



JAVA 1강

양 명 속

[now4ever@naver.com]



curriculum

- Java
 - 기본 문법
 - JDBC - 데이터베이스 연결
 - GUI 윈도우 프로그래밍 (Awt, Swing)
 - 입출력 - I/O, 스레드/ 네트워크(소켓프로그래밍)
- Oracle
- Html5/CSS3, Javascript, jQuery/Ajax
- Jsp/Servlet
- Framework – Spring/mybatis
- Spring Boot
- Project
- 파이썬, 머신러닝/딥러닝



목차

- 자바 실행 환경
 - 자바언어의 역사
 - JDK 17.0 설치/ 경로 설정
- 변수/자료형
- 형변환

자바 실행 과정

C:\WINDOWS\system32\cmd.exe

Hello java!!!

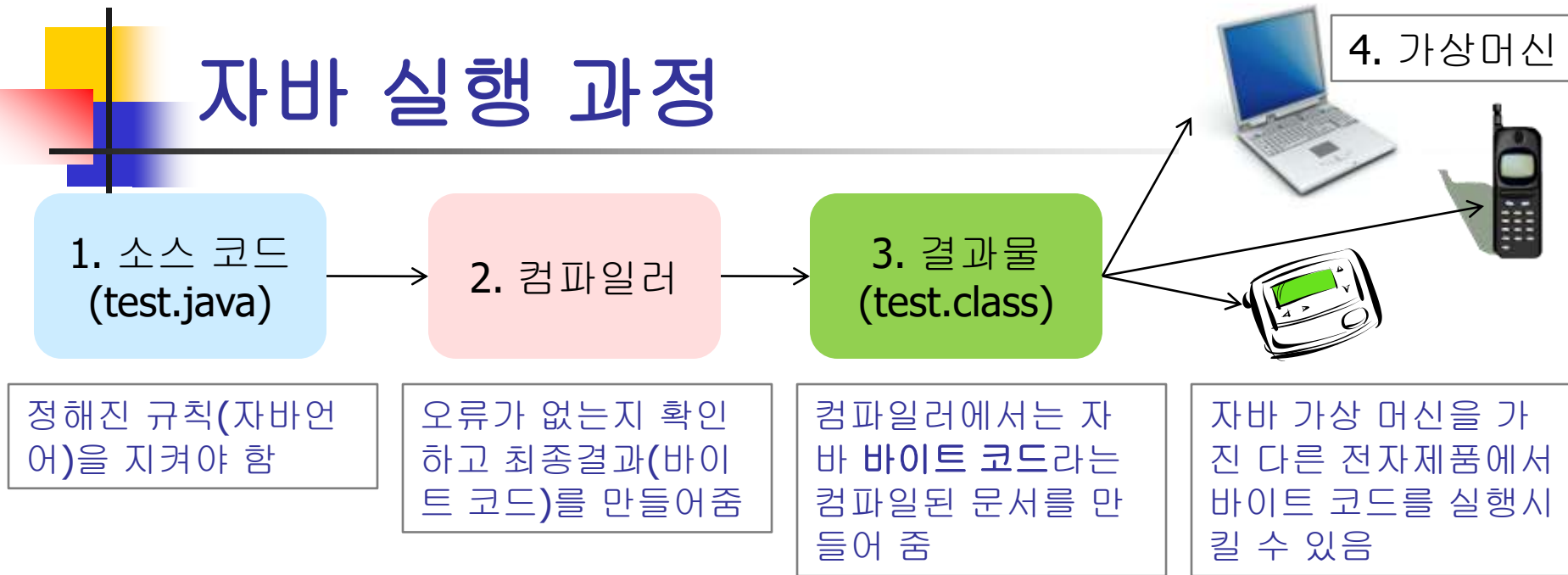
컴퓨터에게 요청(Coding) -> 통역(Compiling) -> 결과 실행(Interpreting)

- 컴파일러 - 바이트 코드(반 기계어)로의 변환
- 인터프리터 - 한 **line**씩 읽어서 실행
 - 컴파일러를 통해서 변환된 언어는 컴퓨터에게 넘겨져서 실행할 수 있는 환경이 필요함
 - 그 환경이 인터프리터

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello JAVA!!!");
    }
}
```

자바 소스코드

자바 실행 과정

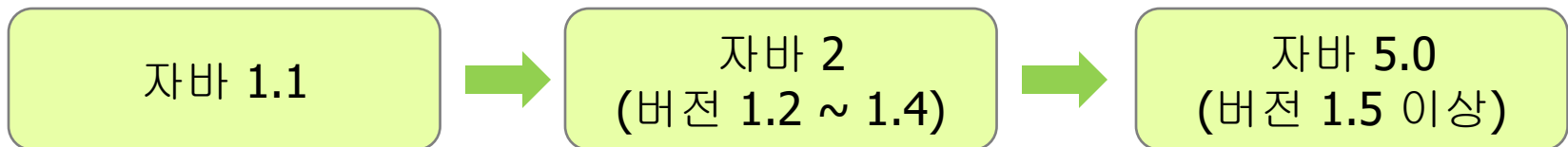


- 소스 코드를 입력하고, **javac** 컴파일러로 컴파일한 다음, 컴파일된 바이트 코드를 자바 가상 머신에서 실행시키면 됨
- 자바 가상 머신(**Java Virtual Machine**) – 바이트 코드를 해당 플랫폼에서 이해할 수 있는 형태로 해석하여 그 프로그램을 실행시켜 줌
- 바이트 코드 – 플랫폼에 무관하게 사용할 수 있음

자바 언어의 역사

■ 자바 언어의 역사

- 1991년 sun사 Green Project 출범 - James Gosling을 주축으로 Oak 라는 인터프리터 개발
 - 가전기기에서 사용할 목적- 하드웨어 독립적인 언어로 구상됨
- 1995년 : sun사와 netscape사 협약
- 1996년 : 자바 지원 netscape 2.0 발표
- 1997년 : jdk1.1 발표
- 1998년 : jdk1.2 발표 (Java2 라는 이름으로 규모를 갖춘 형태의 프로그래밍 언어로 자리매김)
- 2000년 : jdk1.3 발표
- 이후 : jdk1.4/ 5.0 / 6.0 / 7.0 /8.0 /11.0 /17.0 /19.0 버전 발표





JDK 17.0 설치 및 경로 설정

- JDK 17.0 다운로드
 - <https://www.oracle.com/java/technologies/downloads/#java17>
 - Downloads => java for developer
 - 다운로드 : jdk-17_windows-x64_bin.exe (64bit용)
- JRE(Java Runtime Environment)
 - 사용자를 위한 실행환경만 제공
- JDK(Java Development Kit) 자바 개발 도구
 - 실행환경과 함께 반기계어로 변환하는 컴파일러까지도 포함
- Java SE (Java 2 Platform, Standard Edition, J2SE)
 - 일반 솔루션 개발용 자바 (Core and Desktop)
- Java EE (Java 2 Platform, Enterprise Edition, J2EE)
 - 기업 솔루션 개발용 자바 (웹, jsp용 API, 서블릿, EJB)
- Java ME (Java 2 Platform, Micro Edition, J2ME)
 - 마이크로 디바이스(소형기기)에 탑재가 가능한 솔루션 개발용 자바



Java SE Development Kit 8 Update 191 (64-bit) - Setup



Welcome to the Installation Wizard for Java SE Development Kit 8 Update 191

This wizard will guide you through the installation process for the Java SE Development Kit 8 Update 191.

The Java Mission Control profiling and diagnostics tools suite is now available as part of the JDK.

Next >

Cancel



Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

- | | |
|-------------------------------------|-------------------|
| <input checked="" type="checkbox"/> | Development Tools |
| <input checked="" type="checkbox"/> | Source Code |
| <input checked="" type="checkbox"/> | Public JRE |

Feature Description

Java SE Development Kit 8 Update 191 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.

Install to:

C:\Program Files\Java\jdk1.8.0_191

Change...

< Back

Next >

Cancel



Java SE Development Kit 8 Update 191 (64-bit) - Change Folder



Browse to the new destination folder

Look in:



jdk1.8.0_191



Folder name:

C:\Java\jdk1.8.0_191

OK

Cancel



Status: Updating component registration





Oracle Java SE 로드맵에 대한 중요한 정보

향후 Oracle Java SE 릴리스에 대한 액세스에 영향을 주는 변경사항이 있습니다.

기업 사용자는 2019년 1월부터 영향을 받습니다.

이러한 변경사항은 설치하려는 버전에 영향을 주지 않습니다.

추가 지침을 보려면 다음 링크로 이동하십시오.

[추가 정보...](#)

확인(O)



대상 폴더

Java를 다른 폴더에 설치하려면 "변경"을 누르십시오.

설치 위치:

C:\Program Files\Java\jre1.8.0_191

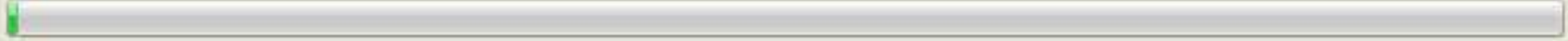
변경(C)...

< 뒤로(B)

다음(N) >



상태: Java 설치



ATMs, Smartcards, POS Terminals, Blu-ray Players, PCs
Set Top Boxes, Mobile Phones, Servers, Switches
Routers, Smart Meters, Embedded Systems, Devices
Automobiles, Marketers, Lottery
Systems, Access Control Systems, Building Controls
Programs

3 Billion Devices Run Java

 **Java™** | #1 Development Platform 



Java SE Development Kit 8 Update 191 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

Next Steps

Close



JDK 17.0 설치 및 경로 설정

- 설치 이후에 해야 할 추가적인 설정
 - c:\ Java\ jdk-17.0.4\ bin\
 - javac.exe : 자바 컴파일러
 - java.exe : 자바 Launcher
 - JDK가 설치되어 있는 경로를 PATH 환경변수에 추가
 - JAVA_HOME 설정 : c:\ Java\ jdk-17.0.4
 - path 설정 : %JAVA_HOME%\ bin
(c:\ Java\ jdk-17.0.4\ bin)
 - classpath 설정 : %classpath%;.
 - 명령 프롬프트 창에서 런처와 컴파일러 실행해보기
 - c:\ >java -version

```
C:\Users\W82106>java -version
java version "17.0.4" 2022-07-19 LTS
Java(TM) SE Runtime Environment (build 17.0.4+11-LTS-179)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.4+11-LTS-179, mixed mode, sharing)
```




JDK Documentation API 설치

- JDK Documentation API

- 사용할 클래스에 대한 내용을 문서화하여 제공
 - <https://www.oracle.com/java/technologies/javase-jdk17-doc-downloads.html>

- **Java SE Development Kit Documentation Downloads**

- 압축 풀고, docs\ api\ index.html 을 실행
- 또는 online으로 확인
 - <https://docs.oracle.com/en/java/javase/17/docs/api/index.html>



Eclipse 설치

■ Eclipse 툴

- 통합 개발 환경
- <http://www.eclipse.org>
- Eclipse IDE for Enterprise Java and Web Developers 다운로드
- zip 파일 - 압축을 풀어주기만 하면 됨



JAVA 프로그램 작성

