A Tour of Java

Sungjoo Ha

March 6th, 2015

Introduction

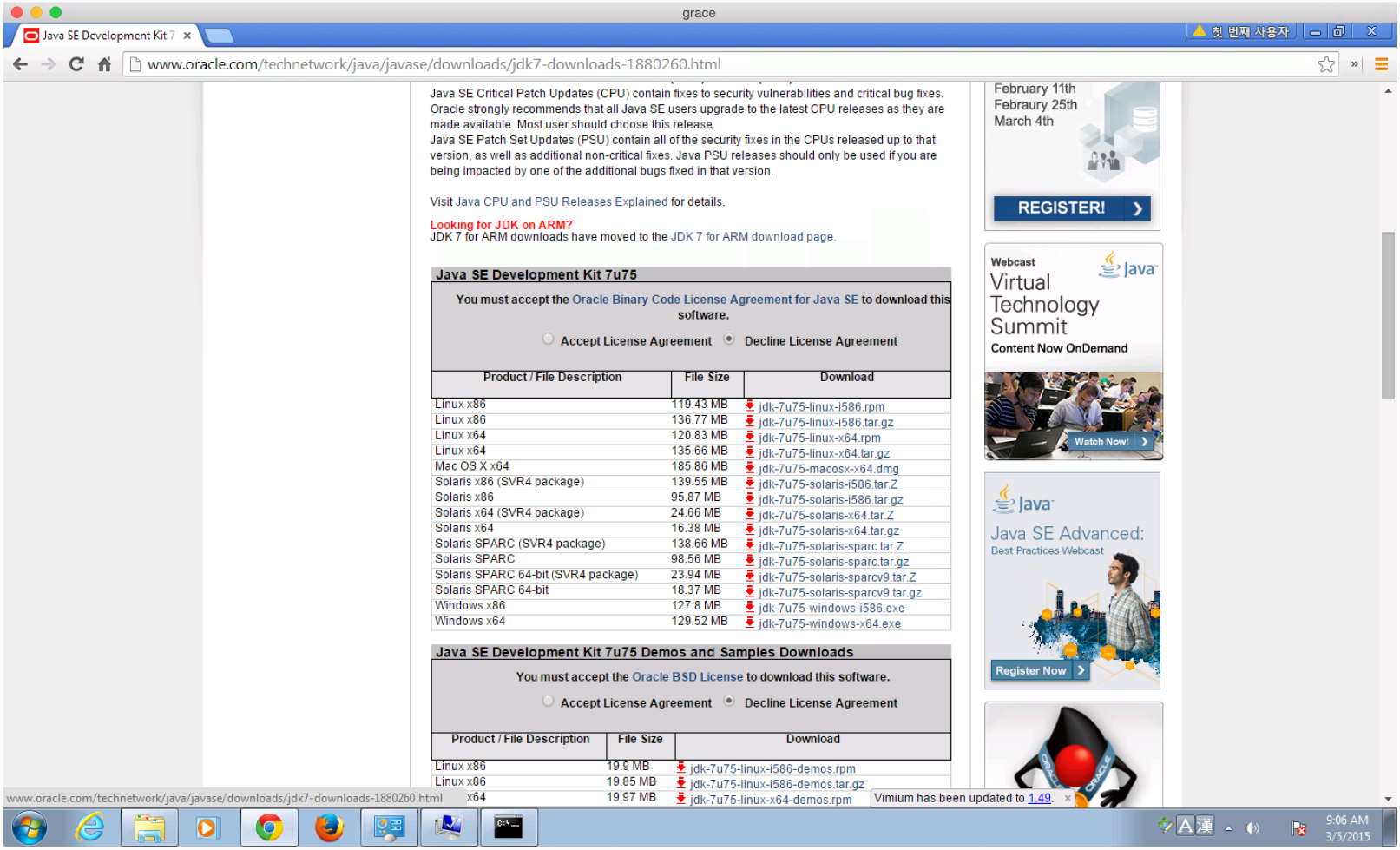
- 앞으로의 계획 (바뀔 수 있음)
  - 환경 구축 및 자바의 기초 소개
  - 사용자 정의 타입 및 모듈화, 프로그래밍 일반론, 디버깅, 등
  - 클래스
  - 제네릭
- 수업 후 실습실에서 실습 예정

First Principle

- 문제가 생기면 침착하게 (영어로) 구글에서 찾아본다.
  - 오류 메시지를 입력한다.
  - 증상을 설명한다.
- 대체로 stackoverflow 등의 사이트에 답변이 있다.

