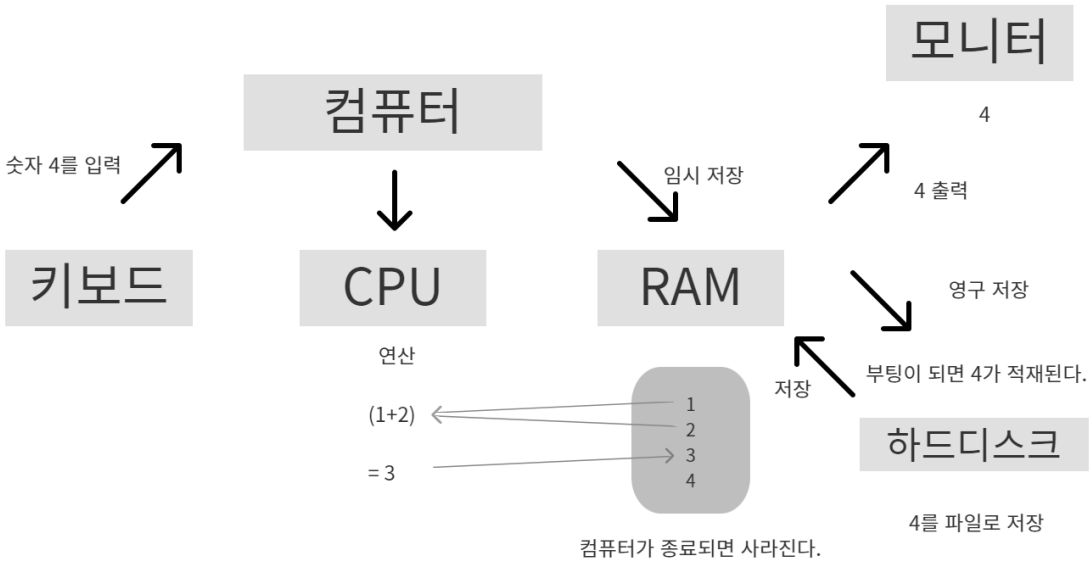
ch1. 자바 JVM 이해하기

하드웨어 : 키보드, CPU, RAM, 모니터, HDD 등



RAM은 전류를 이용하여 데이터를 저장을 한다. → 컴퓨터가 종료되면 데이터가 소멸한다.

하드디스크(HDD)는 스크래치를 내서 기록을 한다. (lp와 비슷) → 전류를 이용한 것이 아니기에 컴퓨터가 종료되어도 데이터가 살아있다. 부팅을 하면 하드 디스크에 있는 데이터를 RAM에다가 적재한다.

운영체제 : window, MAC, 우분투(리눅스)

사용자가 하드웨어에게 직접 명령을 내리기가 어렵다. (제어하기가 어렵다.)

그래서 OS를 이용하여 하드웨어를 제어한다.

Java 언어 사용

개발자가 Java라는 언어로 OS에게 명령을 내리면 OS가 하드웨어를 제어할 것이다.

이때 OS에게 명령을 내리려면 JDK(Java Delvelopment Kit)가 필요하다.

jdk.java.net에서 다운을 받을 수 있다.

이때 OS마다 다운받는 파일이 다른 데 그 이유는 OS마다 명령어가 조금씩 다르기때문이다.

