기초 자바

514770-1
2017년 봄학기
3/8/2017
박경신

Java Version

- Java 1.0
  - 1996
  - 211개 클래스
  - applet
- Java 1.1-1.4
  - 1997-2004
  - 2700개 클래스
  - ME, SE, EE version
  - 내부클래스, AWT, 자바빈즈, Reflection, Collection, Swing, 핫자바 JVM, JavaSound, JNDL, assertion, regular expression, XML parser
- Java 1.5-1.6
  - 2004-2006
  - 3700개 클래스
  - Generic class, foreach, 가변 인수, 오토 박싱, 메타 데이터, 열거형, 정적 import, interface
- Java 1.7-1.8
  - 2011-2016
  - 4200개 클래스
  - Lambda expression, Parallel Array Sorting, Base64 Encoding & Decoding API, Date and Time API, Password-Based-Encryption(PBE)

자바의 특징

- **WORA(Write Once Run Anywhere)**

Write Once!

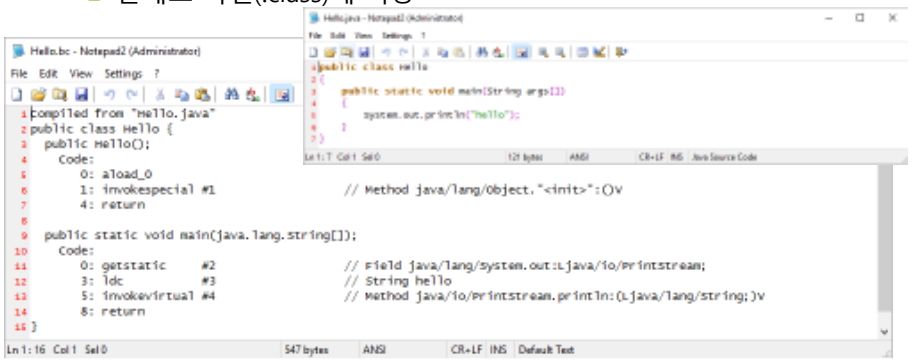
Run Everywhere!

자바 가상 기계

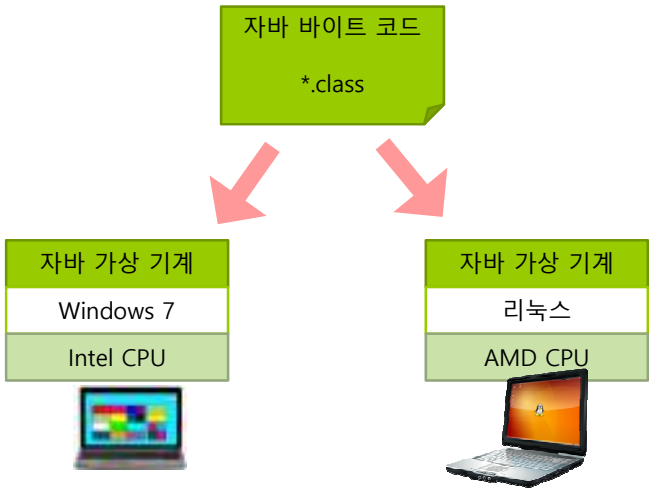
- 자바 가상 기계(JVM : Java Virtual Machine)
  - 각기 다른 플랫폼에 설치
  - 동일한 자바 실행 환경 제공
  - 자바 가상 기계 자체는 플랫폼에 종속적
    - 자바 가상 기계는 플랫폼마다 각각 작성됨
    - 예) 리눅스에서 작동하는 자바 가상 기계는 윈도우에서 작동하지 않음
  - 자바 가상 기계 개발 및 공급
    - 자바 개발사인 오라클 외 IBM, MS 등 다양한 회사에서 제작 공급
- 자바의 실행
  - 자바 가상 기계가 클래스 파일(.class)의 바이트 코드 실행

