재하지만 비상용 편집기도 있으니 여러분들이 원하는 편집기를 선택하시기 바랍니다. 자바 프로그램 작성을 위해서 다음과 같은 코드를 작성하도록 하겠습니다.

§3-1 Sample.java

```
01:  /**
02:  자바의 컴파일과 실행을 테스트하는 프로그램
03:  **/
04:  public class Sample{
05:      public static void main(String[] args){
06:          int a = 5;
07:          int b = 10;
08:          int c = a + b;
09:          System.out.println("결과 c = " + c);
10:      } //end of main
11:  } //end of Sample class
```

```
C:\javasrc\chap03>javac Sample.java
//컴파일된 파일 보기
C:\javasrc\chap03>DIR Sample.*;
08-08  오전 11:33            638 Sample.class
08-08  오전 11:33            388 Sample.java
2개 파일           1,026 바이트
//프로그램 실행하기
C:\javasrc\chap03>java Sample
```

```
결과 c = 15
```

자바 코드를 작성하셨다면 파일명은 다음과 같이 만드시기 바랍니다.

▣ 자바 파일 이름 작성법
◈ 클래스명 + .java
◈ ex) Sample.java
◈ 반드시 클래스명과 파일명은 같아야 한다.(규칙)

위의 파일명은 Sample.java가 될 것입니다. 그리고 이 파일을 컴파일할 때에는 다음과 같이 javac.exe를 이용하면 됩니다.

▣ Sample.java의 컴파일
◈ C:\javasrc\chap03>javac Sample.java
◈ C:\javasrc\chap03>dir Sample.*
◈ 07-03 오전 12:42 638 Sample.class
◈ 06-20 오후 10:32 381 Sample.java
◈ 2개 파일 1,019 바이트

Sample.java를 컴파일하면 Sample.class라는 클래스 파일이 하나 생성됩니다. 이 파일이 바로 main()을 포함한 자바의 실행파일입니다. 자바 실행파일의 이름은 클래스의 이름과 동일합니다. 그 다음으로 Sample.class를 실행해 보도록 하겠습니다. 실행하는 방법은 다음과 같습니다.

▣ Sample.class의 실행
◈ C:\javasrc\chap03>java Sample
◈ 결과 c = 15

실행결과를 콘솔창에서 확인할 수 있습니다. 자바를 실행할 때에는 확장자명은 붙이지 않습니다. 단, 위의 과정이 순서대로 이루어지기 위해서는 반드시 자바를 설치한 후 환경 설정을 맞추어야 합니다. 만약 클래스 패스를 지정하지 않았다면 다음과 같은 방식으로 컴파일하고 실행해 보시기 바랍니다.

▣ 클래스 패스를 동적으로 지정하면서 컴파일하고 실행
◈ C:\javasrc\chap03>javac Sample.java
◈ C:\javasrc\chap03>java -classpath . Sample

이 방법은 컴파일한 후 실행할 때 .class 파일이 현재 디렉토리에 있다는 것을 동적으로 지정하는 방법입니다. 즉 Sample.class를 자바의 디폴트 경로와 현재 작업 디렉토리에서 검색하게 됩니다. 물론 클래스 패스의 경로상에 존재한다면 자바 프로그램이 실행될 것입니다.

3.3 메서드

3.3.1 클래스의 구성요소

클래스의 구성요소는 두 가지로 나눌 수 있습니다. 클래스가 대단해 보이기는 하지만 사실 알고 보면 딱 두 가지 구성요소를 포함하고 있습니다. 변수(Variable)가 클래스의 구성원이 된다는 것은 이미 2장에서 배운 사실입니다. 그리고 메서드(Method)가 추가된다는 것도 언급을 한 적이 있습니다. 맞습니다. 클래스의 구성요소는 변수(Variable)와 메서드(Method)로 이루어져 있습니다.

▣ 클래스의 구성요소 I
◈ 변수(Variable)
◈ 메서드(Method)

클래스의 내부에 존재한다는 의미에서 변수와 메서드를 클래스의 멤버(Member)라고 부릅니다. 일반적으로 변수를 멤버 변수(Member Variable)라고 부르며, 메서드를 멤버 메서드(Member Method)라고 부릅니다.

▣ 클래스의 구성요소 II
◈ 멤버 변수(Member Variable) 또는 멤버 필드(Member Field)
◈ 멤버 메서드(Member Method)

이것이 클래스의 구성요소의 전부입니다. 더 없습니다. 하지만 메서드가 추가된다는 사실만으로도 엄청난 일을 해내게 됩니다. C 언어의 가장 큰 문제는 클래스의 개념이 없다는 것입니다. 즉 변수와 메서드가 붙어있지 않다는 것이죠. 이에 반해 클래스는 구조체의 기능과 메서드의 기능을 합쳐 놓은 것입니다.

▣ 클래스의 구성요소 III
◈ 클래스 = 데이터 + 메서드
◈ 클래스 = C 언어의 구조체 + 메서드

클래스에 대해서 많은 것을 배웠지만 앞장에서 단순한 변수의 집합이라는 측면에서 클래스만을 학습했습니다. 진짜 클래스는 단순한 데이터의 집합으로 이루어진 데이터 타입이 아니라, 데이터와 메서드가 결합된 형태의 사용자 정의 데이터 타입을 말합니다.

3.3.2 메서드

1, 2장에서 데이터의 집합이라는 측면에서 클래스를 공부했습니다. 데이터와 메서드가 결합된 형태의 클래스를 배우기 전에 우선 메서드의 개념부터 파악해 보도록 하죠. 일단 용어적인 정리부터 하고 넘어가도록 하겠습니다. C 언어나 C++ 언어에서는 함수(Function)라는 용어를 사용하

며, 자바에서는 함수라는 단어 대신 메서드(Method)라는 용어를 사용합니다.

▣ 함수(Function)와 메서드(Method)에 대한 용어
◈ 자바에서는 함수(Function)를 메서드(Method)라는 용어로 대체하고 있다.
◈ 앞으로 이 책에서는 함수 대신에 메서드라는 용어를 사용하겠습니다.

메서드란 여러 개의 작업을 하나로 묶어서 관리할 수 있는 작업의 덩어리입니다. 이 작업의 덩어리는 사용자가 필요로 할 때 해당 작업을, 대표하는 이름을 통해서 반복적으로 사용(호출)할 수 있습니다. 메서드는 두 가지 측면에서 그 역할을 조명해 볼 수 있습니다. 일만 하는 메서드와 일을 한 후 값을 리턴하는 메서드로 나눌 수 있습니다.

▣ 메서드의 종류
◈ 일만 하는 메서드
◈ 일을 한 후 값을 리턴하는 메서드

간단히 덧셈을 위한 메서드가 있다고 가정하죠. 그리고 덧셈을 위한 메서드를 위의 두 가지 측면에서 작성해 보도록 하겠습니다.

▣ 일만 하는 메서드
```
◈ void sumA(int x, int y){
◈         int c;
◈         c = x + y;
◈         System.out.println("c=" + c);
◈         return; //값을 리턴하지 않고 단순히 끝나 버림
◈ }
```

▣ 값을 리턴하는 메서드
```
◈ int sumB(int x, int y){
◈         int c;
◈         c = x + y;
◈         return c; //c의 값을 리턴
◈ }
```

일만 하는 메서드의 경우 어떠한 값도 리턴하지 않는다는 이유에서 단순히 return이라고 했습니다. 그리고 값을 리턴하지 않기 때문에 void형으로 메서드를 선언했습니다.

▣ 리턴(return)
◈ 리턴은 메서드의 종료를 의미합니다. 메서드를 끝내기 위해서는 메서드 내의 모든 작업을 완료하든지, 아니면 return을 통해서 메서드의 특정 부분에서 작업을 끝내는 방법을 이용하면 됩니다. return은 값을 가지고 끝낼 수도 있으며, 값을 리턴하지 않고 단순히 끝낼 수도 있습니다. 만약 값을 가지고 리턴한다면 반드시 리턴하는 값의 형을 메서드의 선언부에 명시해 주어야 합니다. 그리고 값을 리턴하지 않는다면 메서드의 선언부에 void형으로 명시해 주면 됩니다.

메서드를 만들었는데 사용하지 않는다면 쓸모가 없겠죠? 이렇게 만들어진 메서드는 여러분들이

작성하는 소스 코드의 원하는 곳에서 반복적으로 호출할 수 있습니다. 호출할 때에는 메서드의 이름으로 호출하게 됩니다. 다음은 메서드의 이름으로 호출하는 예를 보여주고 있습니다.

▣ 일만 하는 메서드의 호출
◈ sumA(3,4);

sumA()는 일만하기 때문에 호출하면 그 자체로 끝입니다. 호출이 한번 이루어지겠죠. 호출한 후 메서드의 작업을 모두 마치면 자동으로 끝이 나게 됩니다. 현재는 sumA() 메서드의 끝부분에 명시적으로 return을 표시하고 있습니다. sumA()와 같이 아무 것도 리턴하지 않는 경우 return 표시를 생략해도 됩니다.

sumB()는 다릅니다. sumB() 메서드는 호출과 동시에 sumB(3,4) 자체가 변수의 역할을 하게 됩니다. sumB(3,4)가 가지는 값은 내부에서 리턴하는 값이 됩니다.

▣ 값을 리턴하는 메서드의 호출
◈ int c = sumB(3,4);

sumB(3,4)가 최종적으로 가지는 값은 7이 되며, sumB(3,4) 자체가 변수의 역할을 하기 때문에 다시 int형 변수 c에 값을 할당하고 있습니다. 리턴값을 가지는 메서드가 변수의 역할을 한다는 것과 메서드의 매개변수에 대해서는 앞으로 계속 논하게 될 것입니다.

3.3.3 메서드는 변수다.

메서드의 역할에 대해서 다시 한번 따져 보도록 하겠습니다. 메서드는 완벽하게 변수의 역할을 하게 됩니다. 다음의 코드를 보시면 변수 a도 7의 값을 가지며, sumB(3,4) 또한 7의 값을 가지고 있습니다.

▣ 변수와 메서드의 할당
◈ int a = 7;
◈ sumB(3,4);

뭐가 다른 것일까요? 둘 다 7의 값을 가지고 있으며 변수로서의 역할을 합니다. 다음의 경우 변수 a와 변수 sumB(3,4)가 변수의 역할을 하는 예를 보여주고 있습니다.

◈ int a = 7;
◈ int b = a;
◈ int c = sumB(3,4);

분명한 것은 a도 7의 값을 가지며, sumB(3,4) 또한 7의 값을 가진다는 것입니다. 그리고 a도 완벽한 변수로서의 기능을 하며, sumB(3,4)도 완벽하게 하나의 변수 역할을 한다는 것입니다. 이것은 할당의 방법을 따져보면 쉽게 알 수 있습니다. 다음의 경우는 기본 데이터 타입 변수와 메서드 변수의 값을 재할당하는 방법을 보여주고 있습니다.

▣ 변수의 재할당
◈ int a = 7;
◈ a = 10;

▣ 메서드의 재할당
◈ int c = sumB(3,4);
◈ c = sumB(5,5);

일반 변수와 리턴값을 가지고 있는 메서드의 차이점은 할당의 방법에 있습니다. 기본 데이터 타입의 변수에 값을 할당하는 방법은 직접할당을 원칙으로 하고 있습니다. 그러나 sumB()에서는 간접할당을 원칙으로 하고 있습니다. 즉 sumB()에서는 매개변수라는 것을 통해서 메서드 내부로 값을 전달하고, 메서드 내부에서 만들어 낸 값을 리턴함으로써 값이 결정됩니다. 결과적으로 메서드는 간접할당을 기본으로 하는 것입니다.

▣ 기본 데이터 타입 변수와 메서드의 차이점
◈ 기본 데이터 타입 변수는 직접할당을 원칙으로 한다.
◈ 메서드는 간접할당을 원칙으로 한다.

리턴값을 가진 메서드는 호출과 동시에 그 자체가 변수의 역할을 할 수 있습니다.

3.3.4 리턴과 매개변수

메서드가 변수의 역할을 하는 것에 대해서 알아보았습니다. 좀 더 자세하게 메서드를 분석해 보도록 하죠. 다음은 기본 데이터 타입 변수의 선언과 앞에서 만든 sumB() 메서드의 선언을 보여주고 있습니다.