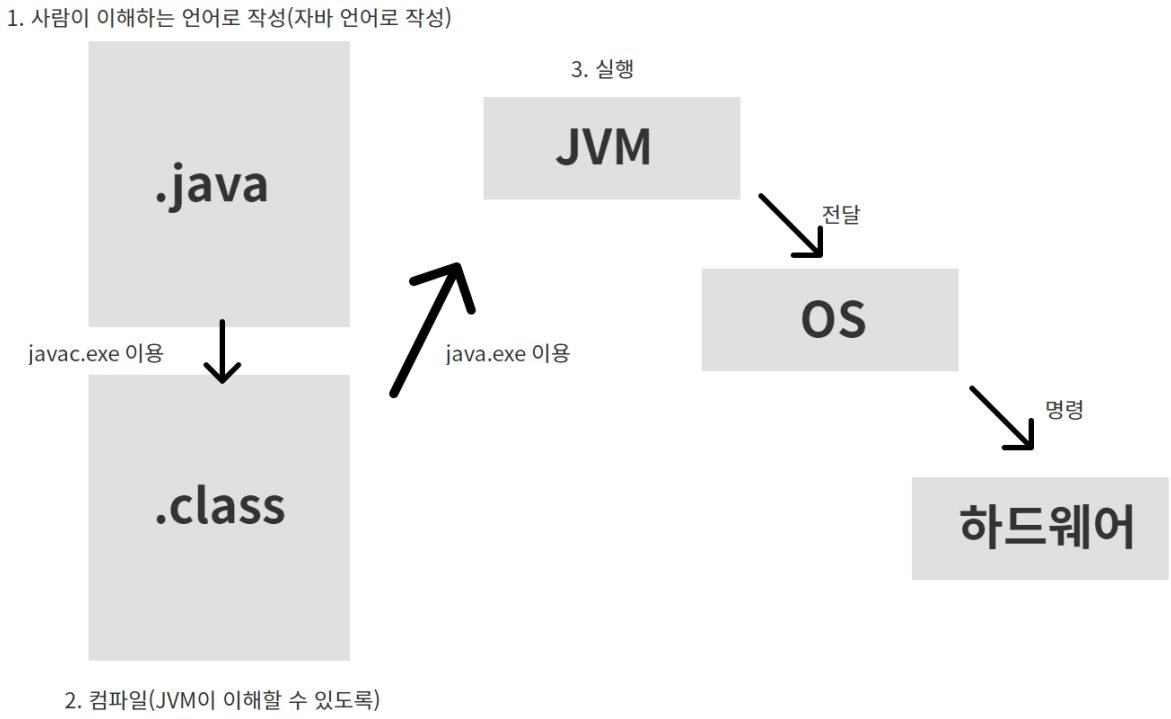
그래서 자신의 운영체제에 맞는 적합한 JDK를 다운받아야 한다.

JDK 환경변수

1. 고급시스템설정 들어가서 환경 변수 클릭
2. 시스템 변수 아래에 새로 만들기 버튼을 클릭하고 변수이름을 `JAVA_HOME`으로, 변수값을 `JDK가 설치된 경로`로 지정
3. 시스템 변수 아래에 변수 Path를 클릭하고 편집 버튼을 누른다음 새로 만들기 버튼을 누르고 %JAVA_HOME%\bin으로 설정 → bin 폴더내부의 javac.exe, java.exe 를 이용해서 Java파일이 어느경로에 있던지 컴파일 및 실행(RunTime)을 진행할 수 있다.

JVM(Java Viture Machine)

JDK를 설치하면 JVM이 설치된다.