```
import java.lang.*; //생략 가능한 패키지
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello JAVA!!");
    }
}
```

```
d:\ java\ study>javac Hello.java
```

```
d:\ java\ study>java Hello
```

```
Hello JAVA!!
```

```
d:\ java\ study>dir
```

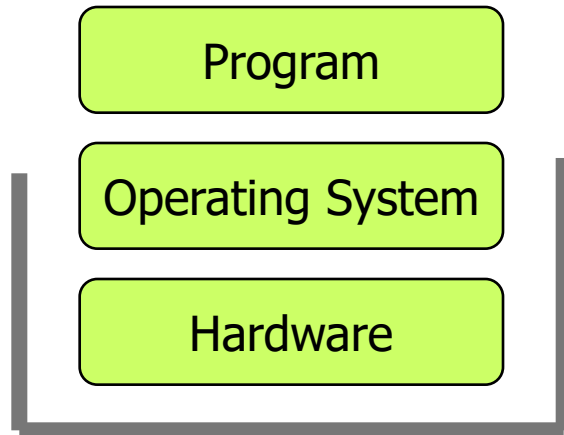
```
Hello.java
```

```
Hello.class
```

자바 프로그램의 실행구조

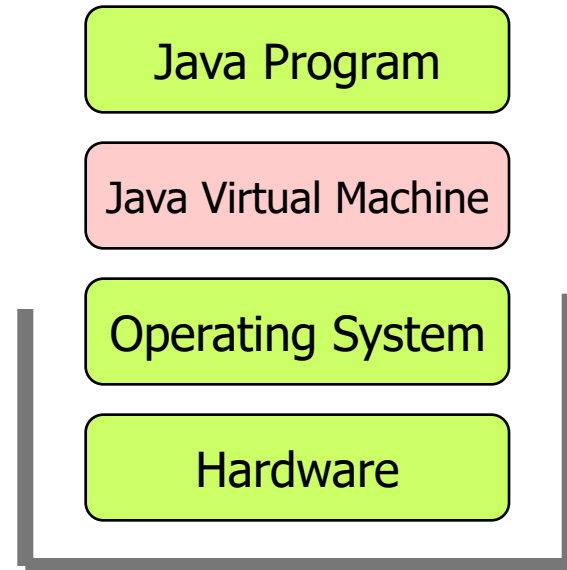
■ 일반적인 프로그램의 실행구조

- 일반적인 프로그램은 Windows 나 Linux 같은 운영체제 (Operating System) 위에서 실행됨



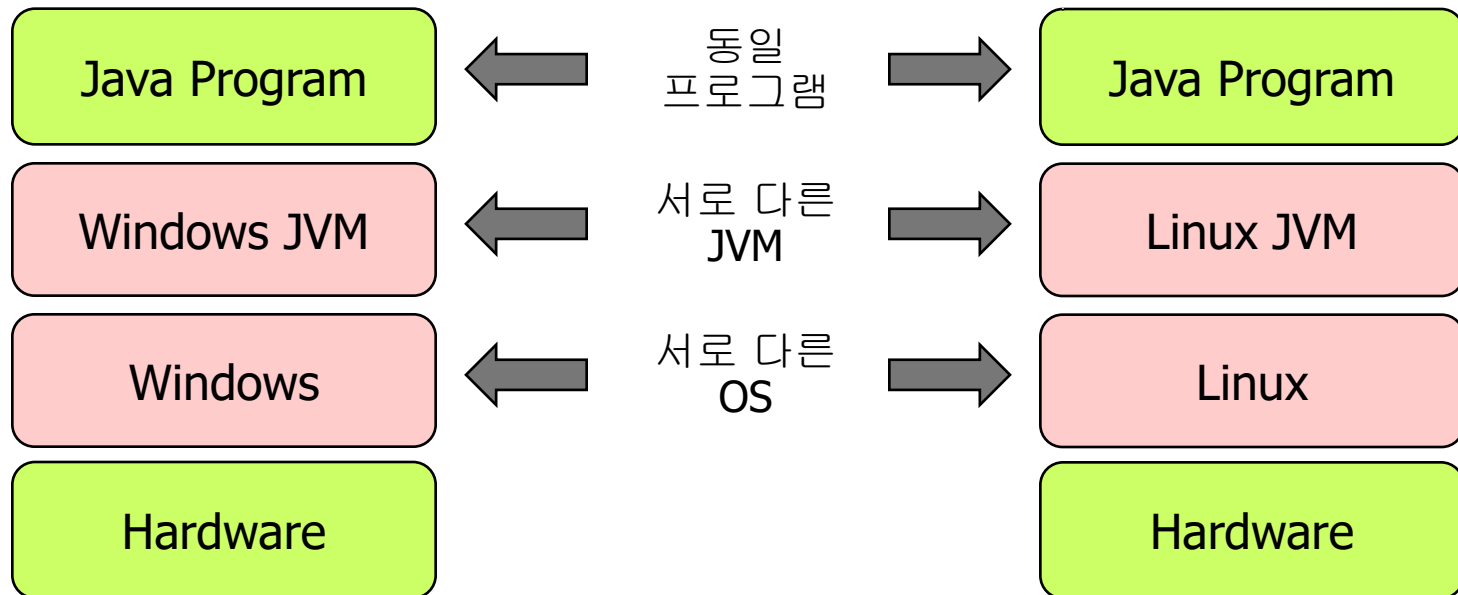
■ 자바 프로그램의 실행구조

- 운영체제는 자바 가상머신을 실행시키고, 자바 가상머신은 자바 프로그램을 실행시키는 구조



자바 프로그램의 실행구조

- 운영체제에 독립적인 자바 프로그램
 - 운영체제 별로 존재하는 차이점을 가상머신이 다 해결해 주기 때문에 자바 프로그램은 운영체제에 상관없이 실행이 됨





자바가상머신(JVM)

■ JVM(Java Virtual Machine)

- 자바 소스코드를 컴파일하게 되면 바이트 코드로 변환 되는데, JVM은 이러한 **바이트 코드**를 읽어서 실행할 수 있도록 해주는 도구
- 인터프리터의 기능을 수행함(한 라인씩 읽어서 실행)
- 프로그램을 실행시키는 도구
- 바이트 코드를 해당 플랫폼에서 이해할 수 있는 형태로 해석하여 그 프로그램을 실행시켜 줌



자바 컴파일러와 자바 바이트 코드

■ 자바 컴파일러

- 소스파일에 저장되어 있는 **소스 코드**를 가상머신이 이해할 수 있는 자바 **바이트 코드로 변환**해주는 프로그램
- **javac.exe**

■ 자바 Launcher

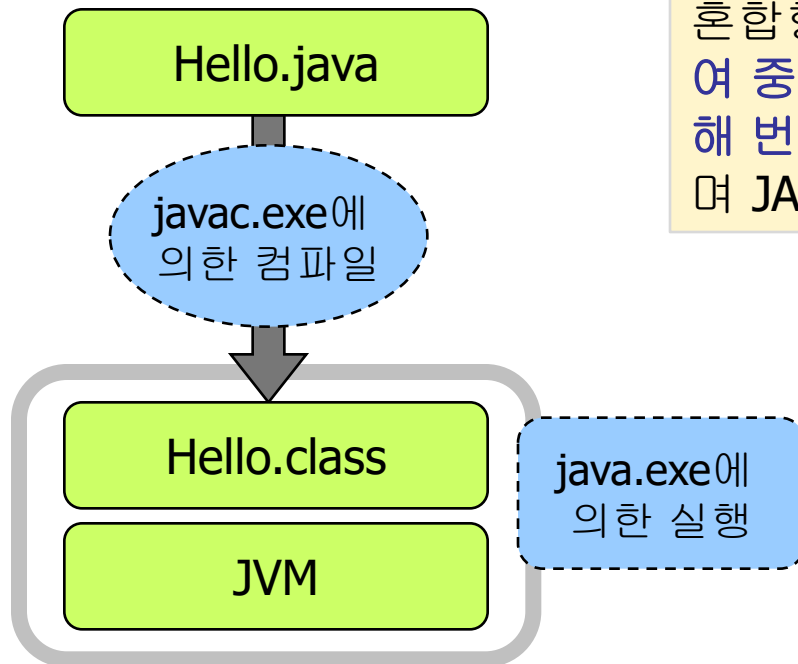
- 자바 가상머신을 구동시키고, 그 위에 자바 프로그램이 실행되도록 돕는 프로그램
- **java.exe**

■ 자바 바이트 코드

- 자바 컴파일러에 의해서 생성되는 코드(반 기계어)
- 플랫폼에 무관하게 사용할 수 있음

자바 컴파일러와 자바 Launcher

- `javac.exe` 와 `java.exe`의 역할
 - 자바의 소스코드가 `javac.exe`에 의해서 컴파일되고, 컴파일된 파일이 `java.exe`에 의해서 가상머신을 기반으로 실행됨



혼합형 방식의 언어: 고급 언어를 컴파일 하여 중간 언어로 변환한 후, 인터프리터에 의해 번역을 실행하는 방식의 언어를 의미하며 JAVA가 이에 속한다.



자바의 특성

- 이식성이 높은 언어
 - 한번 코딩되어 컴파일된 상태의 클래스 파일은 다시 수정하지 않고도 **JVM**이 설치되어 있는 시스템에서는 실행 가능
- 완벽한 객체지향 언어
- 멀티스레드 지원

자바 코드의 구조

```
D:\>java Hello  
Hello Java!!
```

소스 파일

클래스

메소드1

처리할 일

메소드2

처리할 일

```
import java.lang.*; //생략 가능한 패키지  
public class Hello  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Hello JAVA!");  
    }  
}
```

- 모든 자바 애플리케이션에는 최소한 클래스 한 개가 있어야 하며, 적어도 **main** 메서드 하나가 있어야 함
- 자바에서는 모든 것이 클래스 안에 들어감
- JVM이 실행되면 우선 사용자가 명령행에서 지정한 클래스를 살펴보고 나서, **main()** 이라는 특별한 메서드를 찾아봄
- **main()** 을 찾으면 JVM에서는 **main** 메서드의 **{}** 안에 있는 모든 것을 실행시킴
- 프로그램을 실행시킨다는 것은 JVM에 “해당 클래스(**Hello.class**)”를 불러오고 그 **main()** 메서드를 시작하라는 것.



자바 프로그램의 골격과 구성

- `Hello.java` 는 하나의 클래스로 이뤄져 있는 프로그램
- 그 클래스 안에는 하나의 메소드가 존재
- 프로그램을 실행시키면 `main()` 메소드 안에 있는 문장이 순차적으로 실행됨
- 클래스 이름이 `Hello` 이면, 컴파일시 생성되는 파일의 이름은 `Hello.class`
 - **클래스명과 파일명은 동일해야 함**(클래스 이름 `Hello`=> `Hello.java`)
- `System.out.println()` 의 괄호안에 출력하고 싶은 것을 큰 따옴표로 감싸서 넣으면 출력이 됨
 - `System.out.println()` 은 출력을 한 다음에 행을 바꿈
 - `System.out.println()` 과 같이 컴퓨터에게 무엇인가 일을 시키는 문장을 '명령문'이라 하고, 이러한 **명령문의 끝에는 반드시 세미콜론(;)을 붙여서 명령문의 끝을 표시**



자바 프로그램의 골격과 구성

- package

- 비슷한 유형의 클래스들끼리 묶어서 관리
- 해당 package 내에 있는 클래스를 사용하려면 **import** 라는 예약어를 사용

- **import java.lang.***

- **java** 패키지 내의 **lang** 패키지 안의 모든 클래스를 현재 파일에서 사용할 수 있도록 하겠다는 뜻