Java

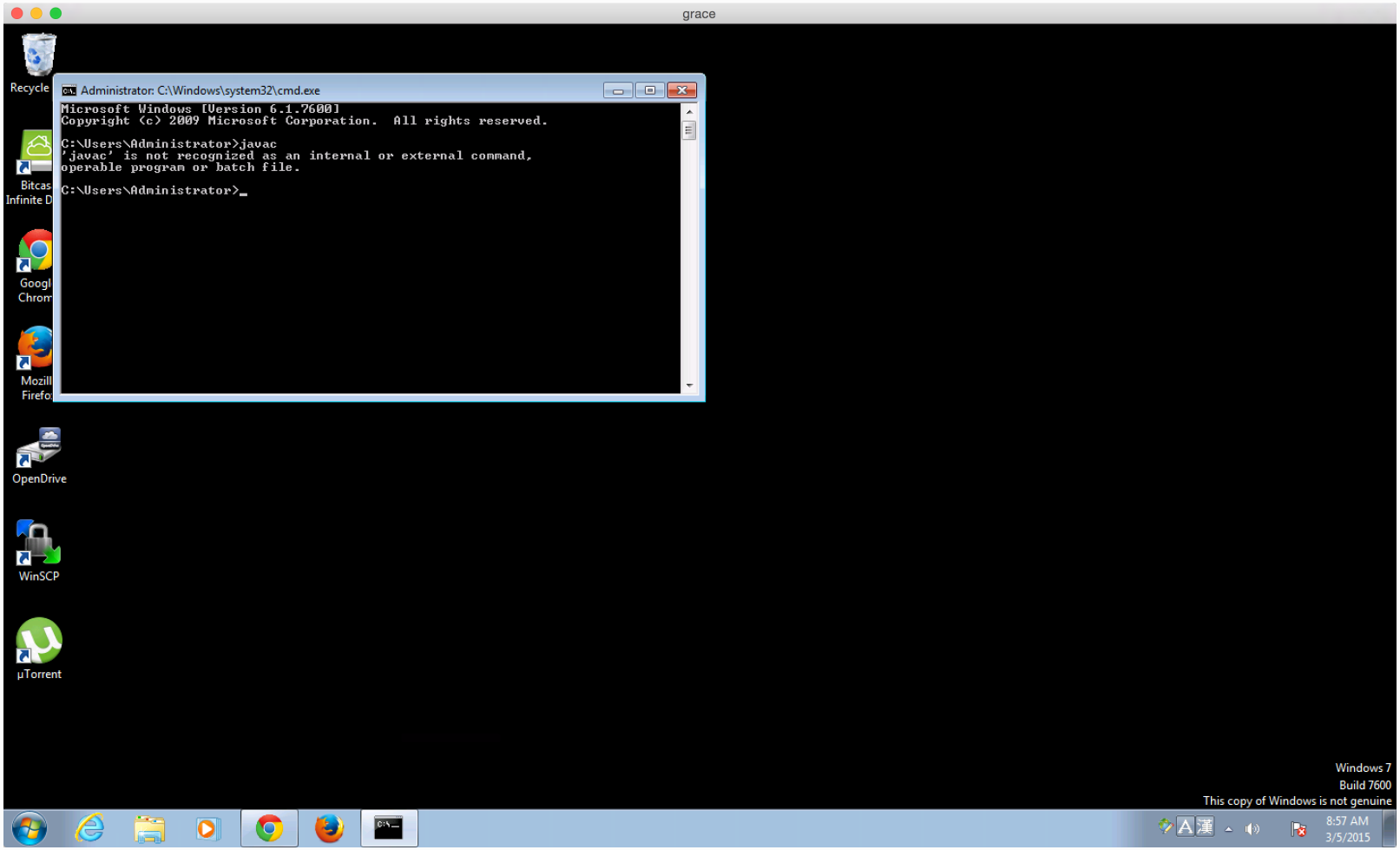
- 최신 버전은 8이지만 채점 서버가 7u75로 되어 있다.
- 자바7과 자바8은 차이가 있으니 염두에 두기 바란다.
- `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`
- JRE가 아니라 JDK를 받아야 한다.