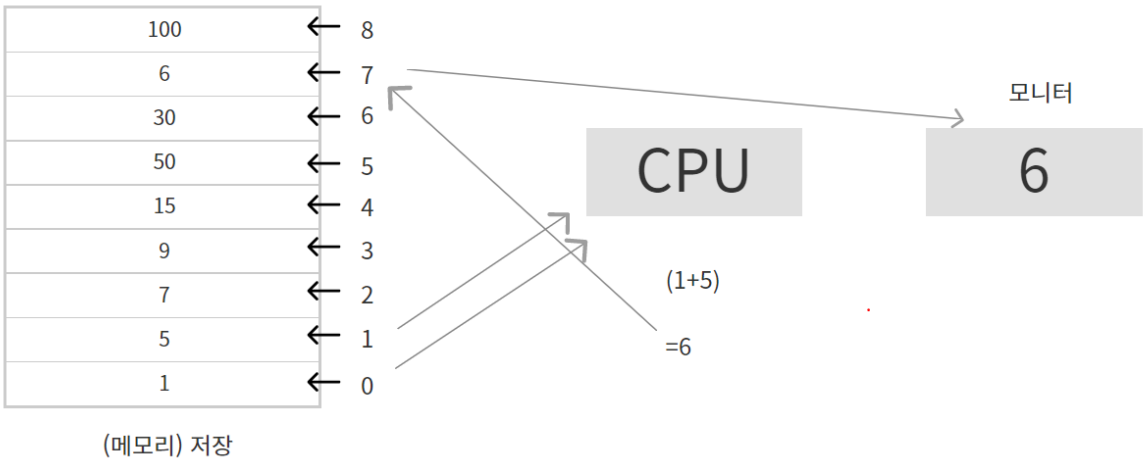


Java Source Code가 컴파일 과정을 거쳐 Java ByteCode로 변환된다.

JVM에 의해서 바이트 코드(ByteCode)를 OS가 이해할 수있는 기계어로 해석되므로 OS의 종류에 종속되지 않는다.

메모리에 관한 개념

메모리는 컴퓨터의 저장영역

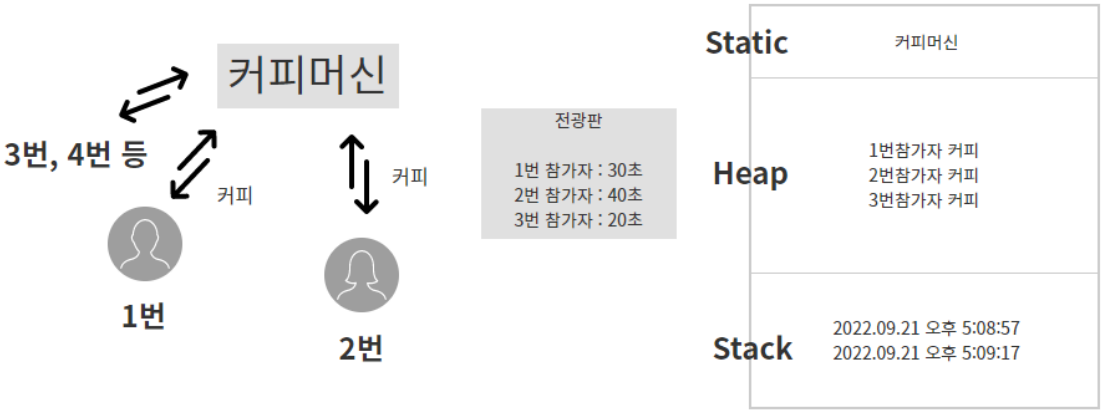


CPU가 메모리 주소 0,1의 값인 1과 5를 더하기 연산을 통해서 값을 구하면 6이다.

이를 메모리 주소7에 저장한다.

그리고 메모리 주소7을 참조해서 모니터에 숫자 6을 출력한다.

static, heap, stack 간단설명



자바로 커피 빨리 마시기 게임을 만들었다. 커피 머신에 와서 커피를 받고 마신 시간이 가장 짧은 참가자가 이기는 게임이다.

여기서 커피머신은 게임시작전부터 있어야하기때문에 static에 넣어야 한다.

static : 프로그램이 시작되기 전에 메모리에 필요한 것들을 올리고 종료될 때까지 소멸되지않고 메모리에 계속 머문다. (정적영역)

참가자는 커피 머신에 와서 커피를 받고 커피를 마시고 마신 시간을 측정한 후 사라진다. 그러므로 참가자와 커피는 heap에 넣어야 한다. 시간은 stack에 넣어야 한다.