▣ 기본 데이터 타입 변수의 선언
◈ int a;

▣ int a의 분해
◈ 데이터 타입 : int
◈ 변수 : a

▣ sumB() 메서드의 선언
◈ int sumB(int x, int y){
◈ 　　　int c;
◈ 　　　c = x+ y;
◈ 　　　return c;
◈ }

▣ sumB()의 분해
◈ 리턴형 : int

◈ 함수이름 : sumB
◈ 매개변수 : (int x, int y)
◈ 작업의 내용 : { .... }
◈ 종료키워드 : return
◈ 리턴값 : c

▣ 참고
◈ 메서드 내부의 변수 c는 지역 변수이며, 매개변수 x, y 또한 지역 변수이다.
◈ 변수 x, y는 메서드 외부와 연결된 지역변수에 해당한다.

기본 데이터 타입 변수와 메서드의 선언 부분을 하나씩 분해해 보도록 하죠. 먼저 int a는 말 그대로 int라는 모양의 메모리를 생성하는데 a라는 이름을 붙여놓은 것입니다. int sumB까지는 int a라는 것과 비슷합니다. 하지만 sumB 옆 부분에 뭔가가 있습니다. 이것을 매개변수(Parameter)라고 합니다. 매개변수를 통해서 메서드 외부에서 메서드 내부로 값을 넘겨받게 됩니다.

sumB 메서드는 블록({}) 내부에 존재하는 작업을 한 후 리턴에 의해서 값을 만들어 내게 됩니다. 그리고 블록 내부에 리턴할 형은 반드시 메서드의 이름 앞에 명시해야 합니다.

예를 들어 어린 아이에게 심부름을 시키는 메서드를 만들어 보도록 하겠습니다. 메서드의 이름은 '휴지사오렴'이라고 정하겠습니다. 그리고 어린 아이에게 돈을 주어야 '휴지'를 사오겠죠. 돈이라는 매개변수를 사용하도록 하겠습니다.

돈을 매개변수에 넣어주고 '휴지사오렴(2000원)'하고 명령을 내리면 어린 아이는 '휴지'를 리턴하게 될 것입니다. 내부의 작업이 어떻게 되든간에 리턴타입이 '휴지'이기 때문에 '휴지사오렴'의 최종 결과는 '휴지'가 될 것입니다. 이것을 메서드로 만들어 보면 다음과 같습니다.

▣ 변수와 메서드의 선언
◈ 휴지 휴지사오렴(돈 m){
◈ m을 가지고 슈퍼로 간다.;
◈ 휴지를 찾아서 잡는다.;
◈ 휴지를 사서 가져온다.
◈ return 두루마리휴지;
◈ }

위의 경우와 같이 매개변수가 있을 경우 돈을 주지 않으면 휴지를 사오지 않을 것입니다. 이 때 돈 자체는 매개변수가 됩니다. 돈을 건네 받을 중간 역할을 하는 것이 바로 매개변수의 기본 원리입니다. 실제 메서드를 호출해 보죠.

▣ 휴지사오렴() 메서드의 호출
◈ 휴지 t = 휴지사오렴(2000);

2000원의 돈을 휴지사오렴() 메서드의 매개변수 m에 넣어주면 작업을 끝낸 뒤 휴지를 리턴할 것입니다. 즉 매개변수는 메서드 외부로부터 들어오는 데이터를 내부로 받아내기 위한 중간 역할

을 하는 것입니다. 여기서 한가지 주의할 것은 매개변수는 메서드 내에 존재하는 지역변수라는 것입니다. 이 매개변수는 메서드 내부의 지역변수이면서 외부로부터 들어오는 값을 넘겨받을 수 있는 유일한 통로가 됩니다.

▣ 메서드의 매력
◈ 메서드는 작업을 하나로 묶어서 관리할 수 있다는 측면에서 아주 효율적이다.
◈ 언제든 필요하면 반복적으로 호출이 가능하다.
◈ 호출할 때 매개변수만 만족시켜 주면 언제든지 호출할 수 있다.

결론적으로 메서드는 데이터를 넘겨받기 위해서 매개변수라는 것을 이용합니다. 그리고 리턴에 의해서 최종적으로 메서드가 가질 값을 결정하게 됩니다. 메서드를 만들 때 반드시 리턴할 데이터의 형을 지정해 주어야 합니다. 마지막으로 메서드의 리턴값이 존재한다면, 호출되었을 때 하나의 변수로서의 역할을 하게 되는 것도 기억해 두시기 바랍니다.

▣ 메서드를 만들 때 주의 사항
◈ 리턴값과 선언부에 명시된 리턴타입은 반드시 일치해야 한다.
◈ 값을 리턴하고자 할 때에만 리턴해야 한다.
◈ 호출할 때 매개변수의 개수와 매개변수의 형을 맞추어서 호출해야 한다.
◈ 메서드의 이름을 짓는 것은 프로그래머의 몫이며, 동작적인 의미를 사용하는 것이 좋다.
◈ 일반적으로 자바에서는 메서드의 이름은 소문자로 시작하고, 새로운 단어가 시작하면 다시 대문자로 시작한다.(예: getData(), executeUpdate())

3.3.5 값복사와 매개변수의 전달

언어를 공부하면서 단순한 원리 하나를 알면 많은 것이 해결되는 경우가 있습니다. 값복사(Value Copy)라는 말이 그런 예에 해당하는 것 같습니다. 만약 값복사를 제대로만 알고 있었다면, 그리고 누군가가 값복사를 한번이라도 언급했더라면, 여러분들이 많은 부분에 있어서 헤매지 않았을 것입니다. 이번에는 값복사에 대해서 알아보도록 하죠. 그리고 값복사와 매개변수가 어떠한 관련이 있는지에 대해서 학습하도록 하겠습니다.

이해를 돕기 위해서 기본 데이터 타입의 값복사부터 시작하겠습니다. 기본 데이터 타입은 변수를 선언했을 때 변수의 선언과 동시에 메모리가 생성됩니다. 기본 데이터 타입은 우선 변수의 선언 자체가 메모리의 생성의 의미를 담고 있기 때문에 new 연산자를 사용하지 않아도 자동으로 메모리가 생성됩니다.

▣ 메모리의 생성
◈ int a;

int a라고만 해도 메모리는 생성됩니다. 즉 4바이트의 메모리가 생성된 것입니다. 묵시적으로 내부에서 메모리의 생성이 이루어지는 것입니다. 그리고 a 자체는 메모리를 그대로 가리키게 되는 것입니다. 이 개념은 모든 기본 데이터 타입(Primitive Data Type)에 적용되는 불변의 법칙입니다.

기본 데이터 타입을 사용할 때 여러분이 주의할 것은 바로 기본 데이터 타입끼리의 할당에서 어떻게 메모리의 값이 복사되느냐의 문제입니다.

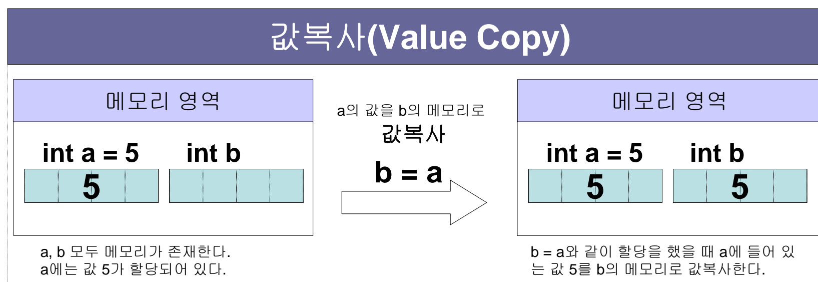
▣ 값복사(Value Copy)란?
◈ 값복사란 두 개의 메모리가 존재하고 한쪽의 메모리에 들어있는 값을 다른 쪽의 메모리로 그 값만을 복사하는 행위를 말한다.

다음과 같은 코드의 경우 할당의 기본 원리는 아주 단순합니다.

▣ 값복사의 예
◈ int a = 5;
◈ int b;
◈ b = a;

너무나 단순한 구문이지만 이것의 원리를 제대로 이해하신다면 기본 데이터 타입뿐만 아니라 참조 타입까지 확장해서 생각할 수 있습니다. 위의 코드에서 a라는 메모리는 이미 생성되어 있습니다. 그리고 b라는 메모리 또한 생성된 상태입니다. 마지막으로 a를 b에 할당하고 있습니다. 문제는 'b = a'라는 구문에서 어떻게 할당이 이루어지느냐의 문제입니다. a에 들어있는 5가 어떠한 원리에 의해서 할당되는 것일까요? 다음의 그림에서 그 해답을 찾아보시기 바랍니다.

그림 3-11 값복사의 예



메모리는 a, b 둘 다 존재합니다. 선언을 했으니 기본 데이터 타입은 무조건적으로 메모리가 생성됩니다. 그리고 메모리가 둘 다 존재하는 상태에서, 'b = a'의 의미는 a의 메모리에 있는 값만을 다른 메모리 b로 값복사를 하겠다는 의미입니다. 즉 b는 a에 있는 값만을 자신의 메모리로 복사해 오는 것입니다. 메모리가 둘 다 존재하는 상태에서 한쪽의 메모리에 들어있는 값을 다른 쪽의 메모리로 복사하는 행위를 값복사(Value Copy)라고 합니다.

▣ 매개변수의 전달
◈ 자바의 매개변수의 전달은 값복사의 기법만을 사용한다.
◈ 이것을 값에 의한 호출(Call By Value) 또는 값복사에 의한 호출이라고 한다.

자바에서 메서드를 호출할 때 매개변수가 전달되는 방식은 몽땅 이 방법을 이용하고 있습니다. 즉 값복사를 이용해서 매개변수를 전달한 후 메서드를 호출하는 것입니다. 다음 구문에서 값복사의 의미를 되새겨 보시죠.

▣ sumB() 메서드의 선언
◈ int sumB(int x, int y){
◈ int c;
◈ c = x + y;
◈ return c;
◈ }

▣ 메서드의 호출
◈ int a = 3;
◈ int b = 4;
◈ int c = sumB(a,b);

지금까지 배운 것을 기초로 해서 본다면 다음 3가지가 값복사에 해당합니다.

▣ 값복사의 예 I
◈ x = a; //메서드를 호출할 때 값복사 발생
◈ y = b; //메서드를 호출할 때 값복사 발생
◈ c = sumB(a,b); //메서드의 리턴값을 c의 메모리에 값복사

알고 보면 다음의 구문도 메모리로 데이터를 값복사하는 행위에 해당합니다.

▣ 값복사의 예 II
◈ int a = 3;
◈ int b = 4;
◈ c = a + b;

자바에서는 값복사의 기법만이 존재합니다. 이후에 배우게 될 모든 매개변수의 전달이나 모든 할당들이 값에 의한 복사(Value Copy)에 해당합니다. 이것에 대해서는 앞으로 아주 자세하게 논하게 될 것입니다.

3.3.6 결론

컴퓨터 언어를 처음 접할 때의 기분이 생각나는군요. 메서드가 뭐지! 저걸 어떻게 만들어!라고 생각할 때가 필자 또한 있었습니다. 처음엔 메서드가 두렵고, 두 번째엔 포인터가 두렵고, 세 번째엔 자료구조가 두렵고, 마지막으로 알고리즘이 두렵더군요. 자바에는 포인터 대신에 참조변수라는 것을 사용하니 포인터 대신에 참조변수가 두렵겠군요.(^.^)

하지만 반복학습과 원리학습 위주로 기초를 튼튼히 하신다면 쉽게 극복할 수 있을 것입니다. 언어를 조금 접하신 분들이야 그렇게 어렵겠습니까? 모르고 있었다면 알았으니 다행이고, 그래도 별 무리 없이 이해할 수 있을 테니 큰 걱정은 하지 않습니다. 하지만 초보에겐 또 다른 장벽이겠죠. 필자가 추천해 주고 싶은 말은 천천히 공부하시라는 것입니다. 그리고 어떠한 책이든 끝까지 천천히 보시기 바랍니다. 너무 빨리 보고 키보드만 두들기면 당장은 빠를 수 있겠지만 나중에는

제일 느린 사람이 되고 맙니다. 천천히 정확하게 보시는 것이 제일 좋은 방법입니다.

아마! 컴퓨터에 관련된 내용을 배우면서 천천히 하나씩이라는 말을 마음속으로 제일 많이 한 것 같습니다. 글은 단어by단어로 줄by줄로 읽고, 코딩은 의심스러울 때 한번 정도만 하시기 바랍니다. 먼 길로 돌아가기 싫으시다면!