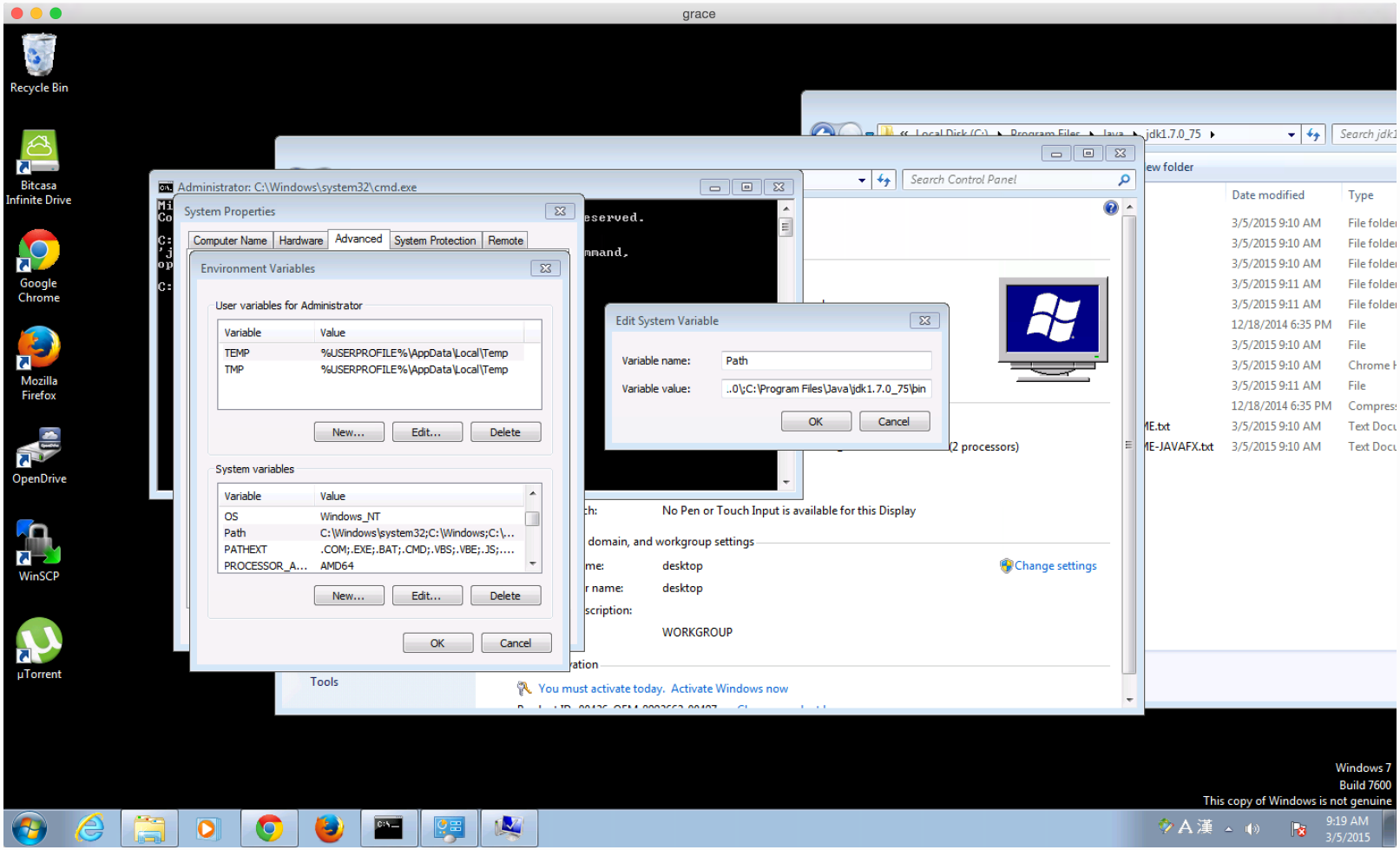


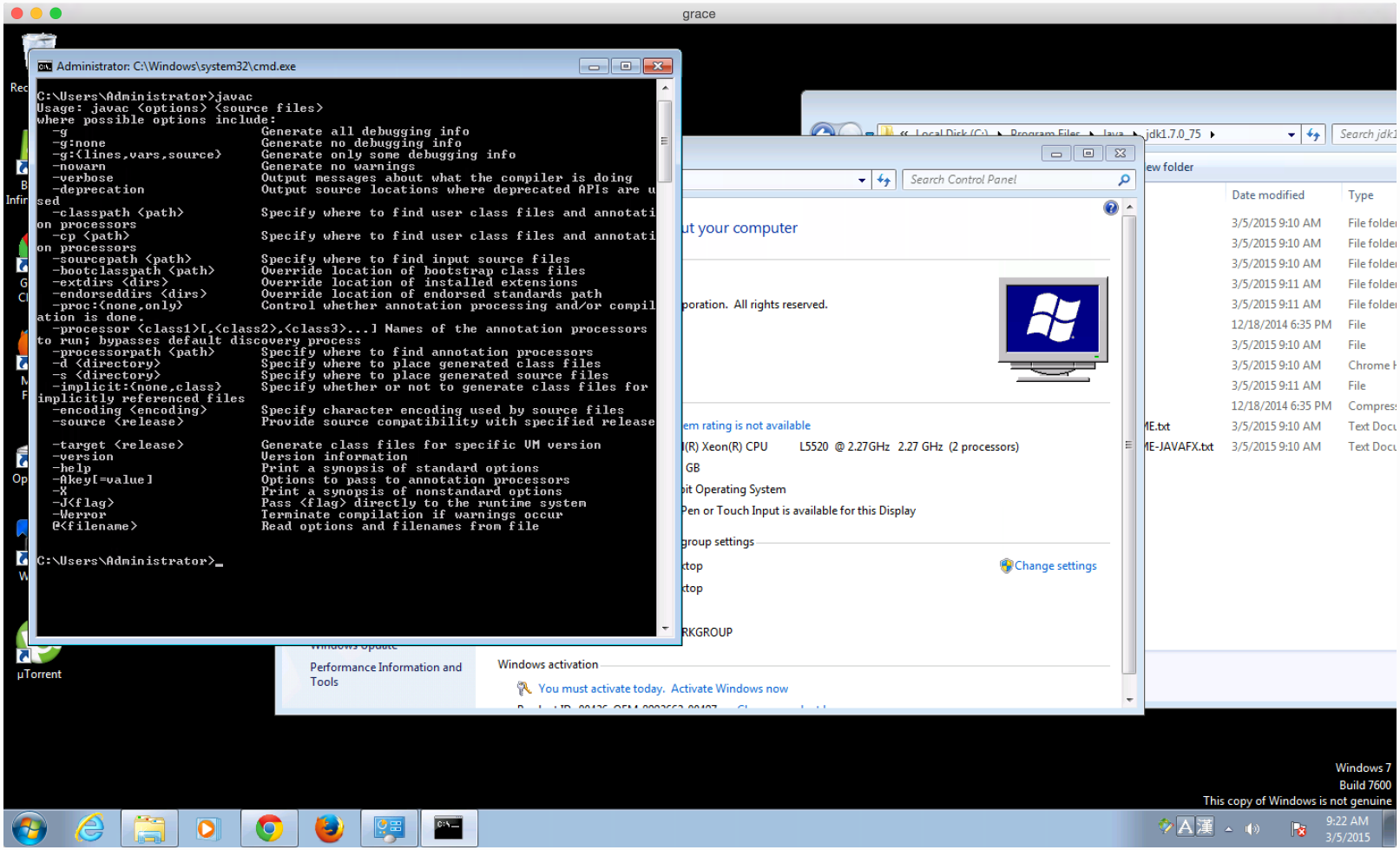
윈도우는 [내 컴퓨터 - 속성]에서 x86인지 x64인지 확인 가능하다.