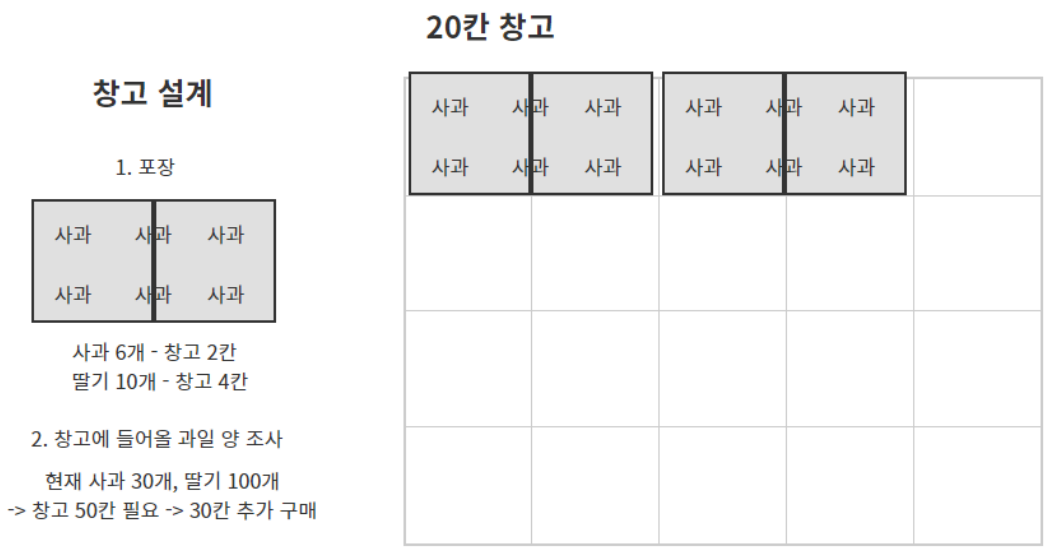
heap : 프로그램이 실행되고 있는 도중에 필요할 때마다 메모리에 올리고 일정시간이 지나면 소멸된다. (동적영역)

사용자에 의해 메모리 공간이 동적으로 할당되고 해제된다. 사용자가 직접 관리해야 하는 메모리 영역이다. 변수를 전역적으로 액세스 할 수 있다.

stack : 행위에 대한 정보가 시작할 때 메모리에 올라가고 행위가 끝나면 메모리에서 사라진다.

정적 메모리 할당, 함수의 호출과 함께 할당되며 함수의 호출과 관계되는 지역변수와 매개변수가 저장되는 영역이다. 함수의 호출이 완료되면 소멸된다.

자료형



과일장사를 하기 위해 창고를 설계하고 그 후 과일 장사를 시작할 것이다.

창고 설계를 하려면 포장을 해야되고 양을 알아야 한다.

1) 포장 → 데이터를 담을 때에는 일정크기(박스 포장) 안에 담아야 적재하기 좋고 나중에 데이터 꺼낼 때도 편리하다.

2) 창고에 들어올 양 조사 → 양을 모른다면 창고가 부족할 수도 있다.

과일장사와 마찬가지로 자바에서도 프로그램을 시작하려면 포장을 하고 데이터 양을 파악해서 설계를 해야한다. 자바에서는 포장을 어덯게 해야되는 지 미리 정해져 있다. 포장은 8가지의 자료형으로 포장할 수 있는데 그 중에서 대표적인 4가지 자료형이 있다.