3.4 클래스와 메서드

3.4.1 클래스 내의 메서드

클래스의 구성요소는 변수와 메서드입니다. 지금까지 메서드에 대해서 알아보았으니 이제 클래스 내부로 메서드가 삽입되는 예를 보기로 하죠. 메서드가 클래스로 삽입되면서 클래스는 전혀 다른 모습이 됩니다. 기존의 구조체에 메서드를 포함시켜 새로운 객체지향의 문법을 만들어내고 있는 것입니다. 클래스에 메서드가 들어 갈 수 있다면 한번 넣어보죠. 어떻게 사용할 수 있는지 실질적인 예를 만들어 보도록 하죠.

▣ 멤버(Member)

◈ 보통 클래스 내에 존재하는 변수나 메서드를 클래스의 멤버(Member)라고 부른다.

◈ 앞으로는 이 책에서도 멤버로 명명하며, 클래스 내의 변수를 멤버 필드(Member Field) 또는 멤버 변수(Member Variable)라고 부르며, 클래스 내의 메서드를 멤버 메서드(Member Method)라고 부르겠습니다.

다음은 클래스 내에 sum() 메서드가 추가된 예를 보여주고 있습니다. 특별한 의미 없이 단순히 클래스 내에 메서드를 삽입한 예입니다.

§3-2 Top.java

```
01:  /**
02:  클래스에 메서드를 포함시킨 예
03:  **/
04:  public class Top{
05:     public int a; //멤버변수
06:     public int b; //멤버변수
07:     public int sum(int x, int y){ //sum 메서드의 선언
08:        return x + y;
09:     }
10:  } //end of Top class
```

```
C:\javasrc\chap03>javac Top.java
```

Top이라는 새로운 데이터 타입을 생성하고 있습니다. Top 클래스 내부에는 3개의 멤버가 있는 것을 볼 수 있죠. 2개의 멤버 필드와 1개의 멤버 메서드를 포함하고 있습니다. 다음으로 Top형의 객체를 생성한 후 해당 객체에 값을 할당하고 메서드를 호출하는 방법을 보도록 하죠.

§3-3 TopMain.java

```
01:  /**
02:  Top 클래스를 테스트하는 예제
03:  **/
04:  public class TopMain {
05:     public static void main(String[] args){
06:        Top t = new Top(); // Top 객체 생성
07:        t.a = 100; // 멤버 변수 a 에 값 할당
08:        t.b = 200; // 멤버 변수 b 에 값 할당
09:        int s = t.sum(3, 5); // sum() 메서드 호출한 후 리턴값을 s 로 값복사
10:
11:        //Top t 의 멤버 변수 출력
12:        System.out.println("a 는:" + t.a);
13:        System.out.println("b 는:" + t.b);
14:        //메서드 호출 결과 출력
15:        System.out.println("t.sum(3,5)의 결과는:" + t.sum(3,5));
16:        System.out.println("s 는:" + s); //s 의 값 출력
17:     } //end of main
18:  } //end of TopMain class
```

```
C:\javasrc\chap03>javac TopMain.java
C:\javasrc\chap03>java TopMain
a는:100
b는:200
t.sum(3,5)의 결과는:8
s는:8
```

일단 데이터 타입을 사용하기 위해서 변수를 생성한 후 메모리를 할당하고 있습니다. 객체의 메모리는 new 연산자와 생성자를 이용해서 만들며 그 구문은 다음과 같습니다.

▣ Top 객체 생성
◈ Top t = new Top();

객체 t를 이용해서 멤버 변수에 값을 할당하기 위해서 점(.)으로 접근하고 있습니다.

▣ Top 객체 t의 멤버에 값할당
◈ t.a = 100; // 멤버 변수 a에 값할당
◈ t.b = 200; // 멤버 변수 b에 값할당

메서드를 호출하는 방법 또한 멤버 변수에 접근하는 것과 같이 점(.)을 이용해서 호출하면 됩니다.

▣ Top 객체 t의 멤버 메서드 호출

◈ int s = t.sum(3, 5); // sum() 메서드 호출 후 s로 값복사

물론 메서드를 호출할 때에는 매개변수를 주어야 하며, 호출과 동시에 리턴값을 넘겨받을 수 있습니다. 메서드도 변수처럼 접근하고 있으며, 메서드의 방식대로 할당하며 꼭 변수처럼 사용됩니다. t.sum(3,5)는 그 자체가 변수의 역할을 할 수 있으며, s라는 변수에 그 값을 다시 할당하고 있습니다.

▣ System.out.println()

◈ 화면으로 출력하기 위한 표준 출력을 의미하는 메서드입니다. 지금은 이 구문이 매개변수로 주어진 값을 콘솔창으로 출력한다는 것만 알아두시기 바랍니다. 4장에서 자세히 배우게 될 것입니다.

3.4.2 변수와 메서드 결합의 의미 I

클래스에 메서드가 추가되는 예를 알아보았습니다. 어떻게 클래스 내에 메서드가 들어가는지 알았을 것입니다. 그렇다면 왜라는 질문을 던져 보죠. 왜 변수들의 집합인 구조체에 메서드를 추가했을까요? 너무나도 간단한 질문일 수도 있으며 너무나도 복잡한 질문일 수도 있습니다. 메모리적인 측면과 방법론적인 측면을 고려해서 생각해 보도록 하죠.

▣ 질문

◈ 왜 변수들의 집합인 구조체에 메서드를 추가했을까?

이것의 차이를 알아보기 위해서 메서드만으로 이루어진 클래스와 메서드와 변수가 결합된 클래스를 테스트해 보기로 하죠. 테스트할 클래스는 다음과 같습니다.

▣ 테스트할 클래스

◈ 메서드로만 이루어진 클래스(SeparatedData 클래스)

◈ 메서드와 변수가 결합된 클래스(UnitedData 클래스)

▣ SeparatedData 클래스의 특징

◈ 메서드로만 이루어진 클래스는 메서드를 호출할 때마다 필요한 데이터를 넣어주어야 한다.

먼저 메서드로만 이루어진 클래스부터 학습하도록 하죠.

§3-4 SeparatedData.java

```
01:  /**
02:  메서드로만 이루어진 클래스
```

```
03:  **/
04:  public class SeparatedData {
05:     public int plus(int x, int y){ //멤버메서드
06:        return x + y;
07:     }
08:     public int minus(int x, int y){ //멤버메서드
09:        return x - y;
10:     }
11:     public int divide(int x, int y){ //멤버메서드
12:        return x / y;
13:     }
14:     public int mul(int x, int y){ //멤버메서드
15:        return x * y;
16:     }
17:  } //end of SeparatedData class
```

```
C:\javasrc\chap03>javac SeparatedData.java
```

SeparatedData라는 클래스는 단순한 메서드의 집합입니다. 클래스 내부에 데이터(멤버 변수)는 존재하지 않습니다. 다음과 같이 객체를 생성한 후 메서드를 호출하기 위해서는 매개변수를 넣어준 후 메서드를 호출해야 합니다. 즉 메서드를 호출할 때마다 데이터를 매개변수로 넣어주어야 합니다.

▣ SeparatedData 객체의 생성과 메서드의 호출

◈ SeparatedData d = new SeparatedData();
◈ int a = d.minus(5,10);
◈ int b = d.plus(5,10);
◈ int c = d.divide(5,10);
◈ int d = d.mul(5,10);

이것을 테스트하는 예는 다음과 같습니다.

§3-5 SeparatedDataMain.java

```
01:  /**
02:  SeparatedData 를 테스트하는 클래스
03:  **/
04:  public class SeparatedDataMain{
05:     public static void main(String[] args){
06:        SeparatedData d = new SeparatedData(); //SeparatedData 객체 생성
07:        System.out.println(d.minus(5,10)); //minus() 메서드 호출 후 결과 출력
08:        System.out.println(d.plus(5,10)); //plus() 메서드 호출 후 결과 출력
09:        System.out.println(d.divide(5,10)); //divide() 메서드 호출 후 결과 출력
10:        System.out.println(d.mul(5,10)); //mul() 메서드 호출 후 결과 출력
```

```
11:     } //end of main
12:  } //end of SeparatedDataMain class
```

```
C:\javasrc\chap03>javac SeparatedDataMain.java
C:\javasrc\chap03>java SeparatedDataMain
-5
15
0
50
```

이러한 프로그램 기법은 C 언어 계열에서 사용하던 프로그램 기법이었습니다. 객체지향이 없던 시절 아주 일반화된 프로그래밍 기법입니다. C 언어의 라이브러리는 함수들의 집합으로 이루어져 있으며, 필요할 때 함수를 코드에 포함시켜 사용했습니다. 즉 함수 형태의 라이브러리였습니다. 필요한 데이터는 함수를 호출할 때마다 매개변수를 통해서 넣어주고 호출했습니다. 이러한 측면에서 본다면 메서드로만 이루어진 클래스는 C 언어의 함수 라이브러리와 별 차이가 없습니다. 그럼, 데이터와 메서드가 결합된 형태를 보고 그 차이점을 알아보도록 하죠.

▣ **참고**

◈ main()이 클래스 내부에 들어 있는 것은 자바에서는 아주 당연한 것입니다. C나 C++ 언어에 익숙하신 분들은 '왜 main() 메서드가 클래스 안에 있지?'라고 생각할 수도 있습니다. C나 C++ 에서는 메서드가 독립적으로 존재할 수 있지만, 그리고 메인은 독립된 함수로 존재해야 되지만, 자바에서는 무조건적으로 클래스 내부에 넣어야 합니다. 자바에서 클래스 외부에 존재하는 변수나 메서드는 아예 존재하지도 않습니다.

3.4.3 변수와 메서드 결합의 의미 II

앞에서는 메서드로만 이루어져 있는 클래스 즉 메서드와 변수가 분리되어 있을 때를 테스트하기 위해서 메서드로만 이루어진 클래스를 작성해 보았습니다. 이제 변수와 메서드가 결합된 형태의 클래스를 작성해 보도록 하죠.

▣ UnitedData 클래스의 특징

◈ 데이터를 멤버 변수로 보유한 상태에서 메서드를 호출한다. 그렇기 때문에 메서드를 호출할 때 매개변수를 통해서 데이터를 넣어줄 필요가 없다.

◈ 변수와 메서드가 결합된 형태로 멤버 변수를 멤버 메서드가 이용하는 예를 보여주고 있다.

메서드와 변수가 결합된 형태의 클래스를 만들기 위해서 앞의 SeparatedData 클래스에 변수를 추가한 후 약간 수정했습니다.

§3-6 UnitedData.java

```
01:  /**
02:  메서드와 변수가 결합된 형태의 클래스
```

```
03:  **/
04:  public class UnitedData {
05:     public int x; //멤버 변수의 선언
06:     public int y; //멤버 변수의 선언
07:     public int plus(){ //멤버 메서드의 선언
08:        return x + y;
09:     }
10:     public int minus(){ //멤버 메서드의 선언
11:        return x - y;
12:     }
13:     public int divide(){ //멤버 메서드의 선언
14:        return x / y;
15:     }
16:     public int mul(){ //멤버 메서드의 선언
17:        return x * y;
18:     }
19:  } //end of UnitedData class
```

```
C:\javasrc\chap03>javac UnitedData.java
```

UnitedData는 2개의 멤버 변수와 4개의 멤버 메서드를 가지고 있습니다. 특징적인 것은 2개의 멤버 변수를 4개의 멤버 메서드가 이용하고 있다는 것입니다. 멤버 변수인 x, y의 값을 한번만 셋팅해 주면, 클래스 내부의 메서드들은 값을 매개변수로 받을 필요가 없는 것입니다. 멤버 변수에 할당된 값을 이용해서 멤버 메서드를 호출할 수 있습니다. 이것은 상태 유지의 개념을 담고 있습니다.

▣ 상태 유지의 개념

◈ UnitedData 클래스에서 멤버 변수 x, y의 값이 한번 설정되면, 이 멤버를 사용하는 메서드에 영향을 미치기 때문에 객체의 상태를 유지하면서(셋팅된 값으로 유지하면서) 지속적으로 메서드를 호출할 수 있다.

객체의 상태 유지라고 해서 특별한 것은 없습니다. 객체의 메모리에 값을 한번 셋팅한 후 계속해서 셋팅된 값을 이용해서 멤버 메서드를 호출할 수 있다는 측면에서 상태유지라는 단어를 사용한 것입니다.