Bytecode

- 바이트 코드(Bytecode)
  - 자바 가상 기계에서 실행 가능한 바이너리 코드
    - 바이트 코드는 컴퓨터 CPU에 의해 직접 실행되지 않음
    - 자바 가상 기계가 작동 중인 플랫폼에서 실행
    - 자바 가상 기계가 인터프리터 방식으로 바이트 코드 해석
  - 클래스 파일(.class)에 저장



자바 가상 기계

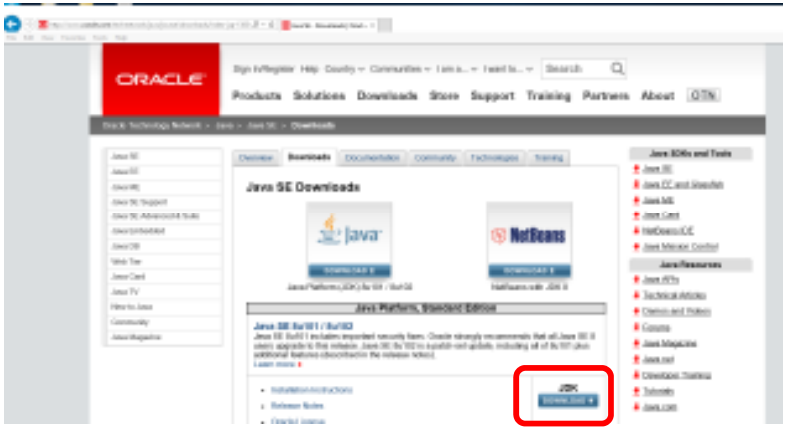


JDK와 JRE

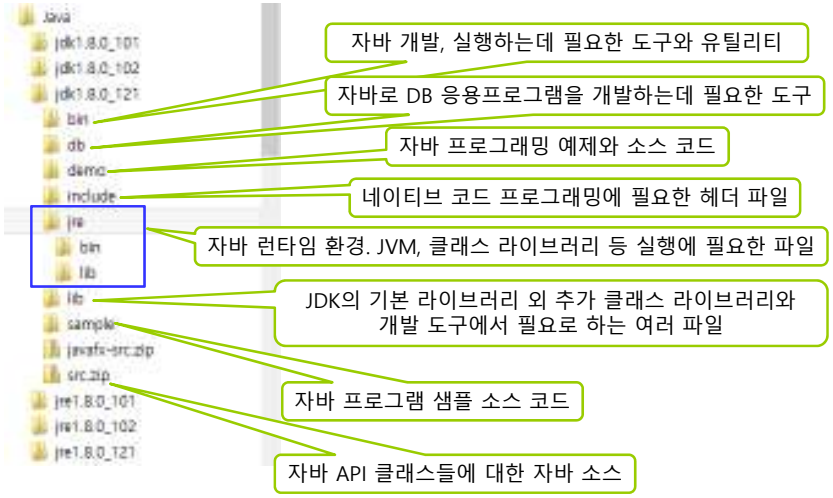
- JDK(Java Development Kit)
  - 자바 응용 개발 환경. 개발에 필요한 도구 포함
    - 컴파일러, JRE (Java Runtime Environment), 클래스 라이브러리, 샘플 등 포함
- JRE(Java Runtime Environment)
  - 자바 실행 환경. JVM 포함
  - 자바 실행 환경만 필요한 경우 JRE만 따로 다운 가능
- JDK와 JRE의 개발 및 배포
  - 오라클의 Technology Network의 자바 사이트에서 다운로드
    - http://www.oracle.com/technetwork/java/index.html
- JDK의 bin 디렉터리에 포함된 주요 개발 도구
  - javac - 자바 소스를 바이트 코드로 변환하는 컴파일러
  - java - JRE의 bin 디렉터리에도 있는 자바 응용프로그램 실행기
  - jar - 자바 아카이브 파일 (JAR)의 생성 및 관리하는 유틸리티
  - jdb - 자바 디버거
  - appletviewer - 웹 브라우저 없이 애플릿을 실행하는 유틸리티