```
import 패키지명.클래스명;
```

- import java.lang.String;
- import java.lang.System;
- import java.io.File;
- import java.util.Date;
- import java.sql.Connection;



주석

- 주석
 - 컴파일의 대상에서 제외되는 문장
 - 특정 코드 부분에 대해 간단한 설명을 달 때 사용
- 주석의 종류
 - `/* ~ */` - 블록 단위 주석(여러 줄 주석)
 - `//` - 행 단위 주석
 - `/** ~ */` - 자바의 고유한 **Document** 주석, 사용자 정의 **Documentation API**를 만들 때 사용하는 주석 처리



자바 프로그램의 골격과 구성

- 클래스
 - 여러 개의 클래스를 하나의 파일에서 사용할 수 있음
 - 주의 : **public**이라는 예약어를 사용할 수 있는 클래스는 하나 뿐.
 - 클래스 모두가 **public**을 사용하지 않아도 되지만, **public**을 사용하려면 그것은 하나여야 하고, **public**이 붙을 수 있는 클래스는 파일명과 동일해야 함
- **main()** 메서드는 파일명과 동일한 클래스내에 있어야 함



예 - Hello2.java

```
import java.lang.*;
public class Hello2{
    public static void main(String[] args){
        System.out.println("Hello Java!!");
        System.out.println("Hello Jsp!!");
    }
}

class Test
{
    public void write()
    {
        System.out.println("Test Class!!");
    }
}
```

- 파일명과 동일한 클래스내에 **main()**이 있어야 하고,
- 파일명과 동일한 클래스만 **public**이 붙을 수 있고,
- **public class** 는 아예 없거나, 하나만 가능

사용자 정의 명칭의 규칙

5abc
ab+9
ab 7
if
ab\$3
_abc

■ package, 클래스, 메서드, 필드의 이름 정의 규칙

- 클래스, 메서드, 필드 : 첫글자는 \$, _, 영문자
- 공백문자는 포함할 수 없음
- 특수문자는 사용할 수 없음
- 숫자는 첫 글자가 아닐 경우 사용 가능
- 예약어는 사용할 수 없음

■ 권장사항

- 클래스 : 첫글자는 \$, _, 대문자
- 메소드(함수) : 첫글자는 \$, _, 소문자
- Field(변수) : 소문자
- 합성어의 첫글자는 대문자

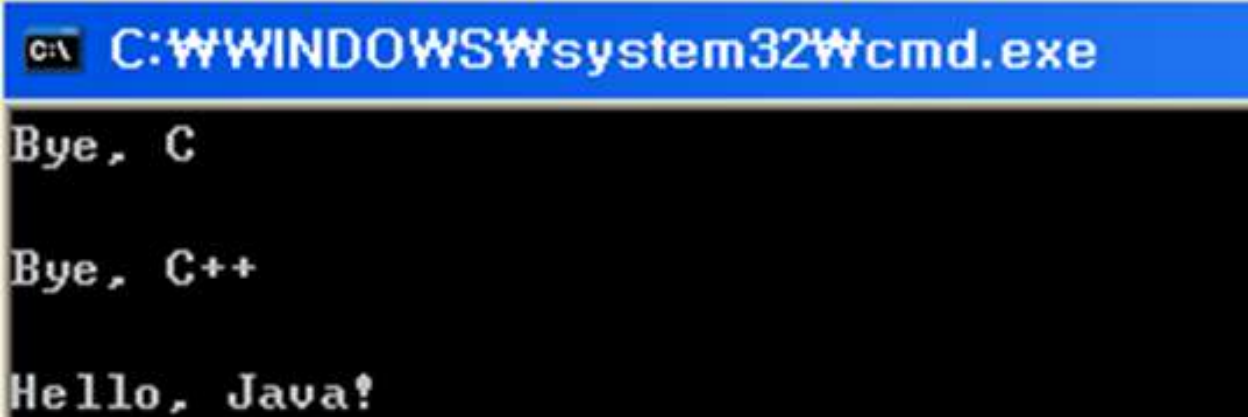
낙타표기법 - camel case

- HelloJava
- 상수 : 모든 문자를 대문자로 구성, 둘 이상의 단어가 연결되는 경우는 _(언더바)를 사용
 - final int COLOR = 7;
 - final int COLOR_RAINBOW = 5;



실습

- 다음 세 문장을 출력하는 프로그램을 작성하시오



```
C:\WINDOWS\system32\cmd.exe
Bye, C
Bye, C++
Hello, Java!
```

- Documentation API 이용
 - 줄바꿈하지 않고, 화면 출력하는 메서드 찾아보기
- 위에서 찾은 메서드를 이용하여 다음과 같이 출력되도록 프로그램을 작성하시오
 - 화면 출력 명령문을 2번 써서 완성하기

이름 : 홍길동, 나이 : 20

- 다음 두 문장의 출력 결과를 확인하는 프로그램을 작성해 보자
 - `System.out.println("2+5=" + 2 + 5);`
 - `System.out.println("2+5=" + (2 + 5));`



변수, 자료형



변수

■ 변수

- 프로그램 실행에 필요한 데이터를 저장하기 위해서 메모리에 공간을 만들어 할당하고, 이름을 부여한 것
- 데이터를 담아 놓는 메모리 공간
- 데이터의 저장과 참조를 위해 할당된 메모리 공간

■ 데이터타입(data type) - 자료형

- 변수 공간을 얼마나 확보할 것인지 크기를 정하는 것
- 자료에 대한 형태, 종류
- 변수를 만들 때 어떠한 데이터가 들어가며 메모리는 몇 바이트나 차지하는지를 명시해 주는 것

변수의 선언

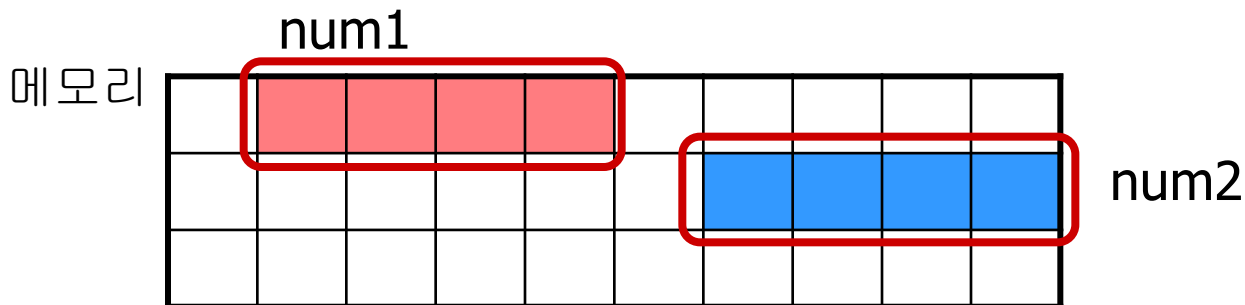
■ 변수 선언

데이터타입 변수명

예) `int num;`

■ `int num;`

- `int` - 정수를 저장할 메모리 공간을 할당하겠다.
- `num` - 그 메모리 공간(변수)에 접근할 때에는 `num` 이라는 이름을 사용하겠다.




```
int num1;  
int num2;
```



변수에 값 할당

- 선언된 변수에 값을 할당하는 방법

```
int a;  
a = 23;
```



- 선언과 동시에 초기화

```
int a = 23;
```



변수에 값 할당

- 여러 개를 동시에 선언과 할당

```
int a=10, b=20;
```

- 분리하여 여러 개를 동시에 선언과 할당

```
int a, b;  
a=10;  
b=20;
```

- 변수끼리의 할당

```
int a=10, b;  
b=a;
```


예제

```
class VarTest1{
    public static void main(String[] args){
        //변수 선언
        int a;
        a=10; //값 할당

        //선언과 동시에 값 할당
        int b=20;

        //여러 개를 동시에 선언과 할당
        int c=30, d=40;

        //여러 개를 선언한 후, 할당
        int e, f;
        e=100;
        f=200;

        //변수끼리의 할당
        int k=45;
        int n = k;

        System.out.println("a=" + a + ", b="+b);
    }
}
```

기본 자료형

자료형	크 기	데이터	기억가능범위
byte	1byte	정수	-128 ~ 127
short	2byte		-32,768 ~ 32,767
int	4byte		-2,147,483,648~ 2,147,483,647
long	8byte		-9223372036854775808 ~ 9,223,372,036,854,775,807
char	2byte	문자	모든 유니코드 문자
float	4byte	실수(부동 소수점수)	$\pm 1.5 \times 10^{-45} \sim \pm 3.4 \times 10^{38}$
double	8byte	float보다 2배 정밀한 실수	$\pm 5.0 \times 10^{-324} \sim \pm 1.7 \times 10^{308}$
boolean	1byte	참과 거짓(논리)	true, false

- 숫자형 – 정수형(**byte, short, int, long**), 실수형(**float, double**)
- 문자형 - **char**
- 논리형 - **boolean**



정수 자료형

■ 정수 자료형

- 연산이 중심이 되는 데이터는 **int**형으로 선언
 - CPU는 **int** 형 정수연산을 가장 고속으로 처리하게끔 설계되어 있음
 - 따라서 정수형 연산을 진행할 때, 모든 피연산자를 **int**형으로 변환하는 과정을 거침
- 데이터의 표현이 중심이 되는 경우는 **byte, short, float** 등으로 데이터를 표현
 - 노래와 같은 음원정보 저장, 게임캐릭터의 그래픽 정보 등



실수 자료형

■ 부동 소수점 자료형

- 정수를 포함한 모든 실수를 다룰 수 있게 해줌
- 소수점 이하 표현 가능
- 컴퓨터가 실수를 표현하는 방식-아주 가까운 근사치를 표현하는 방식(오차가 존재)
 - 값의 표현 범위보다 정밀도 (값을 정확히 표현할 수 있는 능력)를 우선시하여 선택해야 함
- float - 소수점 이하 7자리까지 유효
- double - 소수점 이하 15자리까지 유효

■ 논리형

- 참과 거짓(true, false)을 다루는 자료형
- boolean



문자 자료형

■ char

- 영문자나 한글과 같은 **문자 하나**를 저장해서 사용
- char 형 변수에 값 저장시 **' '**로 묶어서 저장
- 문자 하나를 변수에 저장하면 **실제로는 해당 문자의 유니코드 값이 저장됨** (char은 실상 숫자형 자료형 => 연산도 가능)
 - char a='A'; => char a=65;
 - char b='한' => char b=54620;

■ 아스키코드(영문, 1바이트로 표현)

- 컴퓨터에서 처리하기 용이한 **8비트** 데이터를 이용하기 위하여 숫자, 문자, 특수문자에 번호를 부여한 것
- 아스키는 **128**개의 가능한 문자조합을 제공하는 **7비트(bit)** 부호

■ 유니코드

- 문자 하나를 **2바이트**로 표현하는 문자체계(**65,536** 개의 문자 표현 가능)
- 세계의 모든 언어를 표현할 수 있는 문자체계



참조형

- String 형

- 문자열을 다룰 때 사용
- "ABC" 같은 연속의 문자들을 다룰 때
- " " 사용

상수(Constant)의 분류와 표현

■ 데이터의 분류와 표현법

상수의 분류와 표현법							
	정수형			실수형		논리형	문자열형
데이터타입	int	long	char	float	double	boolean	String
Byte수	4	8	2	4	8	1	
표현법	0, 100	100L	'A', 'C'	0.0F	0.0D	false, true	"str"



할당(Assignment)

- 할당(**Assignment**)

- 변수에 데이터를 넣는 것

- 할당의 기본 법칙

- 오른쪽에서 왼쪽으로만 할당이 가능 (절대적)
- 변수끼리도 할당이 가능 (오른쪽에서 왼쪽으로 할당)

예제