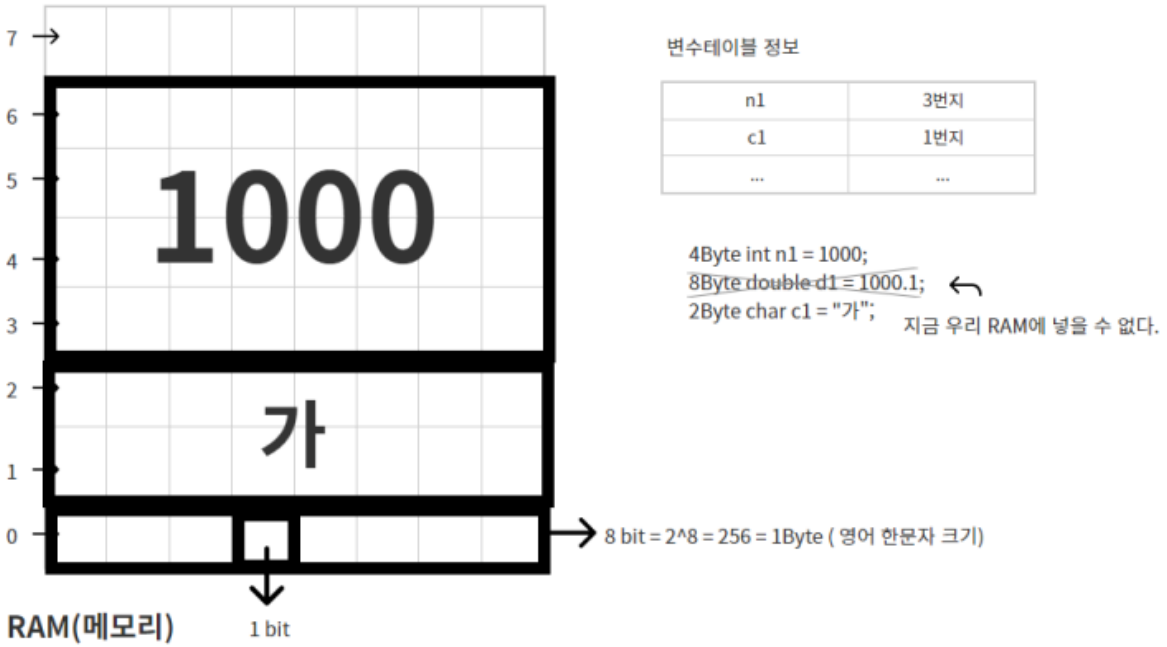
1) boolean type (1bit) : 박스1개(0 or 1) , 2^1가지 , 참or거짓

2) int type (32bit) : 박스32개 , 2^32가지 , 정수 (약 -21억 ~ 21억)

3) double type (64bit) : 박스64개 , 2^64가지 , 실수 (약 -900경 ~ 900경)

4) char type (16bit): 박스16개 , 65536가지, 문자 (-32768 ~ 32767)

자료형 메모리 구조

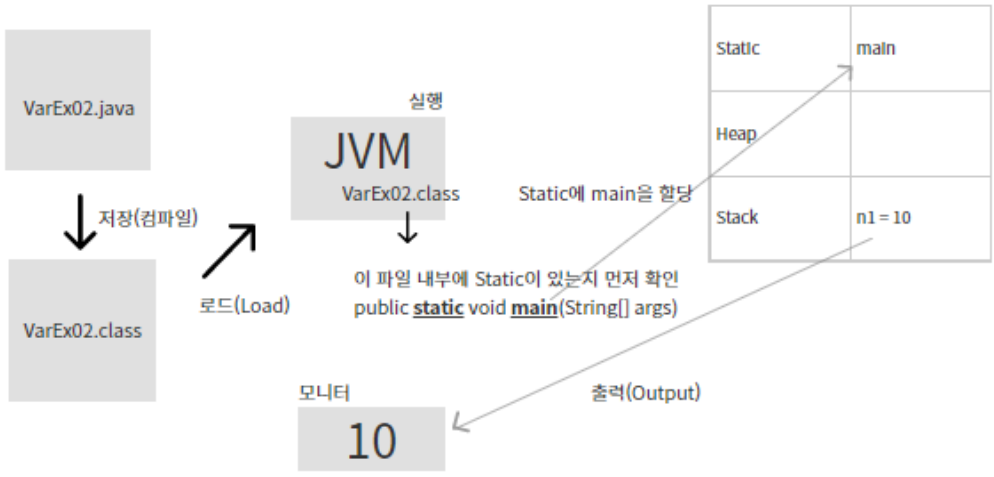


n1, c1과 같은 것을 변수라고 부른다. 개발자가 RAM에서 몇번지가 사용되지 않은 공간인지를 찾아야하는데 쉽지않으므로 변수를 지정해주면 컴퓨터가 알아서 메모리의 남는 공간을 찾고 기록을 해둔다. 메모리에 저장된 어떤 값을 변수에 저장을 하는데 이것을 값을 초기화한다라고 부른다.