Path

- 자바 실행을 위해 PATH 설정을 해줘야 한다.
- 제어판 - 시스템 - 고급 설정 - 환경 변수


grace
Recycle
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\Administrator>javac
'javac' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\Administrator>_
OpenDrive
WinSCP
µTorrent
Windows 7
Build 7600
This copy of Windows is not genuine
8:57 AM
3/5/2015


grace
Recycle Bin
Bitcasa Infinite Drive
Google Chrome
Mozilla Firefox
OpenDrive
WinSCP
µTorrent
Administrator: C:\Windows\system32\cmd.exe
System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
Environment Variables
User variables for Administrator
Variable
Value
TEMP
%USERPROFILE%\AppData\Local\Temp
TMP
%USERPROFILE%\AppData\Local\Temp
New...
Edit...
Delete
System variables
OS
Windows_NT
Path
C:\Windows\system32;C:\Windows;C:\...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...
AMD64
OK
Cancel
Edit System Variable
Variable name:
Path
Variable value:
..0\;C:\Program Files\Java\jdk1.7.0_75\bin
Search Control Panel
Date modified
Type
Tools
No Pen or Touch Input is available for this Display
domain, and workgroup settings
Change settings
WORKGROUP
You must activate today. Activate Windows now
Windows 7
Build 7600
This copy of Windows is not genuine
9:19 AM
3/5/2015

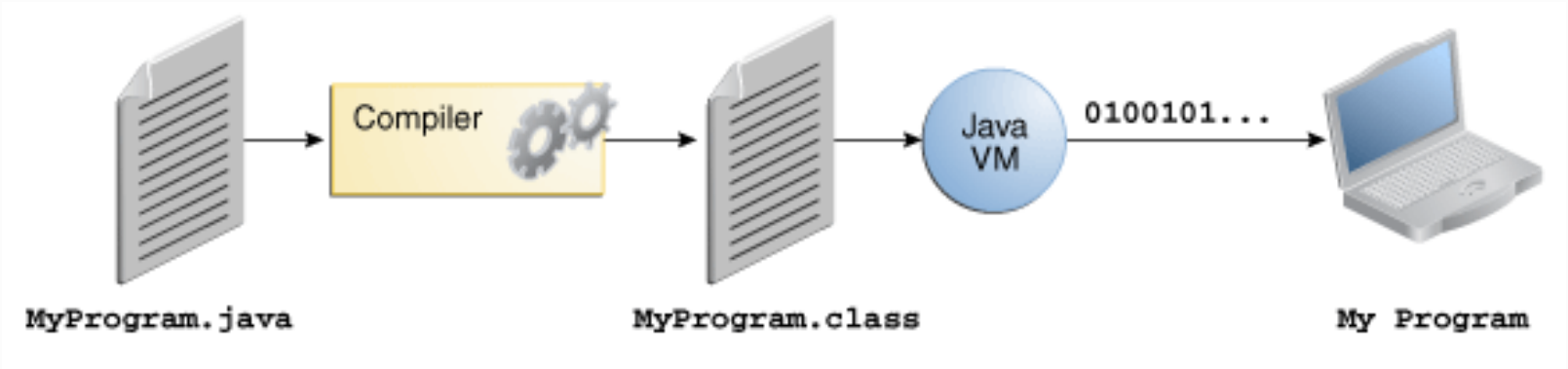

grace
Administrator: C:\Windows\system32\cmd.exe
C:\Users\Administrator>javac
Usage: javac <options> <source files>
where possible options include:
-g Generate all debugging info
-g:none Generate no debugging info
-g:<lines,vars,source> Generate only some debugging info
-nowarn Generate no warnings
-verbose Output messages about what the compiler is doing
-deprecation Output source locations where deprecated APIs are u
sed
-classpath <path> Specify where to find user class files and annotati
on processors
-cp <path> Specify where to find user class files and annotati
on processors
-sourcepath <path> Specify where to find input source files
-bootclasspath <path> Override location of bootstrap class files
-extdirs <dirs> Override location of installed extensions
-endorseddirs <dirs> Override location of endorsed standards path
-proc:<none,only> Control whether annotation processing and/or compil
ation is done.
-processor <class1>[,<class2>,<class3>...] Names of the annotation processors
to run; bypasses default discovery process
-processorpath <path> Specify where to find annotation processors
-d <directory> Specify where to place generated class files
-s <directory> Specify where to place generated source files
-implicit:<none,class> Specify whether or not to generate class files for
implicitly referenced 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
-version Version information
-help Print a synopsis of standard options
-Akey[=value] Options to pass to annotation processors
-X Print a synopsis of nonstandard options
-J<flag> Pass <flag> directly to the runtime system
-Werror Terminate compilation if warnings occur
@<filename> Read options and filenames from file
C:\Users\Administrator>_
Search Control Panel
ut your computer
poration. All rights reserved.
em rating is not available
(R) Xeon(R) CPU L5520 @ 2.27GHz 2.27 GHz (2 processors)
GB
it Operating System
Pen or Touch Input is available for this Display
group settings
Change settings
RKGROUP
Performance Information and Tools
Windows activation
You must activate today. Activate Windows now
jdk1.7.0_75
Search jdk1
ew folder
Date modified
Type
3/5/2015 9:10 AM File folder
3/5/2015 9:10 AM File folder
3/5/2015 9:10 AM File folder
3/5/2015 9:11 AM File folder
3/5/2015 9:11 AM File folder
12/18/2014 6:35 PM File
3/5/2015 9:10 AM File
3/5/2015 9:10 AM Chrome H
3/5/2015 9:11 AM File
12/18/2014 6:35 PM Compres
ME.txt 3/5/2015 9:10 AM Text Docu
ME-JAVAFX.txt 3/5/2015 9:10 AM Text Docu
µTorrent
Windows 7
Build 7600
This copy of Windows is not genuine
9:22 AM
3/5/2015