```
public class VarTest2 {  
    public static void main(String[] ar){  
        //자료형  
        //1.정수형  
        byte a = 127;  
        short b=32000;  
        int c = 2100000000; //21억, 0이 8개  
        long d = 21000000000L; //0이 9개, int형 값의 범위를 넘어서는 경우에는 뒤에 L을 붙여야 함  
  
        long e = (long)a*b*c; //연산시에는 변수의 모든 타입을 제일 큰 자료형으로 일치시켜서 계산  
        //byte*short*int => int*int*int로 변환  
        //int*int*int의 결과값은 int  
  
        System.out.println("a=" + a + ",b=" + b);  
        System.out.println("c=" + c + ",d=" + d);  
        System.out.println("a*b*c => " + e);  
  
        //byte x=128; //error  
        byte y=127;  
        y++;  
        System.out.println("y=" + y);  
    }  
}
```

```
a=127,b=32000  
c=2100000000,d=21000000000  
a*b*c => 8534400000000000  
y=-128  
f=A,g=가  
66  
67  
C
```



예제

//2. 문자형

char f = 'A'; //반드시 작은 따옴표(')로 감싸야 함, 영어는 한글자가 1byte
//실제로는 A 에 해당하는 유니코드 값인 65 가 들어감

char g = '가'; //한글은 1글자가 2byte

//char h = 'abc'; //error, char는 문자 하나만 저장

System.out.println("f=" + f + ",g=" + g);

System.out.println(f+1); //f에는 65가 들어가므로, f+1=>66

int h = f+2; //67

System.out.println(h);

char i = (char)h; //유니코드 67에 해당하는 문자인 C

System.out.println(i);

char k = 65; //유니코드값을 넣어도 됨 'A' => 65

System.out.println(k); //A

System.out.println((int)f); //65

}

}



예제

```
b=true  
i : 3  
j : 4  
i > j : false
```

```
class VarTest3  
{  
    public static void main(String[] args)  
    {  
        //3. 논리형  
        //자바의 논리형은 0, 1로 호환되지 않음  
        boolean b = true;  
        System.out.println("b=" + b);  
  
        int i = 3, j = 4;  
        boolean b2 = (i>j); //결과값은 false  
        System.out.println("i : " + i);  
        System.out.println("j : " + j);  
        System.out.println("i > j : " + b2);  
    }  
}
```

예제

```
float a=3.14,double b=3.1415
a * b :9.864310329556465
float type pie : 3.142857
double type pie : 3.142857142857143
f2=0.567, f3=456.0
```

```
class VarTest4{
    public static void main(String[] args) {
        //4. 실수형 - 소수점 이하 정밀도와 관련 있다
        //float : 뒤에 F를 붙여야 함
        float a = 3.14f; //f를 붙이지 않으면 double로 인식
        double b = 3.1415;
        double c = a*b; //double 과 float 의 곱은 암시적으로 double 로 변환
        System.out.println("float a=" + a + ",double b=" + b);
        System.out.println("a * b :" + c);

        //소수점 이하 정밀도 확인
        float x = 22f, y = 7f;
        float z = x/y;
        System.out.println("float type pie : "+ z);

        double i = 22, j = 7;
        double k = i/j;
        System.out.println("double type pie : "+ k);

        float f2 = .567F;
        float f3 = 456;
        //float 형은 정수(int)를 입력 받더라도 자동으로 float 형으로 형변환되어 소수점 .0 이 붙게 된다.
        System.out.println("f2="+f2+", f3="+f3); // f2=0.567, f3=456.0
    }
}
```

• 소수점 이하 정밀도

예제

```
class VarTest5
```

```
{
    public static void main(String[] args)
    {
        //5. 문자열 자료형(String)-참조형, 여러 개의 문자열을 넣는다.
        //" (큰따옴표)로 감싸준다
        String a = "제 이름은 ";
        String b="홍길동입니다", c = "김연아입니다";
        System.out.println(a+b);
        System.out.println(a+c);

        int year=2015;
        System.out.println("올해 : " + year + "년");

        //+ : 피연산자가 숫자일때는 덧셈
        //    피연산자에 문자열이 있으면 문자열 연결 연산자
        int num1 = 7, num2 = 5;
        System.out.println(num1 + num2 + "<=결과");
        System.out.println("결과=>" + num1 + num2);
        System.out.println("결과=>" + (num1 + num2));
    }
}
```

```
제 이름은 홍길동입니다
제 이름은 김연아입니다
올해 : 2015년
12<=결과
결과=>75
결과=>12
```

실습

■ 1. 주소록

- 변수 - 이름, 나이, 전화번호, 주소
- 임의의 값을 할당하고, 출력하기

이름 : 홍길동, 나이 : 20
전화번호 : 010-100-2000, 주소 : 당산동

■ 2. 상품 재고 조절

- 변수 - 상품, 수량, 원가, 판매가격

상품 : 휴대폰, 수량 : 15
원가 : 260000, 판매가격 : 350000

- 3. 변수 선언하고 출력하기
 - char 형 변수 선언, 값 할당
 - float 형 변수 선언, 값 할당
 - double 형 변수 선언, 값 할당
 - byte 형 변수 선언, 값 할당
 - short 형 변수 선언, 값 할당
 - long 형 변수 선언, 값 할당
 - String 형 변수 선언, 값 할당
 - boolean 형 변수 선언, 값 할당
 - 출력하기



실습

- 4. 사용자 아이디와 비밀번호를 변수에 저장한 후, 화면 출력하기

```
아이디 : hong  
비밀번호 : 1234
```




실습

- 실수형 변수(float)를 하나 선언한 후 값을 할당
- char변수를 하나 선언하면서 동시에 값을 할당
- long변수 2개를 선언한 후 값을 할당
- String변수 2개를 선언함과 동시에 값을 할당



변수의 종류

- 변수의 종류
 - 멤버변수(instance 변수)
 - 클래스변수(static 변수)
 - 지역 변수(local 변수)
 - 메서드나 {} 블록 안에서 선언된 변수
 - 지역변수는 반드시 초기화해야 함(stack 영역의 변수는 초기화해야 함)
 - 초기화하지 않고, 사용하면 Error남
 - 멤버변수와 클래스변수는 초기화하지 않아도 디폴트 값이 들어감



예제

```
static변수 x=0  
지역변수 z=0  
지역변수 s=,d=0.0,b=false
```

```
public class VarTest {  
    static int x; //static변수  
    int y; //멤버변수  
  
    public static void main(String[] ar){  
        int z=0; //지역변수는 반드시 초기화해야함(stack 영역의 변수는 초기화  
        해야함)  
        String s="";  
        double d = 0.0;  
        boolean b = false;  
        System.out.println("static변수 x="+x);  
        System.out.println("지역변수 z="+z);  
        System.out.println("지역변수 s="+s+",d="+d+",b="+b);  
    }  
}
```



Wrapper 클래스

- 자료형을 효율적으로 관리함과 동시에 완벽한 은닉화를 추구하기 위해 만들어진 **자료형 대체 클래스**
 - 완벽한 은닉화를 위해서 prototype 자료형을 대체할 수 있는 클래스를 만들었고 그것들을 통틀어 Wrapper 클래스라고 부름
 - **기본형 변수들도** 때로는 **객체로 다루어져야 하는 경우가** 있는데, 이 때 사용되는 것이 wrapper 클래스
- Boolean/ **Character**/ Byte/ Short/ **Integer**/ Long/ Float/ Double
 - 이 클래스들로 객체를 만들면 메모리를 조사했을 때 레퍼런스 자료형이므로 항상 4byte로 보인다
 - Integer.MAX_VALUE, Integer.MIN_VALUE



예제

```
public class VarMaxMin {  
    public static void main(String[] ar){  
        byte b_min = Byte.MIN_VALUE;  
        byte b_max = Byte.MAX_VALUE;  
  
        int i_min = Integer.MIN_VALUE;  
        int i_max = Integer.MAX_VALUE+1; //int의 범위를 벗어나므로, garbage값이 들어감  
  
        char c_min = Character.MIN_VALUE;  
        char c_max = Character.MAX_VALUE;  
  
        System.out.println("byte min="+b_min);  
        System.out.println("byte max="+b_max);  
        System.out.println("int min="+i_min);  
        System.out.println("int max+1="+i_max);  
        System.out.println("char min="+(int)c_min); //null => 0  
        System.out.println("char max="+(int)c_max);  
    }  
}
```

```
byte min=-128  
byte max=127  
int min=-2147483648  
int max+1=-2147483648  
char min=0  
char max=65535
```



System.out.println() 정리

```
class PrintInTest
{
    public static void main(String[] args)
    {
        int a=10;
        System.out.println(a); //변수만 출력
        System.out.println("a 는 " + a);
        System.out.println("재미있는 Java"); //문자열만 출력
    }
}
```



```
10
a 는 10
재미있는 Java
```

이스케이프 시퀀스(Escape Sequence)

- 문자열 안에서 특별한 의미로 해석되는 문자를 가리켜 '이스케이프 시퀀스'라 함
- 표현하기 어려운 문자 상수를 표현할 수 있는 방법이 Escape Sequence

• \n	개행
• \t	탭(Tab)
• \"	큰 따옴표(Quotation mark)
• \\	역슬래쉬(Backslash)

```
System.out.println("제가 어제 "당신 누구세요?" 라고 물었더니");
```

문자열안에 큰따옴표가 들어가면 이는 문자열의 구분자로 인식됨

```
System.out.println("제가 어제 \"당신 누구세요?\"라고 물었더니");
```

문자열 안에 큰따옴표를 삽입하려면 이스케이프 시퀀스 사용

실습

- Escape Sequence 이용하여 출력하기
 - 큰 따옴표, 탭, 역슬래시, new Line