다음은 위에서 선언한 멤버 변수와 멤버 메서드의 목록입니다.

▣ UnitedData의 멤버 변수

◈ public int x;

◈ public int y;

▣ UnitedData의 멤버 메서드

◈ public int plus(){...}

◈ public int minus(){...}
◈ public int divide(){...}
◈ public int mul(){...}

UnitedData 데이터 타입을 만들었다면 사용해 봐야겠죠. 다음은 UnitedData 클래스를 테스트하는 예입니다.

§3-7 UnitedDataMain.java

```
01:  /**
02:  UnitedData 클래스를 테스트하는 예
03:  **/
04:  public class UnitedDataMain{
05:     public static void main(String[] args){
06:        UnitedData d = new UnitedData(); //UnitedData 객체 생성
07:        d.x = 5; //멤버 변수에 값할당
08:        d.y = 10; //멤버 변수에 값할당
09:
10:        System.out.println(d.minus()); //멤버변수를 이용한 minus() 메서드 호출
11:        System.out.println(d.plus()); //멤버변수를 이용한 plus() 메서드 호출
12:        System.out.println(d.divide()); //멤버변수를 이용한 divide() 메서드 호출
13:        System.out.println(d.mul()); //멤버변수를 이용한 mul() 메서드 호출
14:     } //end of main
15:  } //end of UnitedDataMain class
```

```
C:\javasrc\chap03>javac UnitedDataMain.java
C:\javasrc\chap03>java UnitedDataMain
-5
15
0
50
```

UnitedData 데이터 타입의 변수를 만들고 메모리를 생성하고 있습니다. 그리고 데이터를 직접할당의 원리로 멤버 변수에 값을 할당하고 있습니다.

▣ UnitedData의 객체 생성과 값할당
◈ UnitedData d = new UnitedData();
◈ d.x = 5;
◈ d.y = 10;

그리고 메서드를 호출한 후 결과를 출력하고 있습니다. 메서드를 호출하더라도 매개변수를 넘겨주는 것이 아니라 이미 내부의 메모리(멤버 변수)에 할당된 값을 이용해서 메서드를 호출하고 있습니다.

▣ UnitedData의 멤버 메서드 호출
◈ d.minus()
◈ d.plus()
◈ d.divide()
◈ d.mul()

이것은 앞에서 배웠던 메서드와 데이터가 분리되어 있을 때와 결합되어 있을 때의 차이점을 잘 보여주고 있습니다. 어떠한 차이가 있는지 좀 더 자세히 알아보도록 하죠.

3.4.4 변수와 메서드의 분리와 결합

객체지향이 존재하지 않던 전통적인 프로그램에서는 변수와 메서드는 분리되어 있었습니다. 객체지향의 의미가 등장하면서 결합시키는 방법이 나왔기 때문에 분리되어 있는 것은 당연한 것이었습니다. SeparatedData 클래스의 경우에는 메서드를 호출할 때마다 데이터를 넘겨주어야 하지만, UnitedData 클래스의 경우에는 메서드의 호출에 필요한 데이터를 클래스 내부에 포함하고 있기 때문에 메서드를 호출할 때마다 매개변수를 넣어줄 필요가 없습니다.

SeparatedData 클래스의 메서드들은 호출할 때마다 데이터를 넘겨주고 호출했습니다. 단순한 개념이지만 이것은 다음과 같은 문제점을 가지고 있습니다.

▣ 데이터와 메서드가 분리되어 있을 때의 문제점
◈ 메서드를 호출할 때마다 데이터를 매개변수로 넘겨주어야 한다.
◈ 특정 데이터를 사용하는 메서드가 몇 개인지 알 수 없다.
◈ 특정 데이터를 사용하는 메서드를 묶어서 관리할 수 없다.

실제 예를 들어보죠. 다음과 같이 도면이 한 장 있다고 가정하죠. 그리고 이 도면에 그리기 작업을 하기 위해서 선그리기(), 원그리기(), 사각형그리기()와 같은 메서드를 사용한다고 생각하죠. 그렇다면 다음과 같은 형식으로 프로그램해야 할 것입니다.

▣ 데이터
◈ 도면 r;

▣ 메서드
◈ void 선그리기(도면 w){
◈ 　　　//선그리기 작업
◈ }
◈ void 원그리기(도면 w){
◈ 　　　//원그리는 작업
◈ }
◈ void 사각형그리기(도면 w){
◈ 　　　//사각형그리는 작업
◈ }

위의 데이터와 메서드가 분리되어 있다고 가정한다면 다음과 같이 메서드를 호출해야 할 것입니다. 두 개의 선과 두 개의 원 그리고 하나의 사각형을 도면에 그리는 방법은 다음과 같습니다.

▣ 데이터와 메서드가 분리된 상태에서 메서드의 호출
◈ 도면 r;
◈ 선그리기(r);
◈ 선그리기(r);
◈ 원그리기(r);
◈ 원그리기(r);
◈ 사각형그리기(r);

불행하게도 계속해서 도면 r을 메서드의 매개변수로 넘겨준 후에 메서드를 호출해야만 합니다. 이것은 사용자가 도면 r을 계속 기억하고 있으면서, r을 상대로 뭔가를 그리기 위한 메서드를 찾아서 사용해야 한다는 단점을 가지고 있습니다. 즉 도면 r이라는 메모리와 도면과 관련된 메서드들을 분리해서 관리해야 한다는 단점을 가지고 있습니다. 이러한 문제점을 객체지향에서는 클래스가 해결하고 있습니다.

데이터와 메서드가 결합된 형태라고 생각하면 쉽겠죠. 만약 도면과 사용할 메서드들이 결합된 형태라면 전혀 다른 결과를 가져오게 됩니다. 개략적으로 위의 도면 그리기를 WhiteBoard라는 클래스로 작성해 보도록 하죠. 그 구조는 다음과 같습니다.

▣ WhiteBoard 클래스
◈ public class WhiteBoard{
◈ public 도면 r;
◈ public void 선그리기(){...}
◈ public void 원그리기(){...}
◈ public void 사각형그리기(){...}
◈ }

위의 WhiteBoard 클래스를 이용하기 위해서는 다음과 같이 프로그램할 것입니다

▣ WhiteBoard 클래스를 이용한 그리기 작업
◈ WhiteBoard h = new WhiteBoard();
◈ h.선그리기();
◈ h.선그리기();
◈ h.원그리기();
◈ h.원그리기();
◈ h.사각형그리기();

어떻습니까! 느낌이 다르지 않습니까! 여러분이 알고 있는 C 언어는 데이터와 메서드가 분리된 형태의 프로그램 기법이었습니다. 하지만 이제 객체지향 계열의 자바를 배우시면 데이터와 메서드를 결합시켜 관리할 수 있는 것입니다.

▣ 프로그램의 이면(裏面)

◈ 객체지향적인 설계가 무조건 좋은 것은 아닙니다. 앞에서 만들었던 계산기 프로그램과 같은 경우에 어떤 방식의 프로그램 기법이 좋은지 한번 생각해 보시기 바랍니다. 두 수를 주고 4가지의 경우를 계산하는 사람은 없습니다. 그럴 경우에는 데이터와 변수를 묶을 필요가 없습니다. 단순히 메서드의 집합으로서의 클래스를 사용하는 것이 더 옳을 것입니다. 멤버 변수를 둘 필요가 없는 곳에 멤버 변수를 두는 것 또한 비효율적인 프로그램 기법입니다. 메서드로만 사용해야 할 경우와 데이터+메서드(변수+메서드)를 함께 사용해야 할 경우를 생각하면서 클래스를 디자인해야 합니다.

3.4.5 결론

메서드가 클래스에 포함될 수 있다는 사실은 프로그래밍 언어의 역사에서 혁명적인 사건입니다. 아주 단순해 보이지만 언어의 역사가 그러합니다. 앞에서 우리는 다음과 같은 원리를 배운 적이 있습니다.

▣ 클래스의 구성

◈ 클래스 = 데이터 + 메서드

◈ 클래스 = 구조체 + 메서드

왜 클래스를 설명할 때 데이터와 메서드의 결합이라는 측면에서 논하는지 아셨을 것입니다. 여러분이 앞으로 프로그램할 때 이러한 관계를 생각하면서 클래스를 디자인해야 합니다. 멤버 변수는 상태 유지의 개념을 가지고 있으며, 그 상태를 바꿀 수 있는 역할을 멤버 메서드가 하는 것입니다. 단순히 생각해 보면 다음과 같은 논리가 성립됩니다.

◈ 멤버 메서드에서 사용하지 않는 멤버 변수는 클래스 내에 넣을 필요가 없다.

'메서드' 와 '매개변수' 그리고 '메서드의 클래스 삽입'에 대해서 배워 보았습니다. 다음에는 클래스의 접근에 관한 문제를 학습하도록 하겠습니다. 지금까지 public이라는 것을 많이 보았을 것입니다. 별다른 설명 없이 여기까지 왔지만 메서드를 배우고 나면 제일 먼저 객체지향에서 논하는 주제가 바로 private과 public입니다. 메서드의 학습 없이 private과 public을 논할 수 없기 때문에 지금까지 미루어 왔습니다. 메서드의 기초적인 개념을 배웠으니, 이제 접근제어에 대해서 알아보도록 하죠.

3.5 접근제어와 메서드

3.5.1 접근제어란?

접근제어란 새로운 데이터 타입을 만들고, 그 데이터 타입으로 객체를 선언한 후 객체 내의 멤버 변수에 값을 할당할 때, 값을 직접 할당할 수 있는가 없는가를 결정하는 접근지정자(Access

Identifier)를 말합니다. 구조체의 경우 모든 멤버에 대해 직접할당을 원칙으로 합니다. 하지만 클래스를 이용해서 만든 새로운 데이터 타입일 경우에는 이것을 private과 public으로 제어할 수 있습니다. private인 경우에는 값을 직접 할당할 수 없습니다. 이 말은 멤버의 접근이 private이면 점(.)찍고 접근할 수 없다는 의미입니다. 당연히 public인 경우에만 값을 직접 할당할 수 있습니다.

보통 이러한 접근제어를 자료의 은폐화(Encapsulation)라고 표현합니다. 이 은폐화는 아래와 같이 두 가지 측면에서 생각해 볼 수 있습니다.

▣ 접근제어의 종류
◈ 객체를 이용한 멤버의 접근제어
◈ 상속관계에서 상위 클래스와 하위 클래스간의 접근제어

이 두 접근제어에 대한 분류는 미세한 차이를 보이고 있습니다. 첫 번째 접근제어는 이 절에서 자세하게 다루게 되며, 두 번째 접근제어는 상속을 학습할 때 만나게 될 것입니다.

▣ 접근제어란?
◈ 메모리를 보유한 객체를 이용해서 멤버에 점(.)찍고 접근할 수 있는지 없는 지를 결정한다.

여기서 논의되는 접근제어는 객체를 생성하고 메모리를 할당한 후이며, 상속관계의 접근제어는 상속관계에서 클래스를 만들 때 다시 논의하도록 하겠습니다. 여기서는 객체를 이용한 접근제어에 대해서만 알아보겠습니다.

3.5.2 private의 접근과 컴파일 에러

클래스를 이용해서 변수를 생성하고 메모리를 할당한 후 점(.)을 이용해서 값을 할당하거나 메서드를 호출했습니다. 지금까지는 모든 멤버들은 점(.)을 찍고 값을 할당하거나 메서드를 호출했습니다. 그것은 멤버 변수와 메서드를 선언할 때 public 접근지정자를 사용했기 때문에 가능했던 것입니다. 접근지정자는 전체 3가지 종류가 있습니다.

▣ 접근지정자(Access Identifier)의 종류
◈ private
◈ public
◈ protected

이 절에서는 private과 public에 대해서만 배울 것이며, 상속을 배우고 난 후 protected에 대해서 학습하도록 하겠습니다. 객체의 메모리를 생성한 후 해당 객체의 멤버에 접근할 수 있으면 public으로 접근설정이 되어있는 것입니다. 만약 접근설정이 private으로 되어 있으면 점(.)을 찍고 접근할 수 없습니다.