Eclipse

- 통합 개발 환경 (IDE)
- `https://eclipse.org/`

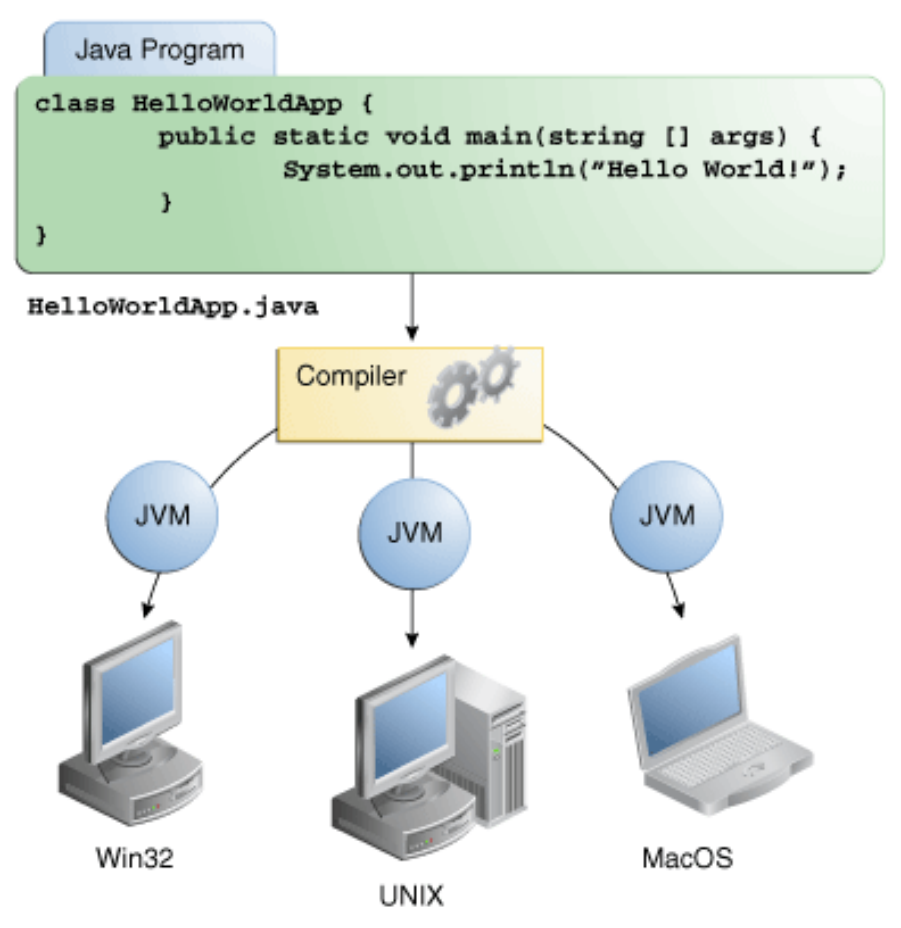
JVM

- 자바는 컴파일 되어야 하는 언어이다.
- 프로그램이 수행되기 위해 소스 텍스트가 컴파일러를 거쳐서 바이트코드가 생성된다.
- 이 바이트코드는 JVM이 이해할 수 있는 언어로 되어 있고, JVM이 이를 실행한다.



JVM

- JVM만 설치 되어 있으면 .class 파일을 어떤 환경에서나 실행할 수 있다.



Java Platform

- 플랫폼은 프로그램이 수행되는 환경이다.
- 자바 플랫폼은 두 개로 나뉜다. JVM과 Java API이다.
  - JVM은 이미 소개하였다.
  - API는 이미 만들어진 프로그램 집합이다.
  - API를 내가 만든 프로그램에서 사용할 수 있다.

Hello, World!

```
public class HelloWorld {
    // 최소한의 자바 프로그램
    public static void main(String[] args) {}
}
```

- *main* 메소드를 정의하고 있다. *args* 라는 커맨드라인 인자를 받는다.
- 메소드는 다른 언어의 함수와 비슷한 위상이라고 생각해도 좋다.
- *public*, *static*, *class*는 무시한다.
- 클래스 이름 *HelloWorld* 는 파일 이름 *HelloWorld.java*와 같아야 한다.