```
<따옴표 사용>
그는 말했다. "나는 java가 재밌어!"
그는 말했다. ✓ 나도 java가 재밌어!
✓
<역슬래시 사용>
c:\download\test.txt
```

국어	영어	수학
90	80	75



자바의 자료형

- 자바의 자료형

- 기본 자료형(값 타입)

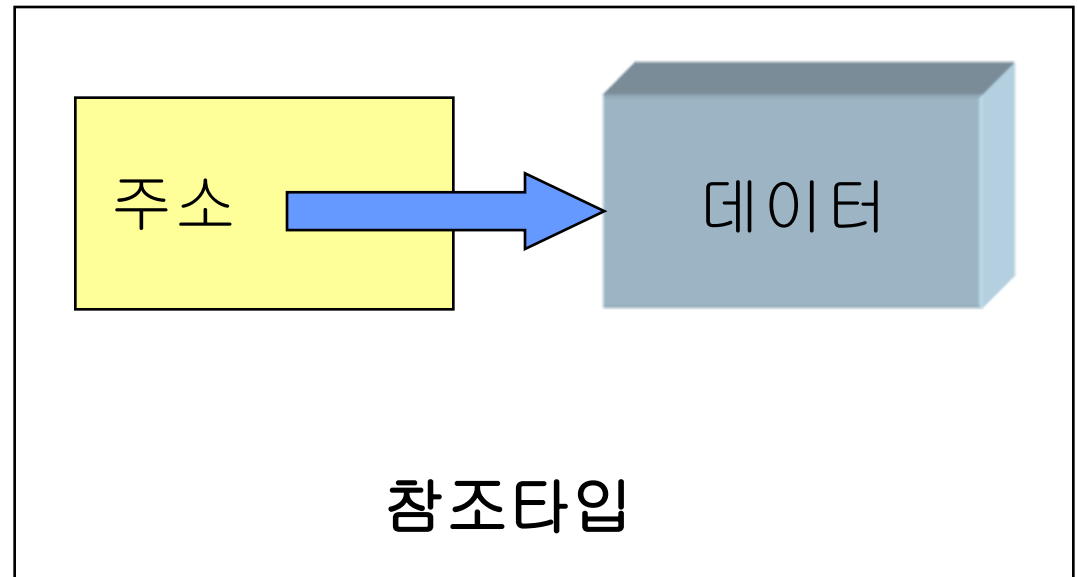
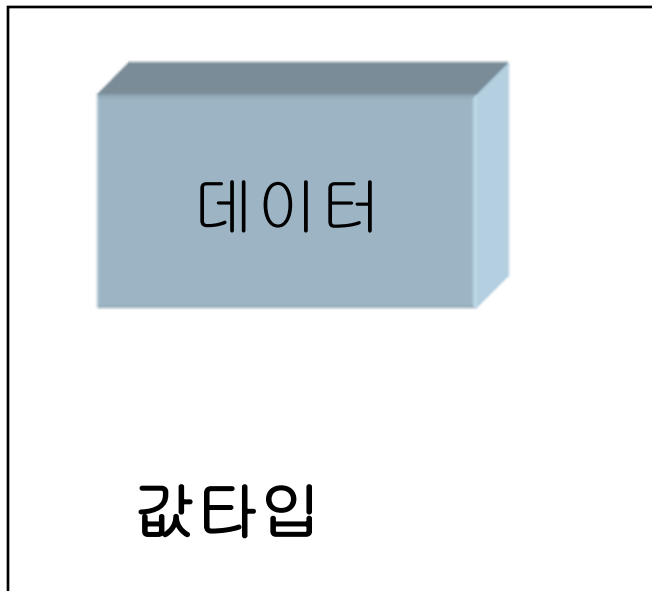
- 수치형(정수형, 실수형), 문자형, 논리형
 - byte, short, int, long, float, double, char, boolean

- 참조형(Reference Type, 참조타입)

- 클래스형, 인터페이스형, 배열
 - ex) String s="java";
String s=new String("java");

자료형(Data Type)

- 자료형 분류
 - 기본자료형 - 값타입(Value Type)
 - 참조타입(Reference Type)





데이터 타입(Data Type)

■ 값타입(Value Type)

- 변수의 **실제값**을 가지고 있는 것
 - 작고 빠르게 처리해야 하는 것은 값타입을 이용
- 메모리의 **스택**에 저장됨
- 변수의 선언과 동시에 메모리 생성
- 변수의 타입에 따라서 메모리의 크기도 달라짐
- 변수를 선언할 때 **new** 연산자를 이용하지 않아도 자동으로 **new** 연산자가 호출되어 메모리가 생성되는 특징



데이터타입(Data Type)

■ 참조타입(Reference Type)

- 실제 값은 다른 곳에 있으며 값이 있는 주소를 가지고 있어서 나중에 그 주소를 참조해서 값을 가져올 수 있는 것
 - 덩치가 크면 실제 데이터는 다른 곳에 두고 그 곳에 대한 참조만을 가지고 작업하면서 필요할 때마다 그 데이터를 열어 사용
- 메모리의 **힙**에 실제 값을 저장하고 그 참조값을 갖는 변수는 스택에 저장
- 변수의 선언과 메모리 생성의 분리
- 클래스(**Class**) 계열들은 모두 **new** 연산자를 이용하여 사용자가 직접 메모리를 생성해 주어야 함
 - 클래스, 인터페이스, 배열 등
- 클래스형 중에서 기본 내장형인 **String**형 등

메모리

메소드 영역 - 메소드의 바이트 코드, **static** 변수
스택영역 - 지역변수, 매개변수
힙 영역 - 인스턴스

■ 프로그램에서 말하는 메모리

- 물리적인 메모리가 아니라, 가상 메모리
 - 운영체제는 하드디스크, **RAM**(메인 메모리), **Cache**, **Register** 를 하나의 커다란 메모리로 보이게끔 해줌

가상메모리
전체영역

메서드 영역

■ 가상 메모리를 나누는 기준

- 코드영역 - 실행할 프로그램의 코드를 올려 놓을 공간
- 데이터 영역(**static**영역) - 프로그램이 종료될 때까지 유지해야 할 데이터를 저장할 공간
- **스택** 영역 - 아주 잠깐 사용하고 삭제할 데이터의 저장공간
- **힙** 영역 - 프로그래머가 원하는 방식으로 쓸 수 있는 공간

코드 영역

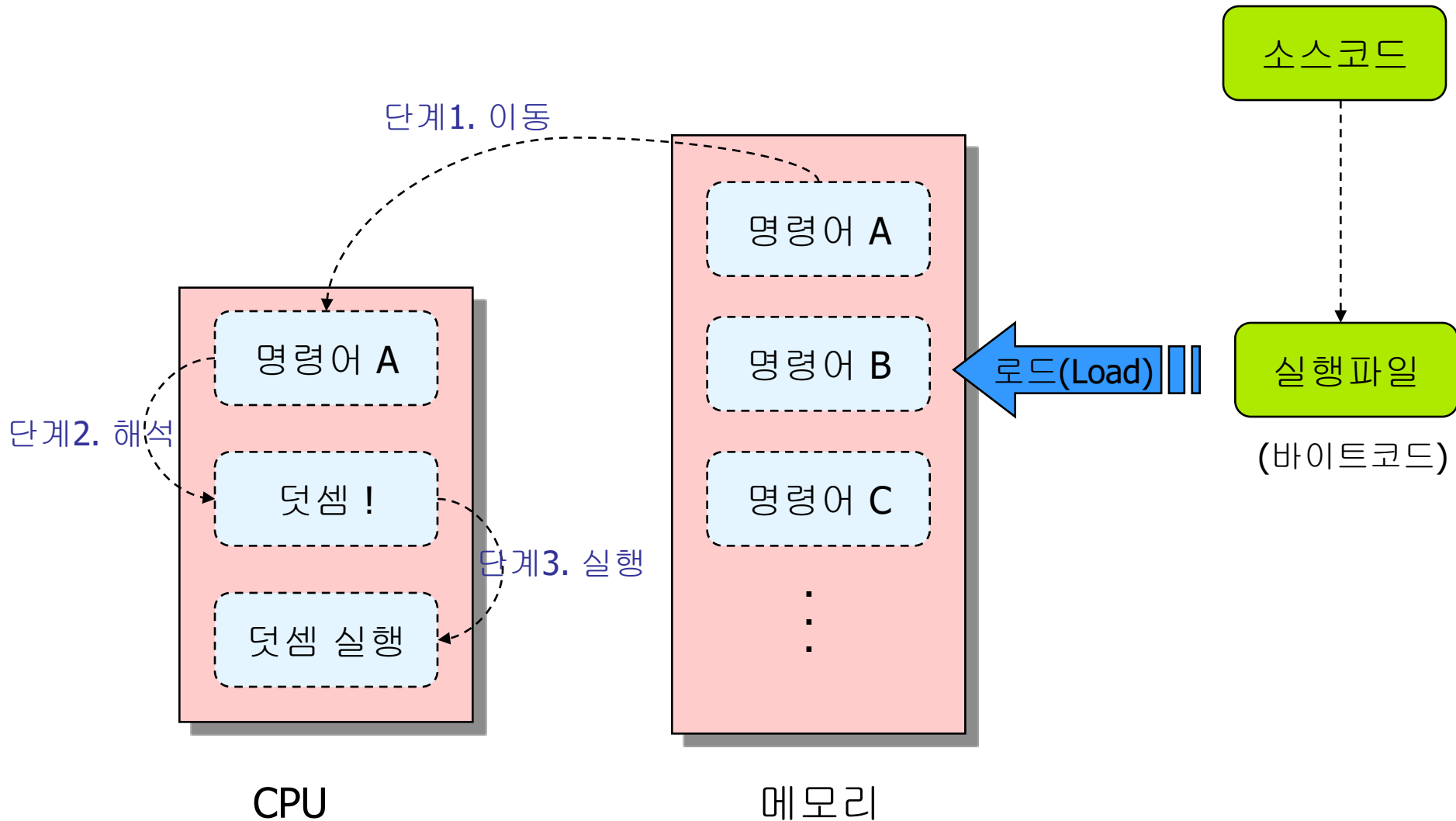
데이터 영역

힙 영역

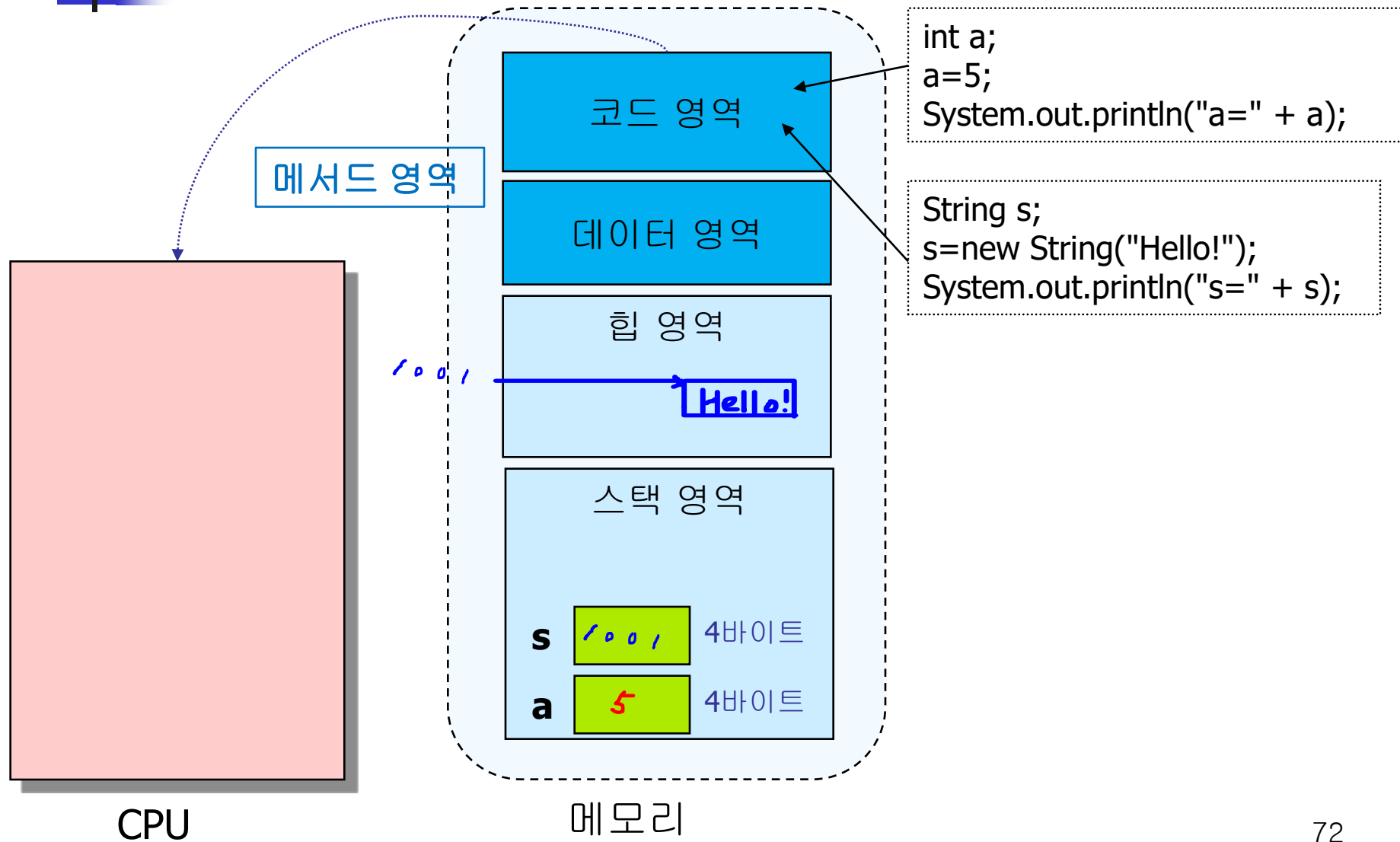
스택 영역

프로그램의 실행과정