▣ 접근 제한의 문제
◈ 여기서 논의되는 접근 제한은 메모리가 생성된 객체를 이용해서 점(.)을 찍고 접근할 수 있느냐

없느냐의 문제이다. 멤버 변수나 메서드가 public으로 설정되어 있으면, 메모리를 보유한 객체에서 점(.)을 찍고 접근할 수 있다. 그러나 private으로 설정되어 있으면 점(.)을 찍고 접근할 수 없다.

▣ private 멤버의 접근
◈ private 멤버에 직접 접근할 수 없다.
◈ private 멤버에 접근하기 위해서 public 메서드를 이용한다.

private으로 설정되어 있는 멤버에 직접 접근한다면 에러가 발생할 것입니다. private 멤버에 직접 접근할 수 없기 때문에 public 메서드를 이용하게 됩니다. 먼저 private에 접근했을 때 에러가 발생하는 예제부터 살펴보기로 하죠.

§3-8 Person.java

```
1:  /**
2:  private 멤버 변수를 포함한 클래스
3:  **/
4:  public class Person {
5:     public int age; //public 멤버 변수 선언
6:     public float height; //public 멤버 변수 선언
7:     private float weight; //private 멤버 변수 선언
8:  } //end of class
```

```
C:\javasrc\chap03>javac Person.java
```

Person 클래스는 3개의 멤버를 포함하고 있으며, 2개는 public 접근 제한으로, 나머지 하나는 private 접근 제한으로 설정되어 있습니다. 접근제한자를 사용하실 때는 데이터 타입 앞에 명시하시면 됩니다. 다음으로 Person의 private 멤버에 접근했을 때 에러가 발생하는 경우를 테스트하는 예제입니다. 아래의 예제는 에러가 나는 것을 확인하기 위한 예제이기 때문에 올바르게 동작하지 않습니다.

§3-9 PrivateAccessMain.java

```
01:  /**
02:  private 에 직접 접근하기 때문에 에러가 발생하는 예
03:  **/
04:  public class PrivateAccessMain{
05:     public static void main(String[] args){
06:        Person brother = new Person(); //객체 생성
07:        brother.age =100; //public 멤버 접근
08:        brother.height = 170.0F; //public 멤버 접근
09:        brother.weight = 67.0F; //private 멤버 접근 - 에러
10:        System.out.println("age:" + brother.age); //public 멤버 접근
11:        System.out.println("height:" + brother.height); //public 멤버 접근
12:        System.out.println("weight:" + brother.weight); //private 멤버 접근 - 에러
13:     } //end of main
```

```
14:   } //end of PrivateAccessMain class
```

```
C:\javasrc\chap03>javac PrivateAccessMain.java
PrivateAccessMain.java:9: weight has private access in Person
brother.weight = 67.0F; //private 멤버 접근 - 에러
^
PrivateAccessMain.java:12: weight has private access in Person
System.out.println("weight:" + brother.weight);
            //private 멤버접근 - 에러
^
2 errors
```

이 예제에서 우리는 다음과 같은 에러를 만나게 됩니다. 이것은 접근제어를 위반했기 때문에 발생하는 컴파일 차원의 에러입니다.

◈ PrivateAccessMain.java:9: weight has private access in Person

여기서 주의 깊게 보아야 할 것은 점(.)으로 접근하는 순간입니다. 언제 접근하고 있습니까? 위에서 보는 바와 같이 객체의 메모리가 생성된 후에 접근하고 있습니다. 접근제어의 기본법칙은 메모리가 생성된 후의 접근을 말합니다. 하나의 데이터 타입을 만들기 위해서 우리는 클래스를 디자인합니다. 그리고 만들어진 클래스로 객체를 선언하고, 그 후에 메모리를 생성합니다. 그 다음 순간에야 비로소 접근할 수 있다는 것을 잊어서는 안됩니다. 객체의 이름을 이용해서 private으로 지정된 멤버에 접근한다면 컴파일할 때 접근 에러가 발생합니다.

▣ 접근제어의 시기
◈ 메모리가 생성된 후 점(.)찍고 접근

▣ 접근제어의 두 가지 측면
◈ new 연산자를 이용해서 메모리를 생성한 후의 접근제어
◈ 상속관계에서의 접근제어

만들어진 클래스를 사용할 때와 클래스를 디자인하고(만들고) 있을 때는 반드시 구분해야 합니다. 클래스를 사용한다는 측면에서 본다면 접근제어는 만들어진 데이터 타입을 사용할 때의 문제입니다. 이것은 객체의 메모리를 생성한 후 점(.)찍고 사용할 수 있느냐 없느냐의 문제입니다. 하지만 클래스를 디자인하고 있을 때는 약간 다릅니다. 클래스를 디자인할 때에는 클래스 내부에서 작업이 이루어지기 때문에 상위 클래스와 하위 클래스 사이의 접근제어를 의미합니다. 결론적으로 클래스를 디자인할 때의 접근제어는 상속에서의 접근제어를 의미합니다.

▣ 접근제어
◈ 클래스를 이용해서 객체를 생성한 후의 접근과 클래스 안쪽에서 상위 클래스의 멤버를 사용하기 위한 접근은 정확하게 구분하고 넘어가야 한다. 현재 여기서 다루는 접근제어는 만들어진 클래스를 사용할 때의 접근제어를 말하는 것이다.

예를 들어 인간이라는 데이터 타입은 이름과 인간의 육체가 주어지는 순간, 하나의 개체로써 활동할 수 있습니다. 이와 마찬가지로 클래스의 모양을 다 만들고 나면, new 연산자를 이용해서 메모리를 생성한 후 사용하는 것이 맞겠죠. 그리고 객체의 내부는 점(.)으로 접근할 수 있지만 접근할 수 있는 부분과 없는 부분이 있습니다. 이것을 제어하는 것을 접근제어라고 합니다. 이것을 구분하는 기준은 데이터 타입 앞에 private과 public을 붙이는 것입니다.

private 멤버에 점(.)찍고 접근할 수 없다고만 했을 뿐 사실상의 접근제어의 설명은 하지 않았습니다. 다음과 같은 질문을 던져 보죠.

▣ 질문
◈ 어떻게 private 멤버에 접근을 할 수 있을까요?

이 문제의 해답은 'public 메서드를 이용한다'입니다. 어떻게 메서드를 이용해서 private 멤버 변수에 접근하며, 어떻게 private 멤버 변수의 값을 외부로 전달하는 지에 대해서 배워 보도록 하죠. 기본원리는 외부로부터 들어오는 데이터를 매개변수(Parameter)를 통해서 전달받고, 매개변수의 데이터를 내부의 멤버 변수에 전달해 주는 방법을 사용합니다. 밖으로 내보낼 때는 메서드의 리턴(Return)을 통해서 외부로 전달하고 있습니다.

3.5.3 private에 접근하는 방법

class라는 키워드로 새로운 데이터 타입을 하나 만드는 과정에서 멤버에 private 접근지정자를 사용해 보았습니다. private으로 설정되어 있는 멤버 필드에 점(.)찍고 접근할 때 컴파일도 되지 않는 황당한 에러를 경험했습니다. 이 때 '어떻게 접근할까요?'가 현재의 주제입니다.

▣ private 멤버에 접근하는 방법
◈ public 멤버 메서드의 매개변수(Parameter)를 통해서 private 멤버에 값을 할당
◈ public 멤버 메서드의 리턴(Return)을 통해서 private 멤버의 값 내보내기
◈ 이 때 매개변수와 리턴값이 할당되는 원리는 값복사의 기법을 이용한다.

해결책은 public 메서드를 통해서 외부에서 들어오는 데이터를 받아내고, 또 다른 public 메서드를 통해서 외부로 값을 내보내는 방법입니다. 즉 메서드의 매개변수(Parameter)와 리턴(Return)을 이용하는 것입니다. 이 때 메서드는 public 접근으로 설정되어 있어야 합니다. 우선, private 멤버에 접근하는 예제를 살펴보죠.

§3-10 TopSecret.java

```
01:  /**
02:  public 메서드를 이용한 private 멤버 변수의 접근
03:  **/
04:  public class TopSecret{
05:      private int secret; //private 멤버 변수 선언
06:      //private 멤버에 값 할당하기
```

```
07:     public void setSecret(int x){ //private에 접근하는 public 멤버 메서드
08:        secret = x;
09:     }
10:     //private 멤버의 값을 외부로 내보내기
11:     public int getSecret(){ //private에 접근하는 public 멤버 메서드
12:        return secret;
13:     }
14:  } //end of TopSecret class
```

```
C:\javasrc\chap03>javac TopSecret.java
```

TopSecret 클래스는 private 멤버 변수를 선언하고, private 멤버 변수를 사용하는 2개의 멤버 메서드를 구현하고 있습니다.

▣ TopSecret의 private 멤버 변수

◈ private int secret;

▣ TopSecret의 public 멤버 메서드

◈ public void setSecret(int x){...}

◈ public int getSecret(){...}

setSecret() 멤버 메서드는 데이터 타입이 void형입니다. 이것은 setSecret() 메서드가 일만하고 리턴을 하지 않기 때문에 void형으로 선언된 것입니다. 그리고 하나의 매개변수를 외부로부터 받을 것입니다. 그것은 setSecret()의 매개변수 x에 들어가겠죠. 다음 것을 보죠. getSecret() 메서드는 매개변수가 없습니다. 매개변수를 주지 않아도 일을 하지만, 이 메서드는 내부에서 뭔가를 리턴하고 있습니다.

▣ 메서드의 이름

◈ 메서드의 이름은 제 마음대로 주었는데, 처음 배울 때는 주고 싶은 이름을 주십시오. 나중에는 좋은 이름을 주시고요. 그리고 약간 배우고 나면 일정한 규칙에 의해서 이름을 준답니다. 보통의 경우 값을 할당하는 메서드는 setXxx(), 값을 리턴하는 메서드는 getXxx()라고 붙여줍니다. set과 get이라는 단어와 멤버 변수의 이름을 합쳐서 사용하며, 새로운 단어가 시작되면 대문자로 시작하게 됩니다.

내부의 멤버끼리는 private과 public을 구분하지 않기 때문에 멤버 메서드 내에서 private 멤버를 사용하고 있습니다. 그리고 메서드의 매개변수와 리턴이라는 특수한 기능을 이용해서 private에 접근하고 있습니다. 다음은 외부에서 들어오는 값을 매개변수로 받아서 private 멤버 변수에 할당하는 원리를 보여주고 있습니다.

▣ 매개변수의 값을 멤버 변수로 값복사

◈ public void setSecret(int x){

◈   secret = x;

◈ }

그리고 private 멤버의 값을 리턴이라는 것을 통해서 외부로 튕겨내고 있습니다.

▣ return을 이용해서 외부로 값을 노출
◈ public int getSecret(){
◈ return secret;
◈ }

물론 이것의 원리는 앞에서 배운 값복사의 원리에 의해서 동작합니다. 그리고 public 메서드가 private에 접근할 수 있는 유일한 방법이라는 것도 기억해 두십시오. TopSecret 클래스를 테스트하는 예를 만들어 보죠.

§3-11 TopSecretMain.java

```
01:  /**
02:  TopSecret 클래스를 테스트하는 예
03:  **/
04:  public class TopSecretMain {
05:     public static void main(String[] args){
06:        TopSecret t = new TopSecret();
07:        t.setSecret(1000); //private 멤버 변수에 값을 할당하는 메서드
08:        int s = t.getSecret(); //private 멤버 변수의 값을 얻어오는 메서드
09:
10:        System.out.println("s의 값은: " + s); //s의 값 출력
11:        System.out.println("t.getSecret(): " + t.getSecret()); //t.getSecret()의
값 출력
12:     } //end of main
13:  } //end of TopSecretMain class
```

```
C:\javasrc\chap03>javac TopSecretMain.java
C:\javasrc\chap03>java TopSecretMain
s의 값은: 1000
t.getSecret(): 1000
```

private 멤버 변수에 점(.)찍고 직접 접근한다면 컴파일도 되지 않겠죠. 당연한 에러입니다. 그래서 public 메서드를 통해서 접근하는 것입니다. 객체를 생성한 후 private 멤버에 접근할 수 없기 때문에 public 메서드를 이용해서 private 멤버 변수에 값을 할당하는 것입니다.

▣ public 메서드를 통해서 private 멤버 변수에 값을 할당
◈ TopSecret t = new TopSecret();
◈ t.setSecret(1000);

그리고 private 멤버의 값을 받아내기 위해서 다시 public 메서드를 이용하고 있습니다.