Hello, World!

```java
public class HelloWorld {
    // 최소한의 자바 프로그램
    public static void main(String[] args) {}
}
```

- 중괄호 {, }는 그룹을 나누는 것을 의미한다. 클래스와 메소드의 시작과 끝을 표현한다.
- "//" 는 주석을 의미한다. 시작부터 줄 마지막까지 적용된다. 주석은 사람이 읽기 위해 존재한다.
- 모든 자바 프로그램은 *main* 메소드를 딱 하나 갖고 있어야 한다. 여기부터 프로그램이 시작한다.
- *void*는 *main* 프로그램의 리턴 타입을 알려주는데, 아무것도 리턴하지 않음을 뜻한다.

Hello, World!

```java
import java.lang.System;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.print("Hello, World!\n");
    }
}
```

- 보통 프로그램은 결과물을 내놓게 마련이다.
- 위의 프로그램은 'Hello, World!'를 출력하는 프로그램이다.

Hello, World!

```java
import java.lang.System;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.print("Hello, World!\n");
    }
}
```

- *import java.lang.System;*은 컴파일러에게 *java.lang.System*의 내용을 포함하라고 지시한다.
- 여기에서는 화면에 출력하기 위한 standard output stream *out*을 활용하기 위해 포함되었다.
- 하지만 기본적으로 *java.lang*의 모든 패키지가 *import*(추가)된다. 그러니 위의 *import*는 필요 없다.

Hello, World!

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.print("Hello, World!\n");
    }
}
```

- 위의 프로그램은 *out*의 *print* 메소드를 호출한다.
- 이 때 인자는 *"Hello, World!\n"* 이다.
- 따옴표로 묶인 문자들의 나열을 문자열 (string) 이라 부른다.
- 역슬래시 \ 와 하나의 문자는 특수 문자이다. *n*은 newline 문자이며 그래서 *Hello, World!*를 출력한 뒤 다음 줄로 넘어가게 된다.

Hello, World!

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

- 새 줄로 넘어가는 기능은 무척 많이 사용하므로 따로 메소드가 있다.

Method Call

```
public class Square {
    static double square(double x) {
        return x*x;
    }
    static void print_square(double x) {
        System.out.println("the square of " + x
                + " is " + square(x));
    }
    public static void main(String[] args) {
        print_square(1.234);
    }
}
```

- 사실상 실행되는 모든 코드는 어딘가의 메소드로 존재하며, *main* 메소드에서 직간접적으로 호출되게 된다.
- *square* 메소드의 앞 쪽 *double*은 리턴 타입이 *double*형임을 알려준다.

Method

- 자바에서 무언가 하는 주된 방법은 클래스에 속한 메소드를 호출하는 것이다.
- 메소드를 정의해서 하고 싶은 일을 설명하게 된다.
- 메소드의 정의는 메소드에 이름을 달아주고, 호출 시 필요한 인자의 개수 및 타입을 알려주며 리턴값의 타입도 알려준다.

Method

- 우리는 대체로 코드가 이해하기 쉽길 원한다. 그래야 유지보수가 쉽다.
  - 이를 위한 첫 번째 단계는 각종 계산을 이해 가능한 수준의 작은 조각으로 자르는 것이다.
  - 이런 작은 조각은 메소드로 (및 클래스) 표현된다.
- “버그”는 프로그램의 복잡도와 길이에 비례하게 된다.
  - 프로그램을 작은 조각으로 나누면 두 문제 다 피할 수 있다.
  - 특정 코드를 메소드로 만들면 해당 코드가 어떤 일을 하는지 이름 붙이게 된다. 이해에 도움이 된다.
  - 같은 코드를 여러 번 작성하지 않아도 되게 한다.

Overloading

```
public class Print {
    static void print(double num) {
        System.out.println("double");
        System.out.println(num);
    }
    static void print(String str) {
        System.out.println("string");
        System.out.println(str);
    }
    public static void main(String[] args) {
        print(3.23);
        print("한글");
    }
}
```

- 두 메소드가 같은 이름을 갖지만 다른 종류의 인자를 가지면 컴파일러가 적당한 메소드를 선택해서 호출해준다.

Overloading

```
public class Print2 {
    static void print(int num1, double num2) {
        System.out.println(num1);
        System.out.println(num2);
    }
    static void print(double num1, int num2) {
        System.out.println(num1);
        System.out.println(num2);
    }
    public static void main(String[] args) {
        //print(0, 0); // compile error
    }
}
```

- 만약 두 메소드가 동등하게 적합하다면 컴파일 오류가 난다.

Overloading

- 이렇게 같은 메소드 이름을 사용하는 기법을 오버로딩(overloading)이라고 한다. 이후에 다시 다루게 될 것이다.
- 같은 이름을 갖는 메소드는 같은 “의미”를 구현하여야 한다.
- 가령 *print* 함수라면 “출력” 하는 의미를 갖도록 구현하자.

Type

```
int cm;
```

- 모든 종류의 이름과 표현식(expression)은 타입이 달려 있다.
- 타입에 의해 우리가 할 수 있는 행동이 결정된다.
- *cm*은 *int* 타입을 가짐을 뜻한다. *cm*은 정수형(int) 변수이다.
- 선언(declaration)은 프로그램서 사용할 이름을 선언하는 행위이다.
  - 이는 이름의 타입을 지정하게 된다.
  - 타입은 가능한 값의 집합과 연산의 집합을 정의한다.
  - 객체(object)는 어떤 타입의 값을 쥐고 있는 메모리이다.
  - 값(value)은 타입에 따라 해석되는 비트이다.
  - 변수(variable)는 이름이 달린 객체이다.

Primitive Data Type