```
int a=5;  
a++;  
System.out.println("a=" + a);
```

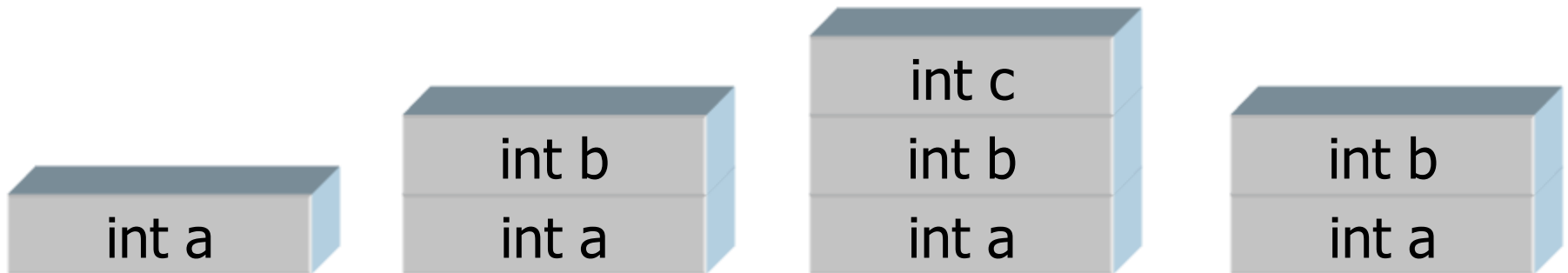


자바 가상 머신의 메모리 모델



스택(stack)/힙(heap)

- 변수는 모두 메모리에 저장됨
- 값타입 - 스택 영역에 데이터가 저장

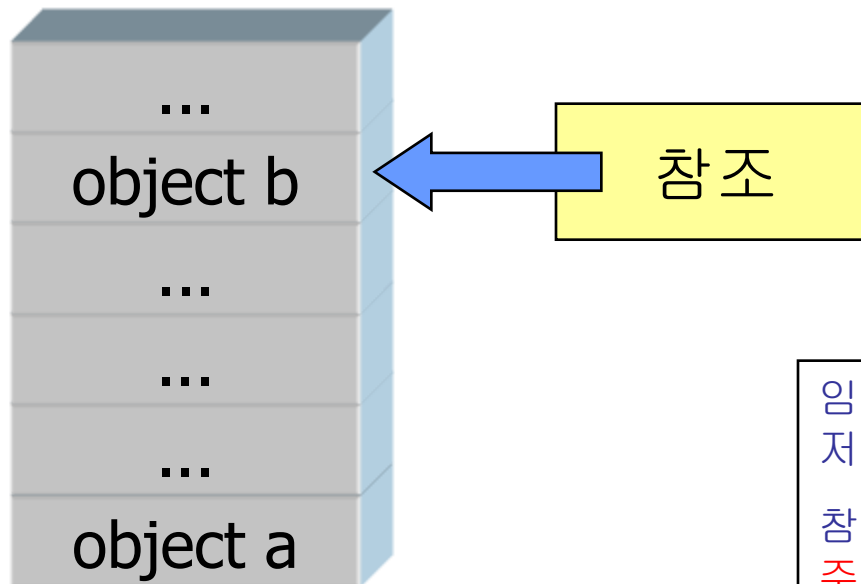


▶ 스택의 메모리 구조

데이터가 아래서부터 차곡차곡
쌓였다가,
제거될때는 맨 위의 데이터부
터 차례로 제거

스택(stack)/힙(heap)

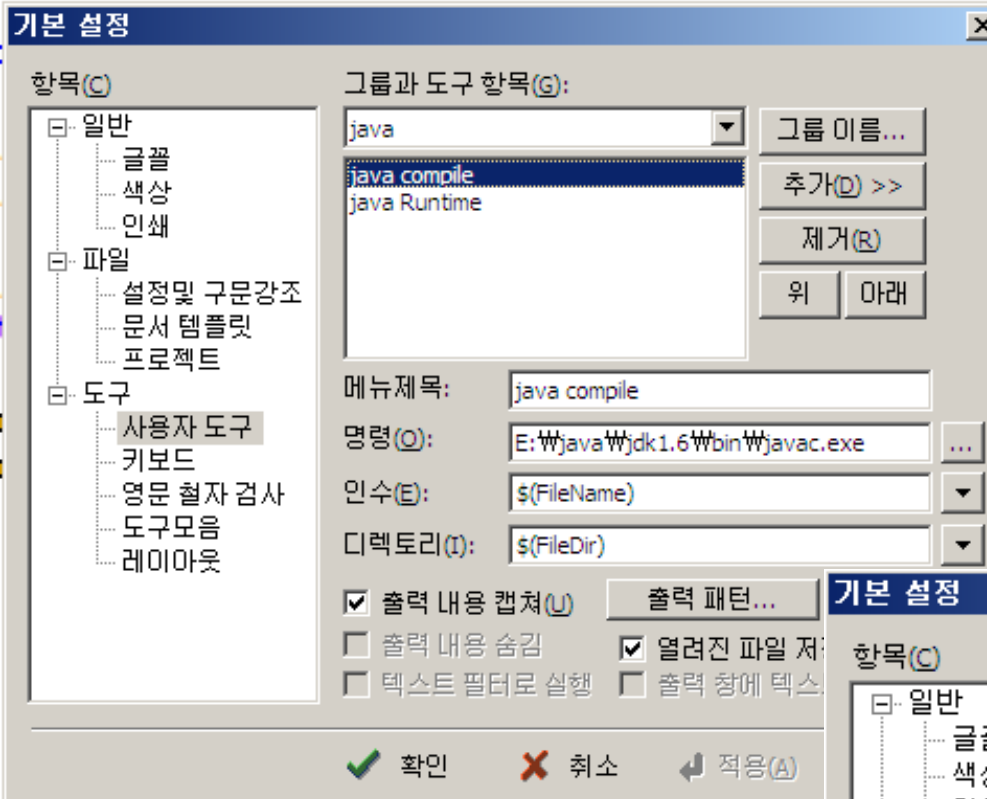
- 참조타입 - 힙 영역(자유 기억공간)에 데이터를 저장



임의의 메모리에 데이터를 자유롭게 저장하고 제거

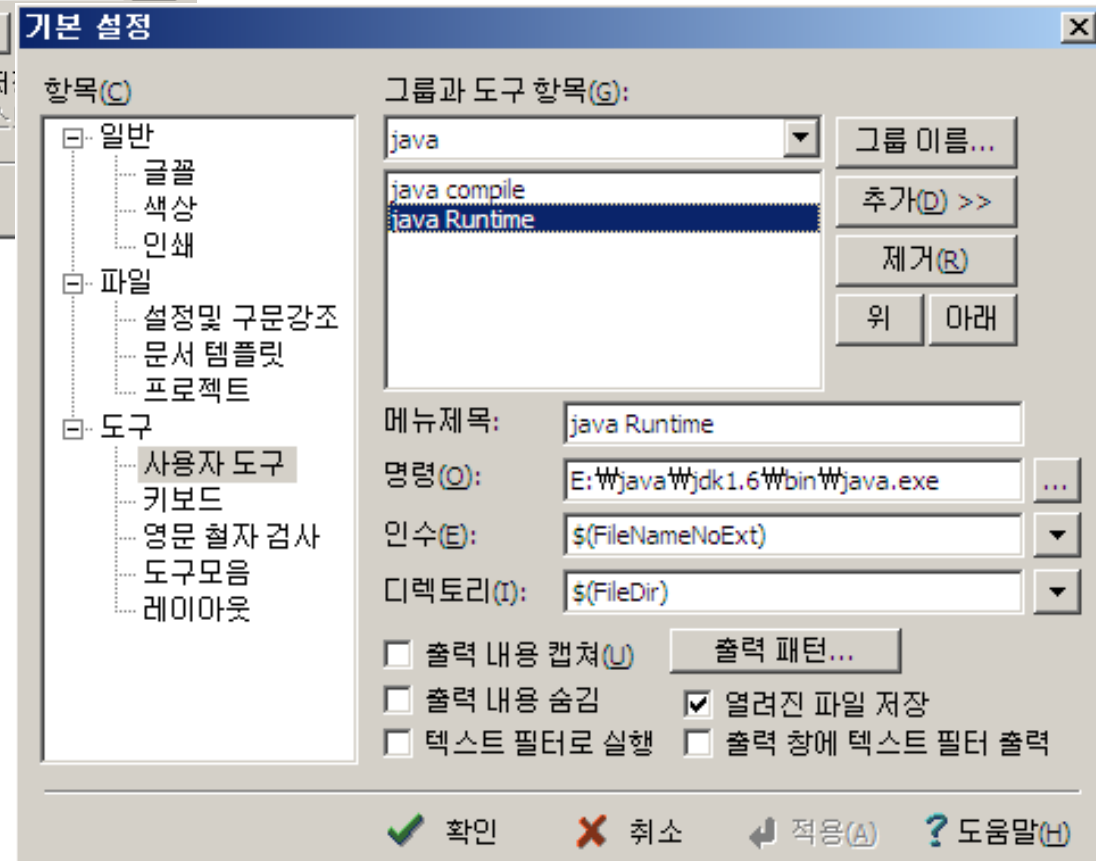
참조는 데이터가 저장되어 있는 힙의 주소를 기억하고 있다가 직접 데이터가 있는 곳에 접근

▶ 힙의 메모리 구조



도구 - 사용자 도구 구성

c:\ java\ jdk17\ bin\ javac.exe





형 변환



형변환 (Type Conversion)

■ 형변환

- 사람이 2진수 정수와 16진수 정수의 덧셈을 할 때, 둘 다 10진수로 변환을 해서 덧셈을 하거나 16진수를 2진수로 바꿔서 덧셈을 하듯이, CPU도 연산을 할 수 있도록 표현법(자료형)을 하나로 통일시켜 줘야 함
- 자동 형변환 - 연산의 대상이 되는 두 피연산자의 자료형이 일치하지 않아서 자동으로 발생하는 형 변환
- 명시적 형변환 - 명시적으로 형변환 연산자를 이용해서 발생시키는 형 변환

형 변환 (Type Conversion)

- 원래의 자료형이 아닌 다른 자료형으로 변환 되는 현상
- 연산 및 대입에서 발생
- [1] 연산
 - 서로 다른 데이터 형으로 계산을 할 때 Java는 더 큰 데이터 형을 사용해 계산함
 - 예) int 형/double 형 => double/double => 결과는 double 형
 - 연산의 우선 순위와 함께 다른 결과 유발
 - 예상치 못한 오류의 원인이 될 수 있으므로 주의
- [2] 대입
 - 변수에 값을 대입할 때나 변수끼리 대입할 때는 좌우변의 타입이 일치해야 함
 - 양쪽 타입이 같지 않은 경우 형변환을 통해 타입을 일치시킨 후 대입해야 함
 - 예) `int a=10;`
`long b = a; //자동형변환, int 인 a가 long으로`



형 변환 (Type Conversion)

- 형 변환 종류

- 암시적 변환(Implicit) - 자동 형변환

- 작은 범위의 데이터를 큰 범위에 할당하는 경우와 같이 값의 손실이 발생하지 않는 변환

- 명시적 변환(Explicit)

- 큰 범위의 데이터를 작은 범위에 할당하는 경우는 변환하고자 하는 자료형을 지정한다.
 - 값의 중요한 숫자가 손실되지 않을 때만 수행해야 함

형변환 - Cast 연산