공간의 값을 찾으려면 지정한 변수 이름으로 찾는다.

예를 들어 c1을 찾는다면 c1의 시작번지를 보고 char는 2Byte이니까 2칸을 읽는다.

자바코드 실행원리



1. 자바의 모든 코드는 클래스 내부에 존재해야한다.
2. 자바는 실행전에 static이라는 키워드를 가지고 있는 모든 친구들을 static 메모리 공간에 로드한다.

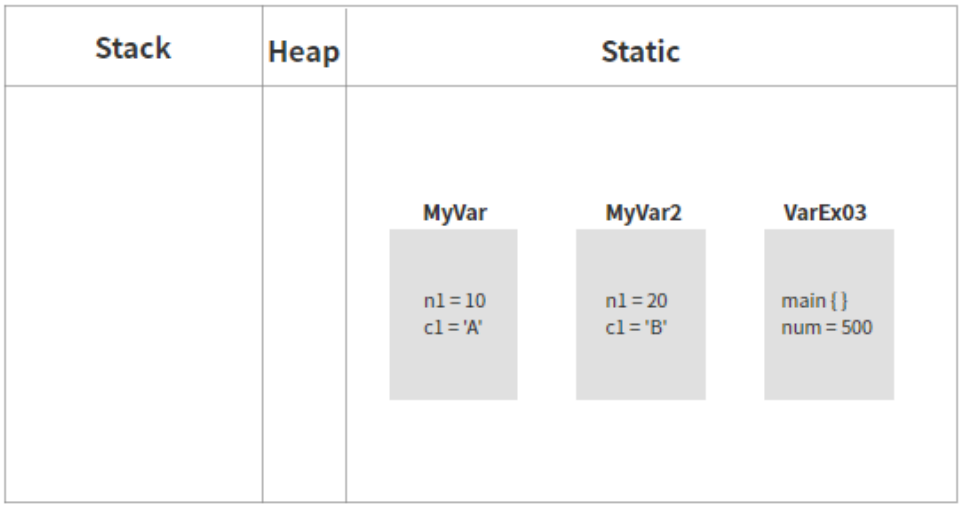
3. 자바는 실행하면 main이라는 친구의 내부를 실행하고 내부{ }가 끝나면 자바가 종료된다. → 실행하려면 main이라는 이름 자체도 메모리에 로드되어야 된다. main은 static에 로드되어있기때문에 실행된다. 내부가 실행될 때 main stack이 열린다.

Beans

여러가지 자료형을 가지고 있는 클래스를 Beans라고 부르며 개발자가 만드는 커스텀 자료형이다.

```
package ch01;

// MyVar은 클래스 자료형 = 개발자 만드는 커스텀 자료형
// 여러가지 데이터를 가지고 있는 클래스를 Beans라고 부름.
class MyVar {
  static int n1 = 10;
  static char c1 = 'A';
}

class MyVar2 {
  static int n1 = 20;
  static char c1 = 'B';
}

public class VarEx03 {

  static int num = 500;

  public static void main(String[] args) {
    System.out.println(MyVar.n1);
    System.out.println(MyVar.c1);
    System.out.println(num); // VarEx03.num인데 같은 공간에 있으니까 VarEx03 생략
    System.out.println(MyVar2.n1);
    System.out.println(MyVar2.c1);
  }

}
```

static 키워드를 가지고 있는 모든것을 static 메모리에 올린다.



static은 정적이다. 프로그램이 시작되기 전에 static 메모리에 올라고 프로그램이 종료되면 없어지기때문에 프로그램 시작 도중에는 관리할 수 없다.