```
public class Types {
    public static void main(String[] args) {
        boolean B = true;
        char 문자 = '가';
        int answer = 42;
        double pi = 3.141592;
    }
}
```

- 자바는 다양한 기본 데이터 타입(primitive data type)을 제공한다.
- 가령 *char* 변수는 문자 하나를 저장하는 변수가 된다.
- 자세한 내용은 `http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html`을 참고한다.
- 변수의 초기화(initialization)는 등호(=)를 사용한다.
- 변수는 선언과 동시에 초기화하기를 강력히 권고한다.

Arithmetic Operator

```java
public class Arithmetic {
    public static void main(String[] args) {
        int x = 10;
        int y = -3;

        System.out.println(x + y);
        System.out.println(+y);
        System.out.println(x - y);
        System.out.println(-x);
        System.out.println(x * y);
        System.out.println(x / y);
        System.out.println(x % y);
    }
}
```

- 산술 연산자를 적합한 타입의 조합과 함께 사용할 수 있다.

Comparison Operator

```
public class Comparison {
    public static void main(String[] args) {
        int x = 10;
        int y = -3;

        System.out.println(x == y);
        System.out.println(x != y);
        System.out.println(x < y);
        System.out.println(x > x);
        System.out.println(x <= y);
        System.out.println(x >= y);
    }
}
```

- 비교 연산자도 사용할 수 있다.

Bitwise Logical Operator

```
public class Logical {
    public static void main(String[] args) {
        int x = 10;
        int y = -3;

        System.out.println(x & y);
        System.out.println(x | y);
        System.out.println(x ^ y);
        System.out.println(~x);
    }
}
```

- 비트 단위 논리 연산자(bitwise logical operator)도 있다.
- 각 인자의 비트 단위로 연산을 적용한다.

Logical Operator

```
public class Logical {
    public static void main(String[] args) {
        boolean a = true;
        boolean b = false;

        System.out.println(a && b);
        System.out.println(a || b);
    }
}
```

- 논리 연산은 참(*true*) 혹은 거짓(*false*)를 리턴한다.

Assignments

```
public class Arithmetic2 {
    public static void main(String[] args) {
        int x = 10;
        int y = -3;
        x += y;
        ++x;
        x-=y;
        --x;
        x*=y;
        x/=y;
        x%=y;
        System.out.println(x);
        System.out.println(y);
    }
}
```

- 기본적인 산술 연산 및 논리 연산 외에 변수를 변경하는 연산자도 제공한다.

Local Scope

```
public class Scope {
    public static void main(String[] args) {
        int x = 10;
        System.out.println(x);
        {
            System.out.println(x);
            int a = 10;
            System.out.println(a);
        }
        //System.out.println(a); // compile error
    }
}
```

- 메소드 내부에서 선언된 변수는 선언 시점부터 블럭의 끝까지를 스코프로 가진다.
- 블럭은 중괄호로 표현된다.
- 함수 인자는 로컬 스코프로 판정된다.

Class Scope

```
public class Scope2 {
    String name;

    public static void main(String[] args) {
        Scope2 s = new Scope2();
        s.name = "이름";

        System.out.println(s.name);
    }
}
```

- 클래스 내부에서 선언되면 클래스 스코프를 갖는다.
- 객체(object)는 생성(construct)되어야 사용할 수 있다.
- 이렇게 생성된 객체는 스코프가 끝나면 파괴된다.
- 맴버는 객체가 파괴될 때 함께 파괴된다.
- 나중에 클래스를 다룰 때 다시 논하기로 한다.

Constants

```
public class Final {
    static int sum(final int a, final int b) {
        // a = 3; // compile error
        return a + b;
    }

    public static void main(String[] args) {
        final int a = 10;
        //a = 2; // compile error

        int b = sum(3, 4);
        System.out.println(b);
    }
}
```

- 불변성을 표현하기 위해 *final* 키워드를 사용한다.
- 대략 “이 값을 변화하지 않겠음” 정도의 의미로 사용된다. .

Array

```
public class Array {
    public static void main(String[] args) {
        char [] v = new char[5];
        v[0] = '한';
        v[1] = '글';
        v[2] = '문';
        v[3] = '자';
        v[4] = '열';
        char [] v2 = {'a', 'b', 'c'};
    }
}
```

- *[]* 는 배열을 의미한다.
- *char [] v* 는 문자형 배열 타입을 갖는 변수 *v*를 의미하게 된다.
- 배열의 인덱스는 0부터 시작한다.
- 초기화는 *new* 키워드를 사용하거나 암묵적으로 초기화할 수 있다.

Array

```
public class Array2 {
    public static void main(String[] args) {
        int[] v1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int[] v2 = new int[10];
        for (int i=0; i < 10; ++i) {
            v2[i] = v1[i];
        }
        System.out.println(v2[3]);
    }
}
```

- *for*는 루프문을 의미한다.

Array

```
public class Array3 {
    public static void main(String[] args) {
        int[] v1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        for (int i: v1) {
            System.out.println(i);
        }
    }
}
```

- *for each*문도 있다.

Pass By Value