```
int a=10; double b=3.78;  
double c = a*b;  
=> int * double => double*double
```

연산

■ 자동 형변환

- 코드에서 나타나지 않지만 **java**에서 자동으로 처리해주는 변환

```
예) int a = 123;  
long b = a; //작은 값(int)을 큰 범위(long)에 넣을 때 자동 형변환
```

대입

■ 명시적 형변환

- 코드를 이용한 형 변환

```
예) float a = 1.7f;  
int b = (int) a;
```

대입

```
int a=1; int b=2;  
float f = a/b; //결과값 :0  
=> float f = (float)a/b;
```

연산

■ 형식 - Cast 연산자 이용

자료형 변수 1 = (자료형) 변수2;

```
int age = 20;  
byte num = (byte)age;
```

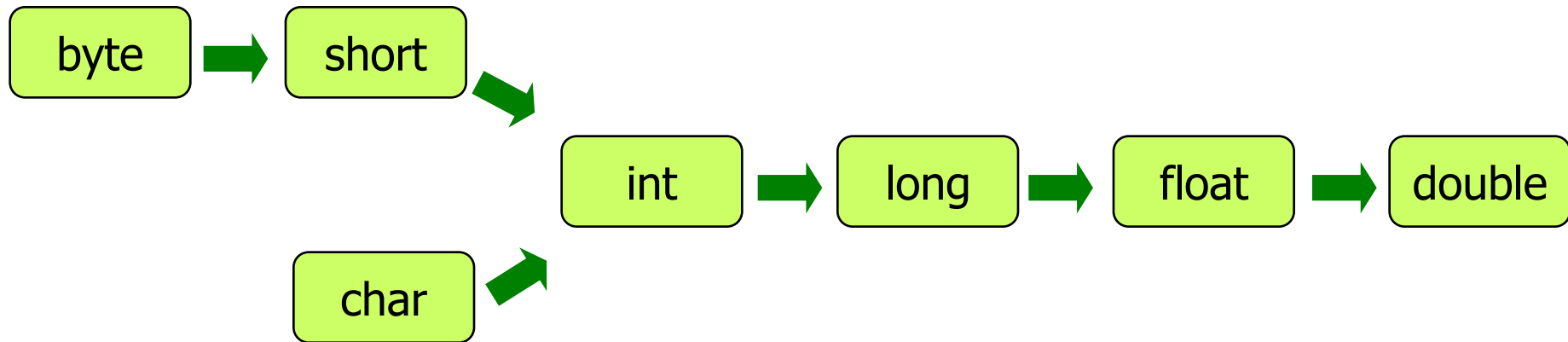



자동 형변환

- `double a = 10;` //정수 10이 10.0 으로 자동 형변환
- `int b=10.5;` //자동 형변환 발생 안함
 - 데이터의 손실이 발생하므로
 - 컴파일 에러
- `double c = 5.7f + 10;`
 - 2번의 형변환 발생
 - int형 데이터 10이 float 형으로 변환이 되어서 덧셈 연산이 진행됨
=> 5.7f+10.0f
 - `double c = 15.7f;` =>double형 상수 15.7로 형변환되어 변수 c에 저장됨
- 데이터의 손실이 발생하지 않거나, 발생하더라도 그 손실이 제한적인 경우에만 자동 형 변환을 허용

자동 형변환

■ 자동 형변환 규칙



- **boolean**을 제외한 나머지 7개의 기본형은 서로 형변환이 가능
- 기본형과 참조형은 서로 형변환할 수 없음
- 서로 다른 타입 변수간의 연산이나 대입에는 형변환 필수
- 값의 범위가 작은 타입에서 큰 타입으로의 형변환은 생략가능(자동 형변환)



명시적 형 변환

- 자동 형 변환 규칙에 위배되는 상황임에도 불구하고 형 변환이 필요한 경우에는 '명시적 형 변환'을 통해 형변환

```
int num=(int)3.15;
```

=> double형 3.15를 int형으로 형 변환



형변환

```
int a : 123, long b : 123
int age : 20, byte num : 20
float e : 1.7, int f : 1
float k : 0.0, float l : 0.5
float c : 3.142857
double d : 3.142857074737549
```

```
class ConversionTest{
    public static void main(String[] args)
    {
        //자동 형변환
        int a = 123;
        long b = a;

        //명시적 형변환
        int age = 20;
        byte num = (byte)age;

        float e = 1.7f;
        int f = (int) e;
        System.out.println("int a : " + a + ", long b : " + b);
        System.out.println("int age : " + age + ", byte num : " + num);
        System.out.println("float e : " + e + ", int f : " + f);

        int i=1; int j=2;
        float k = i/j; //결과값 :0.0
        float l = (float)i/j;
        System.out.println("float k : " + k + ", float l : " + l);

        float c = 22f/7f;
        double d = c;
        System.out.println("float c : " +c);
        System.out.println("double d : " +d);
    }
}
```



예제

```
long p : 2580, float q : 2580.0  
double r : 23.4891234567, float u : 23.489124
```

```
class ConversionTest2  
{  
    public static void main(String[] args)  
    {  
        long p = 2580L;  
        float q = p; //자동 형변환 가능  
        System.out.println("long p : "+p +", float q : "+ q);  
  
        double r = 23.489;  
        float u = (float)r; //명시적 형변환해야 함  
        System.out.println("double r : "+ r + ", float u : " + u);  
    }  
}
```

예제

문자 A의 유니코드 값: 65
문자 Z의 유니코드 값: 90
유니코드 97에 해당하는 문자 : a
유니코드 122에 해당하는 문자 : z

```
class CastingOperation  
{
```

```
    public static void main(String[] args)  
    {
```

```
        char ch1='A';  
        char ch2='Z';
```

char 형은 int형으로 자동 형변환 가능

```
        int num1=ch1; //char => int 자동형변환  
        int num2=(int)ch2;
```

```
        System.out.println("문자 A의 유니코드 값:" + num1);  
        System.out.println("문자 Z의 유니코드 값:" + num2);
```

```
        int n=ch1+1; //ch1 + 1 => char+int => int+int=>int, int로 자동 형변환되서 계산  
        char ch3=(char)(ch1+2);
```

```
        int a = 97;  
        char b = (char)a;  
        int c = 122;  
        char d = (char)c;  
        //char d = c; //에러, int => char 자동 형변환 안됨
```

```
        System.out.println("유니코드 "+a +"에 해당하는 문자 : " + b);  
        System.out.println("유니코드 "+c +"에 해당하는 문자 : " + d);
```

```
    }
```

```
}
```

예제

```
d= 30
z= 350
l= 8534400000000000
l2=-664862720
```

- CPU는 int 형 정수연산을 가장 고속으로 처리하게끔 설계되어 있음
- 따라서 정수형 연산을 진행할 때, 모든 피연산자를 int형으로 변환하는 과정을 거침

```
class ConversionTest3
```

```
{
```

```
    public static void main(String[] args)
    {
```

```
        byte a=10, b=20, c;
```

```
        //c=a+b; //에러 : 암시적으로 'int' 형식을 'byte' 형식으로 변환할 수 없다.
```

```
        int d=a+b;
```

```
        System.out.println("d= " +d);
```

```
        long x=300L;
```

```
        int y=50;
```

```
        //int z = x+y; //에러
```

```
        long z = x+y;
```

```
        System.out.println("z= " +z);
```

```
        byte i = 127;
```

```
        short j = 32000;
```

```
        int k = 2100000000;
```

```
        long l = (long)i * j * k;
```

```
        long l2 = (long)(i * j * k);
```

```
        System.out.println("l= " +l + "\n l2=" + l2);
```

```
}
```

- 피연산자가 int 보다 작으면 무조건 결과는 int에 담아야 함
- int 보다 큰 피연산자들의 연산은 큰 자료형에 담는다.



예제