▣ public 메서드를 통해서 private 멤버의 값을 얻어내기

◈ int s = t.getSecret();

private 멤버 필드에 직접 접근을 할 수 없기 때문에 외부로부터 들어오는 데이터를 매개변수(Parameter)를 통해서 내부의 멤버 변수에 전달해 주었습니다. 외부로 내보낼 때는 메서드의 리턴(Return)을 통해서 전달하였습니다. 이렇게 한다면 private 멤버 변수의 접근은 아주 우아하게 해결됩니다.

그런데 이 부분에서 다음과 같은 질문이 나올 수 있을 겁니다.

▣ 질문 I

◈ 어떻게 public 멤버 메서드는 private 멤버 변수를 직접 사용할 수 있습니까?

이것에 대한 단순한 대답이 있습니다. 일단 클래스 내의 멤버끼리는 private인지 public인지를 따지지 않습니다. 그리고 멤버끼리는 공유가 가능합니다. 그래서, 이런 말을 한 적이 있습니다. 'private과 public의 접근제어는 점(.)찍고 난 후의 접근 문제이다'라는 말 기억하시는지요. 즉 private과 public은 객체를 이용해서 점(.)찍고 접근할 때의 문제이지, 내부의 멤버끼리 사용할 때의 문제가 아닙니다.

▣ 해답

◈ 클래스 내부의 멤버끼리는 private과 public을 따지지 않는다.

◈ 멤버끼리는 공유가 가능하다.

그리고 다음과 같은 질문도 생각해 볼 수 있습니다.

▣ 질문 II

◈ private에 접근하는 방법이 public 메서드밖에 없을까요?

답은 없습니다. 이 방법밖에는 존재하지 않습니다. private은 public 메서드가 아니면 어떠한 경우에도 접근이 불가능합니다. C++를 배우신 분들이라면 friend와 같은 private의 벽을 허무는 것이 있지 않느냐라고 반문하겠지만, 자바에서는 프렌드(friend)가 없습니다. private에 접근하는 방법은 오직 public 메서드 뿐입니다.

▣ private 멤버 변수의 접근

◈ private 멤버 필드에 직접 접근을 할 수 없기 때문에 외부로부터 들어오는 데이터를 매개변수(Parameter)를 통해서 내부의 멤버 변수에 전달하고, 외부로 내보낼 때는 메서드의 리턴(Return)을 통해서 전달한다.

◈ private에 접근할 수 있는 방법은 public 메서드밖에는 존재하지 않는다.

◈ 오직! public 메서드만이 private에 접근할 수 있다.

3.5.4 private의 사용 이유

private을 사용하는 이유에 대해 3가지 질문을 던지고, 질문을 해결하면서 private의 또 다른 의미에 대해서 알아보죠.

▣ private을 사용하는 이유에 대한 3가지 질문
◈ private 멤버 메서드도 있을까요?
◈ private 멤버 변수에 접근하는 방법이 public 메서드밖에는 없을까요?
◈ private 멤버 변수를 왜 사용할까요?

일단 private 멤버 메서드부터 해결을 하죠. public 멤버 메서드도 있는데 private 멤버 메서드가 없겠습니까? 당연히 있습니다. 그렇다면 점(.)찍고 private에 접근할 수 없으니 'private 메서드는 필요가 없지 않습니까?'라고 하겠죠. 맞습니다. 외부에서 사용할 때는 정말 필요 없습니다. 하지만 클래스 내부에서만 사용하기 위해서 private 메서드를 사용합니다. 앞에서 멤버끼리는 공유가 가능하다고 언급한 적이 있습니다. 멤버끼리는 private인지 public인지를 따지지 않습니다. 그렇기 때문에 내부에서만 사용하는 메서드를 private으로 만들어서 사용하면 되는 것입니다.

▣ private 멤버 메서드
◈ 해당 클래스 내부에서만 사용 가능하다.

두 번째 질문을 해결하죠. private 멤버 변수에 접근하고자 한다면 public 멤버 메서드밖에는 없을까요? 답은 없습니다. private 멤버 변수를 어떻게 하고 싶으면, public 메서드를 통하지 않고서는 방법이 없습니다.

세 번째 질문으로 넘어가죠. 이것이 가장 큰 질문입니다. private 멤버 변수를 왜 사용할까요? 객체지향 프로그래밍(Object Oriented Programming) 기법에서 처음부터 나오지만 잘 설명이 되지 않고 아주 어렵게 설명되는 부분이죠. 이것을 보통 은폐화, 캡슐화, 또는 자료 보호라고 하죠. 자료를 보호하기 위해서 public 멤버 메서드를 통해서만 private 멤버에 접근하게 하는 것입니다. private 멤버 변수를 왜 사용하는지 조금 더 자세히 알아보도록 하죠.

3.5.5 private 멤버 필드를 사용하는 이유

private 멤버 변수를 사용하는 이유에 대해서 보다 쉬운 말로 설명해 보겠습니다. 사실 이 부분은 제가 처음 객체지향 언어를 공부할 때 잘 설명되어 있는 책을 봐도 느낌이 안 오더군요. 거의 대부분의 책에서도 설명이 모호하죠. 저도 이 부분에서는 여러분의 느낌만을 바랄 뿐입니다. 느낌을 얻어 보시죠.

제일 먼저 외부의 데이터-사과가 있습니다. 사과를 private 멤버 변수인 우리의 위(胃)속으로 넣으려고 합니다. 그냥은 안됩니다. 사과를 위 속으로 그냥 집어넣다간 난리 나죠. 입을 크게 벌리고, 그리고 목구멍을 최대로 하고, 그리고 나서 밀어 넣으면, 이런 일은 있으면 안됩니다. private의 공간에는 메서드를 통해서 데이터를 걸러서 집어 넣어야 합니다. 메서드를 통해서 넣어 보겠습니다. 사과라는 데이터를 '먹다()'라는 public 메서드를 통해서 잘게 부수고, 침을 바르

고, 갈아서 목구멍을 통해서 사뿐히 삼켜야, private 위(胃)라는 공간으로 들어 갈 것입니다.

그런데 생각하기는 힘들지만 사과를 다시 받아내려고 합니다. 사과를 다시 받아낼 수 있을까요? 당연히 이것도 '소화()'라는 public 메서드를 통해서 찌꺼기만을 밑으로 내보내야겠죠. 다음은 이러한 과정을 클래스로 디자인한 것입니다.

▣ private 멤버의 역할을 증명하는 클래스
◈ class Human{
◈ private 소화기관 위;
◈ public void 먹다(음식 x){....}
◈ public 찌꺼기 소화(){...}
◈ }

정리를 해 보죠. public 메서드의 역할은 가공하지 않은 특정 데이터를 매개변수 형식으로 받아냅니다. 그리고 내부의 private 멤버 변수의 형식에 맞추기 위해서 메서드 내에서 가공할 것입니다. 적절히 가공되었다면 필요한 데이터만을 private 멤버 변수에 담을 것입니다. 역(逆)으로 데이터를 내보낼 때에도 내부에서 사용하던 데이터를 다시 사용자가 원하는 형식으로 가공해서 내보낼 수 있는 것입니다.

데이터를 넣고 빼는 사람은 내부에 무슨 일이 일어나는지 몰라도 됩니다. public 메서드에 합당한 데이터만 넣어주면 되니까요! 그리고 데이터를 받아낼 때에도 내부가 어떻게 되어있든 상관하지 않습니다. 오직! 자신이 원하는 값만을 얻으면 되니까요.

▣ private 멤버 변수의 사용
◈ 자료를 보호하기 위해서
◈ 내부적으로만 사용하기 위해서

이제까지 설명한 것을 예제로 한번 풀어보죠. 예제의 스토리는 이렇습니다. 외부에서 들어오는 4개의 정수를 받아서 합과 개수를 멤버 변수에 저장해 둡니다. 반대로 내보낼 때는 멤버 변수를 이용해서 '합/개수' 의 값을 내보도록 하겠습니다.

§3-12 MeanCalc.java

```
01:  /**
02:  은폐화(Encapsulation)를 증명하는 클래스
03:  **/
04:  public class MeanCalc {
05:     private int sum; //합을 저장하기 위한 private 멤버
06:     private int num; //값의 갯수를 저장하기 위한 private 멤버
07:     //외부로부터 들어온 값을 가공하는 public 멤버 메서드
08:     public void setValue(int w, int x, int y, int z, int n){
09:        sum = w + x + y + z;
10:        num = n;
```

```
11:     }
12:     //내부의 값을 가공해서 내보내는 public 멤버 메서드
13:     public int getMean(){
14:        return sum / num;
15:     }
16:  } //end of MeanCalc class
```

```
C:\javasrc\chap03>javac MeanCalc.java
```

MeanCalc는 외부에서 값을 할당하면 평균값을 자동으로 얻을 수 있는 역할을 담당하는 클래스입니다. setValue()의 매개변수로 데이터를 할당받아서 다음과 같은 계산에 의해서 private 멤버에 값을 할당하게 됩니다.

◈ sum = w + x + y + z;
◈ num = n;

그리고 외부에서는 평균값을 원하기 때문에 private 멤버를 다음과 같이 가공한 후 getMean() 메서드의 결과로 내보내게 됩니다.

◈ return sum / n;

즉 MeanCalc 클래스를 사용하는 사용자 입장에서는 값을 할당해도 어떠한 방식으로 내부에 저장되는지 몰라도 되며, 값을 얻어 갈 때에도 내부에서 어떻게 계산되는지 몰라도 원하는 값을 얻을 수 있는 것입니다. MeanCalc 클래스를 테스트하는 예를 작성해 보겠습니다.

§3-13 MeanCalcMain.java

```
01:  /**
02:  MeanCalc 를 테스트하는 예
03:  **/
04:  public class MeanCalcMain {
05:     public static void main(String[] args){
06:        MeanCalc m = new MeanCalc(); //객체 생성
07:        m.setValue(3, 5, 120, 40, 4); //가공해서 private 멤버에 값할당
08:        int s = m.getMean(); //private 멤버의 가공된 값 얻기
09:        System.out.println("평균=" + s); //데이터 출력
10:     } //end of main
11:  } //end of MeanCalcMain class
```

```
C:\javasrc\chap03>javac MeanCalcMain.java
C:\javasrc\chap03>java MeanCalcMain
평균=42
```

다음과 같이 MeanCalc 클래스의 객체를 선언한 후 값을 할당하고 평균값을 받아내고 있습니다.

◈ MeanCalc m = new MeanCalc(); //객체 생성
◈ m.setValue(3, 5, 120, 40, 4); //private 멤버에 가공해서 값 할당
◈ int s = m.getMean(); //private 멤버의 가공된 값 얻기

햄버거 가게에 가서 여러분이 매개변수로 돈만 넘겨주면 햄버거를 받아낼 수 있습니다. 여러분은 그 햄버거가 어떻게 만들어지는지 몰라도 됩니다. 단지 여러분이 원하는 것만 얻으면 되는 것입니다. 돈이 햄버거로 바뀌는 과정은 패스트푸드 가게 내부에서 알아서 할 것입니다. 이 단순한 원리가 바로 데이터의 은폐화(Encapsulation) 기술입니다.

3.5.6 결론

여러분이 은폐화라고 말하는 이유를 알았으면 좋겠군요. 여러 면에서 private은 아주 획기적인 것입니다. 기존에는 데이터를 정제하는 과정을 사용자가 직접 프로그램으로 제어를 해서 만들었는데, 앞으로는 하나의 클래스로 새로운 데이터 타입을 만들어 두면 언제라도 그것을 사용할 수 있는 것입니다. 이러한 측면에서 자바를 바라본다면 또 다른 면이 보이리라 생각됩니다.

3.6 메모리, 객체, 참조변수

3.6.1 객체의 메모리 생성

class라는 키워드로 우리는 새로운 데이터 타입을 하나 만들었습니다. 새로운 사용자 정의 데이터 타입으로 변수를 선언했으며, new 연산자와 생성자를 이용해서 메모리까지 만들었습니다. 메모리의 생성이란 데이터 타입에 해당하는 만큼의 메모리를 확보하는 일입니다.

▣ 클래스를 이용한 메모리의 생성
◈ 클래스 데이터 타입으로 변수를 선언하고, 컴퓨터 내의 메모리 속에 데이터 타입에 해당하는 만큼의 메모리를 확보하는 일

메모리를 확보하기 위해서 우리는 new 연산자를 사용하고 무조건적으로 생성자를 호출해야 합니다. 이러한 일련의 과정에서 일어나는 메모리의 변화에 대해서 좀 더 자세히 알아보도록 하죠.