```
public class Pass {
    static void swap(int a, int b) {
        // does not work as intended
        int temp = a;
        a = b;
        b = temp;
    }
    public static void main(String[] args) {
        int x = 1;
        int y = 2;
        swap(x, y);
        System.out.println(x);
        System.out.println(y);
    }
}
```

Pass By Value

기본형은 값의 복사가 일어난다.

- *swap(x, y)*는 *x*와 *y*의 값을 각각 *a*와 *b*에 복사한다.
- 그러므로 *a*와 *b*가 변해도 *x*와 *y*는 변하지 않는다.

Pass By Value

```
public class Ref {
    static void setZero(int[] v) {
        v[0] = 10;
        return;
    }
    public static void main(String[] args) {
        int[] v1 = {4, 3, 2, 1, 0};
        for (int i: v1) {
            System.out.println(i);
        }
        setZero(v1);
        for (int i: v1) {
            System.out.println(i);
        }
    }
}
```

Pass By Value

기본형이 아니면 이름표가 걸린다.

- 기본형이 아니라면 변수명은 이름표처럼 동작한다.
- *v1*은 *[4, 3, 2, 1, 0]*의 데이터가 저장된 메모리 공간에 이름표로 걸린다.
- *setZero* 메소드 호출의 인자로 *v1*을 넘기면 이 이름표가 걸린 메모리 공간에 *v* 이름표도 걸린다.
- *v[0] = 10* 은 해당 메모리 공간을 수정하게 된다.
- 그러므로 값이 변경된다.

Pass By Value

```
public class Ref {
    static void newArray(int[] v) {
        v = new int[5];
        v[0] = 3;
        for (int i: v) {
            System.out.println(i);
        }
    }
    public static void main(String[] args) {
        int[] v1 = {4, 3, 2, 1, 0};
        newArray(v1);
        for (int i: v1) {
            System.out.println(i);
        }
    }
}
```

Pass By Value

기본형이 아니면 이름표가 걸린다.

- *newArray* 메소드는 *v* 이름표가 *v1* 이름표가 걸려 있던 메모리 공간에 같이 걸린다.
- 다시 *v*는 *new int[5]*로 생성된 메모리 공간에 걸린다.
- 새로운 공간의 0번과 1번 위치의 값을 3으로 고쳐 쓴다.
- *v1*은 여전히 *[4, 3, 2, 1, 0]*을 가리키므로 변화는 없다.

Conditional

IF

```
import java.util.Scanner;
public class Input {
    static boolean accept() {
        System.out.println("진행할까요? (y/n)");
        Scanner reader = new Scanner(System.in);
        String answer  = reader.nextLine();
        if (answer.charAt(0) == 'y' || answer.charAt(0) == 'Y') {
            return true;
        }
        else {
            return false;
        }
    }
    public static void main(String[] args) {
        boolean answer = accept();
        System.out.println(answer);
    }
}
```

Conditional

SWTICH

```
import java.util.Scanner;
public class Input2 {
    static boolean accept() {
        System.out.println("진행할까요? (y/n)");
        Scanner reader = new Scanner(System.in);
        String answer  = reader.nextLine();
        switch(answer.charAt(0)) {
            case 'y':
            case 'Y':
                return true;
            case 'n':
            case 'N':
                return false;
            default:
                System.out.println("아닌걸로 간주할게요.");
                return false;
        }
    }
    public static void main(String[] args) {
        boolean answer = accept();
        System.out.println(answer);
    }
}
```

Loop and Conditional

```
import java.util.Scanner;
public class Action {
    public static void main(String[] args) {
        int x = 0;
        int y = 0;
        Scanner reader = new Scanner(System.in);
        while (true) {
            String action = reader.nextLine();
            for (char ch: action.toCharArray()) {
                switch (ch) {
                    case 'u': // up
                    case 'n': // north
                        ++y;
                        break;
                    case 'r': // right
                    case 'e': // east
                        ++x;
                        break;
                    // something more
                    default:
                        System.out.println("그대로 멈춰라!");
                        break;
                }
            }
        }
```

Advice

- 모르는 것이 있어도 너무 걱정하지 않아도 좋다. 나중에 다시 다루게 된다.
- 좋은 프로그램을 작성하기 위해 언어의 모든 측면을 알아야 하는 것은 아니다.
- 언어의 세세한 특징이 아니라 프로그래밍 테크닉에 집중하라.
- 언어의 자세한 면은 자바 공식 문서를 참고한다.
  `http://docs.oracle.com/javase/tutorial/index.html`
- 의미 있는 단위의 계산을 하나의 메소드로 묶고 알맞은 메소드 이름을 정해준다.
- 하나의 메소드는 하나의 논리적인 계산을 수행해야 한다.
- 메소드의 길이는 짧게 유지한다.

Advice

- 오버로딩은 개념적으로 같은 일을 다른 타입에 대해 수행하는 메소드에 사용한다.
- 상수에는 이름을 붙여서 사용한다.
- 자주 사용하는 지역 이름은 짧게, 가끔 사용하는 전역 이름은 길게 짓는다.
- 비슷한 이름은 피한다.
- 초기화하지 않은 선언은 피한다.
- 하나의 스코프는 가능하면 작게 유지한다.
- 코드에서 명확한 내용을 주석으로 남기지 않는다.
- 주석에는 의도를 표현한다.
- 인덴트 등을 일관성 있게 유지한다.
- 너무 복잡한 표현식(expression)은 피한다.