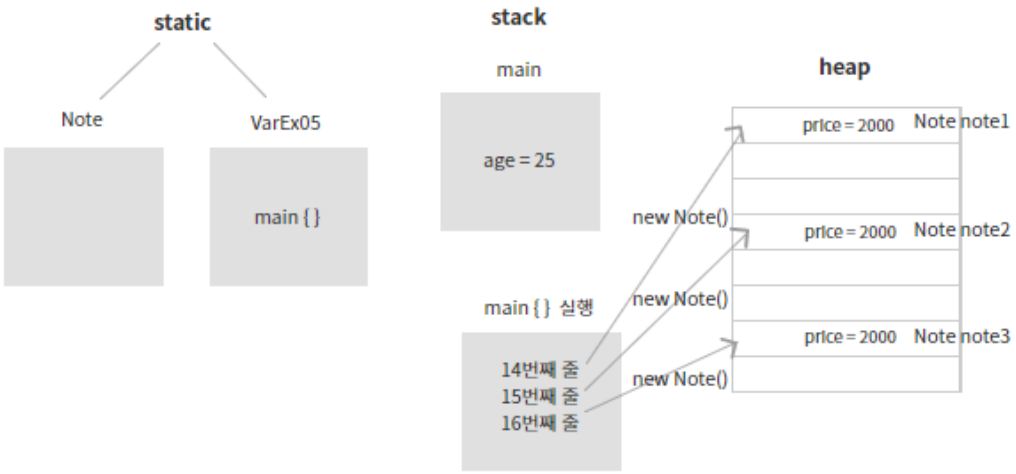
많은 데이터를 저장하고 싶으면 클래스 자료형을 여러개 만들어야한다. 계속 추가하는 식으로 일일이 한다면 불편할 것이다.

```
package ch01;


class Note1 {
  static int num = 1;
  static int time = 1015;
  static int price = 3000;
}

class Note2 {
  static int num = 2;
  static int time = 1020;
  static int price = 1000;
}

// 노트 100개가 필요? - 클래스 자료형을 100개를 만들어야 한다. (단점)


public class VarEx04 {

  public static void main(String[] args) {
    System.out.println(Note1.num);
    System.out.println(Note1.time);
    System.out.println(Note1.price);
  }

}
```

프로그램 시작 중간에 동적으로 관리가 가능할 수 있게 heap을 이용할 것이다.

```
package ch01;

// VarEx05 -> main

// Note -> X

class Note {
  int price = 2000;
}

public class VarEx05 {

  public static void main(String[] args) {
    Note note1 = new Note(); // heap 공간에 Note 클래스가 가지고 있는 모든 데이터를 할당해!! (대신 static 제외해)
    Note note2 = new Note();
    Note note3 = new Note(); // heap 공간
    int age = 25; // main stack 공간
    System.out.println(age);
    System.out.println(note1.price);
    System.out.println(note2.price);
    System.out.println(note3.price);
```

```
    note3.price = 30000;
    System.out.println(note1.price);
    System.out.println(note2.price);
    System.out.println(note3.price);
  }

}
```



클래스이름 참조변수 = new 클래스이름( );으로 정의하며 이는 heap 공간에 클래스이름( ) 내부에 가지고 있는 모든 데이터를 할당한다는 의미이고 static은 제외한다.

Note 라는 커스텀자료형 클래스의 실제 데이터(int price=2000)는 heap에 저장되고 note1, note2, note3 이라는 변수는 stack 영역의 공간에 있고 실제 데이터가 저장된 heap 영역의 참조값을 new 연산자를 통해서 return 받는다.

즉, 실제 데이터를 갖고 있는 heap 영역의 참조 값을 stack 영역의 객체가 갖고 있다.

출처 : 이지업 컨텐츠 내의 데어프로그래밍 자바강의