새로운 클래스를 하나 만드는 것은 새로운 데이터 타입을 만드는 것입니다. 새로운 데이터 타입을 이용해서 변수를 만들었을 때 우리는 변수를 선언했다고 합니다. 기본 데이터 타입 변수와 구별하기 위해서 클래스를 이용한 변수를 객체 또는 객체변수라고 합니다.

▣ 객체(Object)
◈ 클래스로 변수를 만들었을 때 일반적인 기본 데이터 타입의 변수와 구별하기 위해서 객체 또는

객체변수라고 한다.

하지만 객체변수의 선언은 기존의 C 언어에서와 다른 의미를 담고 있습니다. 즉 객체의 이름을 하나 만든 것이지 아직 완전한 객체로서의 역할을 수행할 수 없습니다. 그럼, 언제 이용할 수 있는가라는 의문을 제기할 것입니다.

▣ new 연산자
◈ 객체의 메모리를 생성시켜 주는 역할을 담당

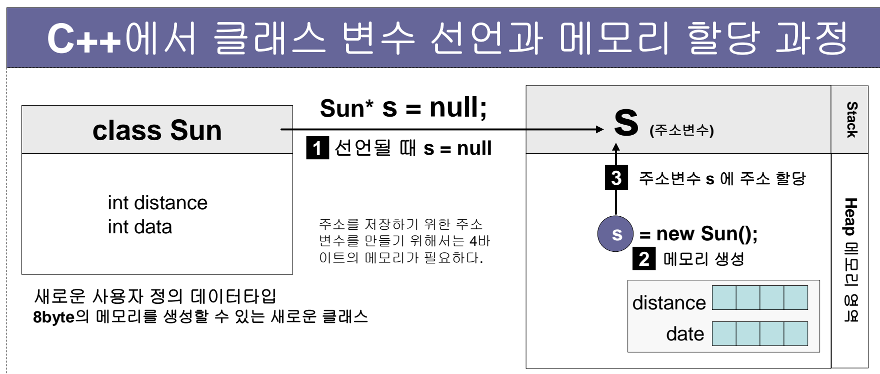
클래스는 객체변수를 선언하고 new 연산자와 함께 생성자를 호출했을 때 완전한 객체가 만들어집니다. 여기서 우리는 new 연산자가 하는 일을 다시 학습하도록 하겠습니다. new 연산자는 객체의 메모리를 생성시켜 주는 역할을 합니다. 다른 말로 바꾸면 객체변수가 제대로 된 역할을 할 수 있는 순간은 바로 객체의 메모리가 생성되었을 때입니다.

3.6.2 주소를 누가 챙기는가

new 연산자와 생성자를 이용해서 메모리를 하나 생성했다면 당연히 하나의 객체로써 사용할 수 있습니다. 하지만 new 연산자가 메모리를 생성한다고 했으며, 생성자가 호출된다고만 했지, 도대체 어떻게 메모리가 생성되어서 해당 메모리의 주소와 연결되는 지에 대해서는 설명하지 않았습니다.

C++ 언어에서는 new를 이용해서 메모리를 생성했다면 직접적으로 주소 그 자체를 넘겨줍니다. 그렇기 때문에 주소를 저장하기 위한 포인터 변수를 이용해서 주소를 관리하게 됩니다. 다음의 그림은 C++의 클래스로 메모리를 만들었을 때 메모리의 주소를 주소변수에 할당하는 예를 보여주고 있습니다.

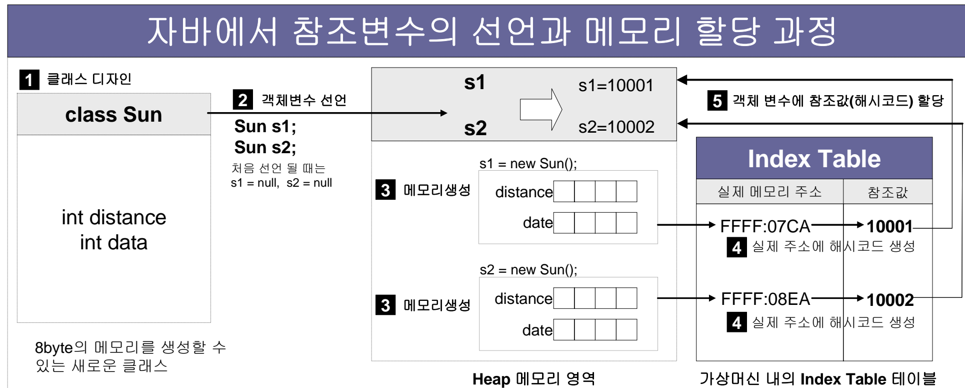
그림 3-12 C++에서 클래스 변수 선언과 메모리 할당 과정



C++에서는 사용자가 직접 주소를 핸들하게 됩니다. 그러나 자바 언어에서는 주소를 바로 주지 않습니다. 클래스로 만든 데이터 타입으로 변수와 메모리를 생성했을 때 주소를 찾아보기는 힘듭니다. 자바에서는 객체의 주소 대신 참조값이라는 것을 할당받게 됩니다. 그렇기 때문에 자바에서의 객체변수를 참조변수(Reference Variable)라고 말합니다. 다음의 그림은 자바에서 참조값

이 생성되는 순서를 보여주고 있습니다.

그림 3-13 자바에서 참조변수의 선언과 메모리 할당 과정



과정을 하나씩 따라 가보도록 하죠. 위의 참조변수의 메모리 할당의 순서대로 따라 가보도록 하겠습니다. 제일 먼저 Sun 클래스를 하나 만들도록 하겠습니다.

§ 3-14 Sun.java

```
1:  /**
2:  참조의 원리를 테스트하기 위한 클래스
3:  **/
4:  public class Sun {
5:      public int distance;
6:      public int data;
7:  } //end of Sun class
```

```
C:\javasrc\chap03>javac Sun.java
```

그리고 나서 두 번째로 해당 변수를 선언합니다. 메모리가 생성되기 전의 상태이기 때문에 null로 초기화합니다.

◈ Sun s1 = null;
◈ Sun s2 = null;

그리고 세 번째로 메모리를 생성할 것입니다.

◈ s1 = new Sun();
◈ s2 = new Sun();

네 번째로 메모리가 생성되면 내부에서 인덱스 테이블(Index Table)에서 주소를 맵핑하는 참조값을 하나씩 만들 것입니다. 마지막으로 만들어진 참조값은 다시 참조변수 s1과 s2에 할당될 것

입니다.

■ 참조값(해시코드)은 누가 만드는가?

◈ 참조값은 가상머신(Virtual Machine)에서 자동으로 생성되며, 객체를 구분하기 위한 유일한 키(Key)값이 된다.

실제 s1과 s2에 참조값이 들어 있는 지를 확인하기 위해서 s1과 s2를 출력해 보도록 하겠습니다. 다음은 위의 절차대로 객체를 생성한 후 s1과 s2 에 들어있는 참조값을 출력하는 예입니다.

§3-15 SunMain.java

```
01:  /**
02:  Sun 클래스를 이용한 참조값 출력
03:  **/
04:  public class SunMain{
05:     public static void main(String[] args){
06:        Sun s1 = null; //객체 변수 선언
07:        Sun s2 = null; //객체 변수 선언
08:        s1 = new Sun(); //메모리 할당
09:        s2 = new Sun(); //메모리 할당
10:        System.out.println("Sun의 변수 s1의 값은:" + s1); //객체 변수 출력
11:        System.out.println("Sun의 변수 s2의 값은:" + s2); //객체 변수 출력
12:     } //end of main
13:  } //end of SunMain class
```

```
C:\javasrc\chap03>javac SunMain.java
C:\javasrc\chap03>java SunMain
Sun의 변수 s1의 값은:Sun@192d342
Sun의 변수 s2의 값은:Sun@6b97fd
```

실행결과를 보시면 숫자와 같이 출력되는 것을 확인할 수 있습니다. @ 다음에 표시되는 16진수의 숫자값이 참조값입니다.

C 언어나 C++ 언어에서 메모리를 생성하는 작업과 자바에서 메모리를 생성하는 작업은 그 자체가 전혀 다른 의미를 지니고 있습니다. C++ 언어에서 new 연산자를 이용해서 메모리를 생성했다면 해당 메모리의 주소를 챙기게 됩니다. 하지만 자바는 메모리를 얻긴 얻지만 해당 객체변수가 주소를 가지지 않습니다. 주소에 연결되어 있는 참조값을 가지게 됩니다. 이런 의미에서 자바의 객체변수를 참조변수라고 하며, 참조변수에는 참조값이라는 정수값이 들어 있습니다.

3.6.3 참조변수의 특징

참조변수의 개념을 이해하는 것은 아주 [중요]합니다. 자바를 이해하는데 있어서 클래스를 이해

하는 것만큼이나 중요한 것이 바로 참조변수입니다. 클래스를 이해할 수 없으면 객체지향 프로그램을 아예 할 수도 없습니다. 그리고 클래스의 개념을 안다손 치더라도 참조변수의 개념을 깨닫지 못한다면 자바 프로그램이 상당히 난해해집니다.

▣ 참조값(Reference Value)
- ◈ 참조값이란 객체의 메모리를 생성했을 때 메모리와 연결된 유일한 숫자값을 말한다.
- ◈ 이 숫자값을 참조변수가 받으며, 자바에서는 참조값을 가지고 있으면 해당 객체를 핸들할 수 있다.
- ◈ 참조값으로 해당 객체를 핸들할 수 있다.
- ◈ 참조값으로 작업하면 내부에서 참조값에 연결된 메모리로 작업하는 것과 같은 효과가 있다.

객체를 만들면 자바를 실행시켜 주는 시스템 내에서 객체의 실제 주소에 연결된 숫자 하나를 던져 줍니다. 이 숫자를 객체가 받아 챙기게 되며, 객체는 이 숫자(참조값)를 이용해서 해당 객체의 메모리에 값을 할당하거나 메서드를 호출할 수 있습니다.

참조변수는 4바이트짜리 정수값입니다. 그렇기 때문에 객체의 메모리를 할당할 때 4바이트의 정수값을 객체변수에 할당하게 되는 것입니다. 다음과 같이 메모리 없는 객체변수를 선언했다고 가정하죠. 이것이 무엇을 의미하는지 생각해 보시기 바랍니다.

◈ Sun s1 = null;

단순히 메모리 없는 객체변수를 선언한 것일까요? 이제 참조의 의미를 배웠으니 다음과 같이 해석하셔야 합니다.

▣ 'Sun s1 = null'의 의미
- ◈ s1은 4바이트짜리 참조변수가 만들어진 것이다.
- ◈ s1 자체는 4바이트의 메모리 생성의 의미를 담고 있다.
- ◈ s1은 현재 4바이트의 메모리에 null값이 들어있다.
- ◈ s1에는 객체의 실제 메모리와 연결된 Sun형의 참조값을 넣을 수 있다.

참조변수가 만들어졌다면 다음 구문을 해석해 보시기 바랍니다.

◈ s1 = new Sun();

s1이라는 변수에는 참조값을 받을 수 있습니다. 그리고 new Sun()이라고 했을 때 참조값이 생성된 후 s1에 할당되는 것입니다. s1이라는 참조변수는 아무 것도 아닙니다. 단순히 정수형 숫자를 담을 수 있는 4바이트짜리 메모리라는 것을 잊지 마시기 바랍니다. 그렇다면 실제 참조값을 누가 만드는가?라는 질문을 할 것입니다. new를 이용해서 객체를 생성했을 때 자바 시스템 내부에서 자동으로 참조값을 만들어 줍니다. 즉 전자동이라는 의미입니다.

마지막으로 다음 구문을 해석해 보시기 바랍니다.

◈ s1.distance = 1000;

이 구문을 참조변수와 연결해서 해석해 보면, 다음과 같은 절차에 의해서 값이 객체 s1과 연결된 메모리로 할당될 것입니다.

▣ 's1.distance = 1000'의 의미
◈ s1에는 참조값이 할당되어 있다.
◈ s1의 참조값과 연결된 주소를 검색하기 위해서는 자바 시스템 내부의 인덱스 테이블을 검색한다.
◈ s1의 참조값에 연결된 주소를 인덱스 테이블에서 찾았다면 해당 메모리에 distance의 자리를 찾는다.
◈ s1의 [참조값]-[메모리]-[distance]를 찾았다면 데이터 1000을 할당한다.