```
<int> 9223372036854775807= -1  
<int> 495000= 495000
```

```
class ConversionTest4  
{  
    public static void main(String[] args)  
    {  
        long a = Long.MAX_VALUE; //long 타입의 max 값 (Long.MAX_VALUE)  
        int b = (int)a; //long 형의 최대값을 int형 변수에 넣어서 오버플로(Overflow) 발생  
        System.out.println("(int) " + a + " = " + b);  
  
        long c = 495000L;  
        int d = (int)c;  
        System.out.println("(int) " + c + " = " + d);  
    }  
}
```




형변환 정리

■ 1. 대입시

```
float a = 3.17f;  
int b = (int)a; //float => int
```

- 값의 범위가 큰 타입에서 작은 타입으로 변환될 때는 명시적 형 변환

■ 2. 연산시

```
byte a=10, b=20  
int c = a+b; //byte+byte => int+int => int
```

- [1] int형보다 크기가 작은 자료형은 int형으로 형변환 후에 연산을 수행

```
int a = 30;  
double b = 5.78;  
double c = a*b; //int*double => double*double =>double
```

- [2] int 형보다 큰 자료형은,
피연산자 중 자료형의 표현범위가 큰 쪽에 맞춰서 형변환 된 후 연산 수행



실습

- **byte**변수, **short** 변수 선언, 값 할당하고
결과값을 받는 변수를 선언하여 (**short**변수 - **byte** 변수) 연산하기
- **int** 변수, **long**변수 선언, 값 할당하고
결과값을 받는 변수를 선언하여 (**long**변수 * **int**변수) 연산하기
- **long**변수를 선언하여 값을 할당하고, **double**변수에 다시 할당하기
- **float**변수를 선언하여 값을 할당하고, **long**변수에 다시 할당하기
- **char** 변수를 선언하여 값을 할당하고, **int**변수에 다시 할당하기
- 출력하기



실습

- 소문자를 대문자로 변환하는 프로그램 작성하기
 - `char` 변수를 선언하여 소문자 'a' 를 할당하고, 이를 대문자 'A'로 변환해서 출력하기
 - 소문자 `a`=>97, 대문자 `A`=>65
 - 소문자와 대문자의 코드값의 차이는 32
 - 형변환 이용

```
a => 대문자로 변환 : A
```



실습

- float형 변수 pi의 값을 소수점 셋째 자리까지만 빼내서 출력하기
 - pi= 3.141592f
 - int / int 연산의 결과는 int
 - 나눗셈의 결과를 반올림하는 것이 아니라 버린다
 - 1) Pi 값에 1000을 곱한 후 int 로 형변환해서 소수 이하 자리수를 버린다
 - 2) 1의 결과값을 다시 1000으로 나눈다

형 변환- Integer.parseInt()

- 문자열 => 숫자 형 변환 : Integer.parseInt("123")

```
String a="12345";  
int b = (int) a; //에러
```

```
String a = "12345";  
int b = Integer.parseInt(a); //문자열을 정수로 변환
```

- Integer의 parseInt()메소드
 - String으로 부터 숫자를 분석해서 읽어올 수 있는 기능
 - 숫자의 문자열 표현을 해당하는 32비트 부호 있는 정수로 변환함.

```
Float.parseFloat()
```

```
Double.parseDouble()
```

```
Byte.parseByte()
```

```
public static int parseInt(String s)
```



형 변환- Integer.toString()

- 숫자형 => 문자열
 - Integer.toString(25)

```
int a = 10;  
String str = Integer.toString(a);
```

```
Double.toString();  
Long.toString();  
Float.toString();
```



명시적 형 변환

- 개발자가 분명하게 형의 변환을 표시하여 형 변환 유도
- 캐스트 연산자

(자료형) 변수명 ...

- 기본형 사이의 변환시 사용

- 수치자료형.parseInt() 메서드

문자열을 수치 데이터로 형변환시에 사용

예) **Integer.parseInt**(문자열)

- 자료형.toString() 메서드

수치형을 문자열로 형변환

예) **Integer.toString**(값)



형 변환

```
class ConversionTest6
{
    public static void main(String[] args)
    {
        String a = "31234";
        //int b = (int)a; //에러
        int b = Integer.parseInt(a); //문자열을 수치로 형 변환
        System.out.println("String a : " +(a+10));
        System.out.println("int b : " +(b+10));
    }
}
```


형변환 - toString() 메서드

```
class ConversionTest5
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        int i = 1234;
```

```
        String s = Integer.toString(i);
```

```
        System.out.println("int i=" + i + ", String s=" + s);
```

```
        double d = 34.157;
```

```
        s = Double.toString(d);
```

```
        System.out.println("String s=" + s);
```

```
        char c = 'A';
```

```
        s = Character.toString(c);
```

```
        System.out.println("String s=" + s);
```

```
        boolean b = true;
```

```
        s = Boolean.toString(b);
```

```
        System.out.println("String s="+ s);
```

```
    }
```

```
}
```

```
int i=1234, String s=1234
String s=34.157
String s=A
String s=true
```

콘솔 입력 - Scanner 클래스 이용

```
import java.util.*;
```

```
class ScannerTest  
{
```

```
    public static void main(String[] args)  
    {
```

```
        System.out.println("당신의 이름은 무엇입니까?");
```

```
        Scanner sc = new Scanner(System.in); //키보드 대상
```

```
        String name = sc.nextLine();
```

```
        System.out.println("반갑습니다." + name + "님!Wn");
```

```
        System.out.println("나이를 입력하세요");
```

```
        int age = sc.nextInt();
```

```
        System.out.println("나이는 "+age + "이군요!");
```

```
    }
```

```
}
```

```
당신의 이름은 무엇입니까?  
홍길동  
반갑습니다. 홍길동님!  
  
나이를 입력하세요  
20  
나이는 20이군요!
```



Scanner 클래스를 구성하는 다양한 메소드

- 읽어들이는 데이터의 유형에 따른 메소드 정의

- `public boolean nextBoolean()`
- `public byte nextByte()`
- `public short nextShort()`
- `public int nextInt()`
- `public long nextLong()`
- `public float nextFloat()`
- `public double nextDouble()`
- `public String nextLine()`



실습

- 사각형의 너비(가로)와 높이(세로)를 입력 받아 넓이를 계산하여 출력하시오
 - 사각형의 면적 = 가로 * 세로
 - `nextLine()` 이용하여 형변환 해보기

```
사각형의 너비를 입력하세요:50
사각형의 높이를 입력하세요:30
사각형 가로 : 50
사각형 세로 : 30
사각형 면적 : 1500
```



실습

```
지방의 그램을 입력하세요:10
탄수화물의 그램을 입력하세요:123
단백질의 그램을 입력하세요:24

총 칼로리 : 678
```

- 지방(fat), 탄수화물(carbohydrate), 단백질(protein) 칼로리의 합계를 계산하는 프로그램 작성
 - 사용자가 지방, 탄수화물, 단백질의 그램을 입력
 - 총 칼로리(calory) 구하기
 - 지방 1그램은 9칼로리, 단백질과 탄수화물은 1그램에 4칼로리
 - 총 칼로리 = 지방 * 9 + 단백질 * 4 + 탄수화물 * 4
 - 화면 출력
 - nextInt() 이용
- kcal 구해보기(결과가 실수가 나오도록)
 - 총 칼로리/1000

실수에 대한 e 표기법과 정수에 대한 16진수 8진수 표기법

- 자바는 소수부가 큰 실수 표현의 편의를 위해 e표기법을 지원
- 데이터의 성격에 적절한 정수의 표현을 위해 16진수와 8진수 표현을 지원함

class ENotation

{

public static void main(String[] args)

{

double d1 = 1.5e-3; //e표기법, $1.5 \times 10^{-3} \Rightarrow 0.0015$

double d2 = 1.5E+3; // $1.5 \times 10^3 \Rightarrow 1500.0$

double d3 = 1.5E3; // 1.5×10^3

int n1=0xA0E; //16진수 - 0x 로 시작하면 16진수 표현 $\Rightarrow 10 \times 256 + 0 \times 16 + 14 \Rightarrow$ 십진수 2574

int n2 = 0125; //8진수 - 0로 시작하면 8진수 표현 $\Rightarrow 1 \times 64 + 2 \times 8 + 5 \Rightarrow$ 십진수 85

System.out.println("d1="+d1+", d2="+d2+", d3="+d3);

System.out.println("n1="+n1+", n2="+n2);

}

}

d1=0.0015, d2=1500.0, d3=1500.0
n1=2574, n2=85

유니코드

- 자바는 유니코드라는 표준을 근거로 문자를 표현함
- 유니코드
 - 문자 하나를 2바이트로 표현하는 문자 체계
 - 65,536 개의 문자 표현이 가능
 - 세계의 모든 언어를 표현할 수 있는 문자 체계
 - <http://unicode.org/charts/index.html> - 유니코드 정보를 알 수 있다
 - 유니코드라는 하나의 문자 체계를 가지고 세계 모든 나라의 언어를 표현할 수 있기 때문에, 자바의 유니코드 지원은 한글을 영어와 동일한 수준으로 컨트롤할 수 있다는 장점으로 이어짐

한글 유니코드의 일부

	AC0	AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8	AC9	ACA	ACB	ACC	ACD	ACE	ACF
0	가 AC00	감 AC10	갸 AC20	갹 AC30	갈 AC40	각 AC50	갬 AC60	거 AC70	검 AC80	겐 AC90	겉 ACA0	결 ACB0	격 ACC0	겻 ACD0	고 ACE0	곰 ACF0
1	각 AC01	갑 AC11	갸 AC21	갹 AC31	갈 AC41	각 AC51	갬 AC61	거 AC71	검 AC81	겐 AC91	겉 ACA1	결 ACB1	격 ACC1	겻 ACD1	곡 ACE1	곱 ACF1
2	각 AC02	갑 AC12	갸 AC22	갹 AC32	갈 AC42	각 AC52	갬 AC62	거 AC72	검 AC82	겐 AC92	겉 ACA2	결 ACB2	격 ACC2	겻 ACD2	곡 ACE2	곱 ACF2

일본어(히라가나) 유니코드의 일부

	304	305	306	307	308	309
0		ぐ 3050	だ 3060	ば 3070	む 3080	み 3090
1	あ 3041	け 3051	ち 3061	ぱ 3071	め 3081	ゑ 3091
2	あ 3042	げ 3052	ぢ 3062	ひ 3072	も 3082	を 3092



예제

```
ch1=A, ch2=가  
ch3=공, ch4=†, ch5=†
```

```
class CharTest  
{  
    public static void main(String[] args)  
    {  
        char ch1='A';  
        char ch2='가';  
        char ch3=0xACF0;  
        char ch4=0x3051;  
        char ch5='\ u3051';  
  
        System.out.println("ch1="+ch1+", ch2="+ch2);  
        System.out.println("ch3="+ch3+", ch4="+ch4+", ch5="+ch5);  
    }  
}
```