이것이 내부에서 이루어지기 때문에 자바를 배우는 분들이 잘 모르고 지나칠 수도 있습니다. 여기서 정확하게 알아두시기 바랍니다.

3.6.4 참조변수끼리의 할당

참조변수의 의미를 알았으니 이번에는 참조변수의 할당의 측면에서 알아보겠습니다. 일반적인 변수의 경우 다음과 같이 값을 할당할 수 있습니다.

◈ int a = 5;
◈ int b = a;

아주 단순한 할당이죠. 참조에도 이 법칙을 적용해 보도록 하겠습니다. 먼저 여러분이 알아야 할 것은 자바에서 참조값을 만드는 방법은 new 연산자를 이용하는 방법밖에 없다는 것입니다. 참조값을 사용자가 직접 수(手)작업으로 만드는 방법은 없습니다. 가상머신의 허락을 얻어야만 참조값을 얻을 수 있습니다. 그 허락을 받는 것이 바로 new 연산자입니다.

참조값을 테스트하기 위해서 다음과 같이 단순한 클래스를 하나 만들도록 하겠습니다.

§3-16 MotorCycle.java

```
01:  /**
02:  참조를 증명하기 위한 클래스
03:  **/
04:  public class MotorCycle {
05:     private int id;
06:     private int speed;
07:     public void setData(int i, int s){
08:        id = i;
09:        speed = s;
10:     }
11:     public void drive(){
```

```
12:         System.out.println("이 오토바이의 번호판은 " + id + " 입니다.");
13:         System.out.println("오토바이는 현재 " + speed + " Km 속도로 달립니다.");
14:     }
15: } //end of MotorCycle class
```

```
C:\javasrc\chap03>javac MotorCycle.java
```

오토바이의 번호판(id)과 속도(speed)를 셋팅한 후 운전을 할 수 있는 MotoCycle 클래스를 만들어 보겠습니다. 다음과 같이 메모리가 있는 객체와 메모리가 없는 객체를 만들도록 하겠습니다.

▣ 메모리 없는 객체 m과 메모리 있는 객체 c의 생성

◈ MotorCycle m = null;

◈ MotorCycle c = new MotorCycle();

현재 m은 참조값을 저장할 수 있는 4바이트의 메모리만 확보한 상태입니다. m에 연결된 객체는 없습니다. 그리고 객체 c는 참조변수에 실제 객체가 연결되어 있습니다. 기본 데이터 타입 변수의 할당의 방법처럼 참조변수끼리 할당해 보도록 하죠.

▣ 참조변수끼리의 할당

◈ m = c;

이렇게 하면 뭐가 복사될까요? 이것은 참조값복사가 됩니다. c에는 참조값이 들어 있기 때문에 c의 참조값이 m으로 값복사되어 들어간 것입니다. 여러분은 앞에서 값복사에 대해서 배웠습니다. 두 개의 메모리가 존재하고, 한 쪽의 메모리에 들어 있는 값을 다른 쪽에 값만을 복사한다라고 배웠습니다. 현재의 'm = c'는 c에 들어 있는 정수값 즉 4바이트의 참조값을 m에 복사하는 것입니다. 결국 m과 c는 똑같은 메모리를 가리키고 있는 것입니다. 이것을 증명하는 예를 만들면 아래와 같습니다.

§3-17 MotorCycleMain.java

```
01: /**
02: 참조 값복사를 테스트하는 예
03: **/
04: public class MotorCycleMain {
05:     public static void main(String[] args){
06:         MotorCycle c = new MotorCycle(); //메모리 있는 객체 변수 선언
07:         c.setData(9872, 150);
08:         c.drive();
09:         System.out.println("객체 c : " + c); //객체 출력
10:         System.out.println(); //한줄 추가
11:
12:         MotorCycle m = c; //참조값복사
13:         m.drive(); //복사된 참조값을 이용한 메서드 호출
```

```
14:         System.out.println("객체 m : " + m); //객체 출력
15:     } //end of main
16:  } //end of MotorCycleMain class
```

```
C:\javasrc\chap03>javac MotorCycleMain.java
C:\javasrc\chap03>java MotorCycleMain
이 오토바이의 번호판은 9872 입니다.
오토바이는 현재 150 Km 속도로 달립니다.
객체 c : MotorCycle@12498b5

이 오토바이의 번호판은 9872 입니다.
오토바이는 현재 150 Km 속도로 달립니다.
객체 m : MotorCycle@12498b5
```

위의 예제에서 객체 c를 m에 참조값복사를 하고 있습니다.

◈ MotorCycle m = c; //참조값복사

참조값복사 이후에 m은 새로운 객체의 메모리를 생성한 것이 아니라 c의 참조값을 이용하고 있는 것입니다. 하나의 메모리를 두 참조변수가 가리키게 되는 것입니다. 이제 결론을 내려야겠군요.

▣ 객체변수와 참조변수

◈ 객체변수는 참조변수이며 참조변수끼리의 할당은 참조값복사가 된다. 참조값끼리 아무리 복사를 하더라도 객체 내부의 메모리끼리의 복사는 이루어지지 않는다. 단지 참조값만이 복사되어진다. 이러한 이유에서 자바에서는 값에 의한 호출(Call by Value)만 존재한다. 참조값복사도 값복사입니다!

3.6.5 참조변수는 타입이 있다.

참조변수는 보잘 것 없는 4바이트의 정수라고 했습니다. 그리고 참조값끼리 서로 할당할 수 있다고 했습니다. 여기서 우리는 한가지를 짚고 넘어갈 것이 있습니다. 그것은 1장에서 배웠던 데이터 타입의 의미입니다. 데이터 타입은 메모리의 크기와 형태를 지정하는 것이라고 배웠습니다. 아무리 객체변수가 4바이트의 참조값을 가지고 있다 하더라도 타입은 있습니다. 이 타입(형)을 무시할 수는 없습니다. 객체를 출력했을 때 무엇이 출력되었는지 생각해 보십시오.

◈ MotorCycle@16930e2

분명 16진수로 표현된 참조값과 그리고 참조값에 대한 타입이 출력되었습니다. 참조값에는 타입이 있는 것입니다. 즉 참조값끼리 할당을 하더라도 같은 타입끼리 할당할 수 있습니다. 다음의 예를 한번 보시기 바랍니다.

§3-18 RefMain.java

```
01:  /**
02:  잘못된 형식끼리의 참조값 할당을 증명하는 예제
03:  **/
04:  class Sos{
05:      //비어 있는 클래스
06:  } //end of Sos class
07:
08:  class Tot{
09:      //비어 있는 클래스
10:  } //end of Tot class
11:
12:  public class RefMain {
13:      public static void main(String[] args){
14:          Sos s = new Sos();
15:          Tot t = new Tot();
16:          //이 부분은 에러가 발생합니다.
17:          s = t; //서로 다른 타입끼리의 참조값 할당
18:      }
19:  } //end of RefMain class
```

```
C:\javasrc\chap03>javac RefMain.java
RefMain.java:17: incompatible types
found   : Tot
required: Sos
s = t; //서로 다른 타입끼리의 참조값 할당
^
1 error
```

▣ **참고**

◈ 하나의 파일에 여러 개의 클래스가 존재할 때 단 하나의 클래스만이 public 클래스가 될 수 있다. 보통 main을 포함한 클래스를 public 클래스로 둔다.

위의 예제는 컴파일되지 않습니다. 서로 다른 타입끼리의 참조값 할당을 했기 때문에 컴파일 에러가 발생하는 것입니다. 참조값이 단순히 정수값이라고 해서 참조값의 타입을 무시하면 에러를 만나게 될 것입니다.

3.6.6 결론

지금까지 참조변수에 대해서 알아보았습니다. 여러분은 객체를 생성하는 단 하나의 구문으로 상당한 시간을 투자한 것입니다. 단 한줄이지만 그만한 가치가 있었을 것입니다. 마무리하는 의미

에서 다시 정리를 해보죠.

▣ MotorCycle m = new MotorCycle();
◈ MotorCycle : 새로운 사용자 정의 데이터 타입
◈ m : MotorCycle 데이터 타입으로 선언한 참조변수(객체변수)
◈ new : 메모리를 생성하는 연산자, 참조값을 리턴한다.
◈ MotorCycle() : 메모리 생성 후 해당 메모리의 초기화 작업을 담당하는 생성자

이쯤에서 결론을 내려보도록 하겠습니다.

▣ 결론
◈ 객체변수는 참조변수이다.
◈ 객체의 이름은 참조값(Reference)이 할당되기 때문에 다른 참조값을 할당한다 하더라도 실제 객체에 연결된 내부의 메모리는 복사가 되지 않는다. 단순한 참조값에 대한 값복사가 이루어진다.
◈ 참조값만 가지고 있다면 해당 객체를 핸들할 수 있다.

객체를 복사하면 참조값만 복사되기 때문에 클론(clone)이라는 기법을 이용해서 연결된 메모리까지 복사하는 기법이 있습니다. 이것은 10장의 Object 클래스를 학습하면서 배우게 될 것입니다.

▣ 메모리 차원의 객체 복사 기법
◈ 객체의 실제 메모리를 복사하기 위해서 클론(clone)이라는 기법을 사용한다.

객체 복사를 위한 클론(clone)의 기법은 10장에서 배우게 될 것입니다. 이 절에서 나오는 객체의 할당에서 가장 중요한 부분은 바로 객체는 참조값이라는 것입니다. 주소의 참조값 즉 레퍼런스(Reference)라는 것은 할당을 하더라도 단순한 레퍼런스만 복사될 뿐 객체의 메모리는 복사되지 않습니다. 객체를 복사할 수 없기 때문에 자바에서 객체에 대한 메모리 차원의 복사를 위해서 Cloneable이라는 인터페이스를 제공해 주고 있습니다. 객체복사는 차후에 상세한 설명을 덧붙이도록 하겠습니다.

여기서는 객체끼리 할당하려고 한다면 당연히 데이터 타입이 같아야 한다는 것과 할당이 이루어질 때 참조값만이 복사된다는 것을 기억해 두시기 바랍니다.

3.7 마무리

3.7.1 결론

3장의 The Class에서는 구조체에 없는 클래스에서만 존재하는 특징들을 중심으로 클래스 자체에 대해서 알아보았습니다.

▣ **클래스에 추가된 기능 두 가지**
- ◈ 클래스에 메서드 추가
- ◈ 접근제한자

물론 이것이 클래스의 전부는 아니지만 기존의 언어와 구별되기 때문에, 그리고 아주 기초적인 클래스의 개념들이기 때문에 반드시 섭렵(涉獵)하셔야만 하는 개념입니다. 이것을 이해한다면 보다 쉽게 자바에 접근할 수 있으리라 생각됩니다. 그리고 C++의 주소 개념과 다른, 너무나도 중요한 참조의 개념을 배워 보았습니다.

이 장이 마무리된다면 여러분들은 자바의 Hello World를 출력할 자격이 주어졌다고 생각합니다. 대부분 무조건 Hello World부터 출력하지만 기초적인 배경지식 없이 자바를 접하게 된다면 오히려 혼동만 더하게 될 것입니다. 다음 장부터 클래스의 객체지향 개념을 배워보도록 하겠습니다.

▣ **윈도우의 핸들(HANDLE)과 자바의 참조(Reference)**

◈ 참고적으로 여러분이 Win32 API를 공부하시면 이 참조를 만나실 수 있습니다. Win32 API는 객체지향 프로그램이 아니라 순수한 C 언어로 만들어진 윈도우 운영체제의 시스템 라이브러리입니다. 클래스가 없던 시절 C에서는 이 참조의 기법을 이용해서 데이터를 은폐화하고 관리한 것입니다. Win32 API에서 핸들(HANDLE)이라고 소개되는 것이 바로 참조의 기법을 사용하고 있습니다. 핸들 변수는 4바이트의 정수 즉 참조변수가 되는 것입니다. 그리고 핸들의 참조값은 실제 메모리와 연결되어 있습니다. 개념상으론 윈도우의 핸들과 자바의 참조변수는 완벽하게 동일합니다. 하지만 윈도우의 핸들은 C 방식으로 구현되어 있으며, 자바는 그것을 발전시킨 객체지향 기법이라는 것이 다릅니다.