JDK 설치

- Java SE Download
  - http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html

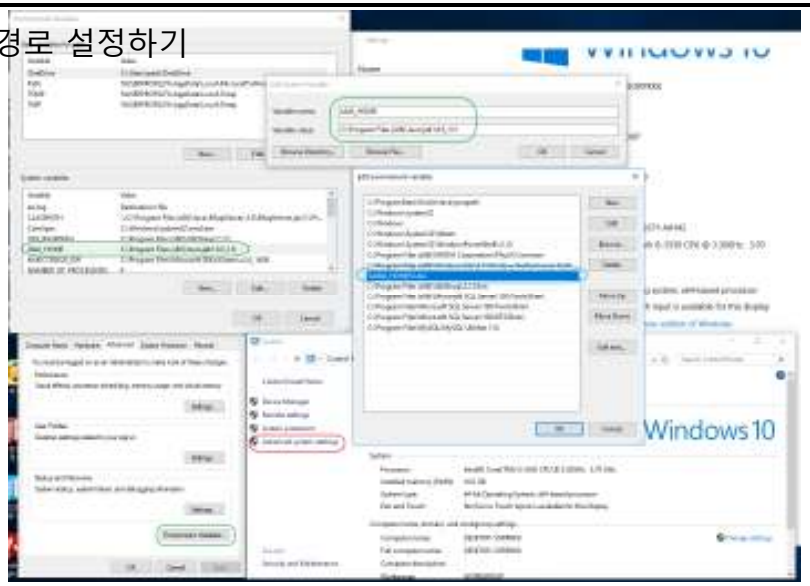


JDK 설치 후 디렉터리 구조



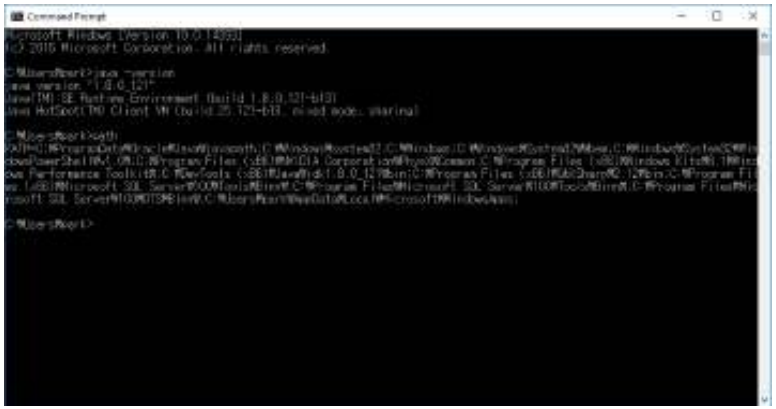
JDK 설치

- 경로 설정하기



JDK 설치

- 설치된 자바 버전 확인



자바 프로그램 개발 단계

- 소스 파일 .java
- 컴파일 후 바이트코드로 변환 .class
- 바이트코드를 메모리로 적재 및 검증
- JVM에서 바이트

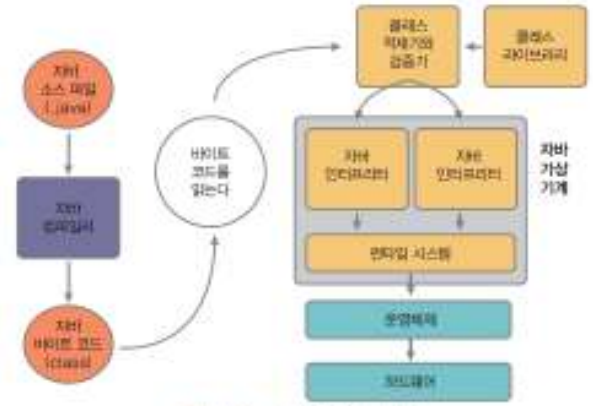
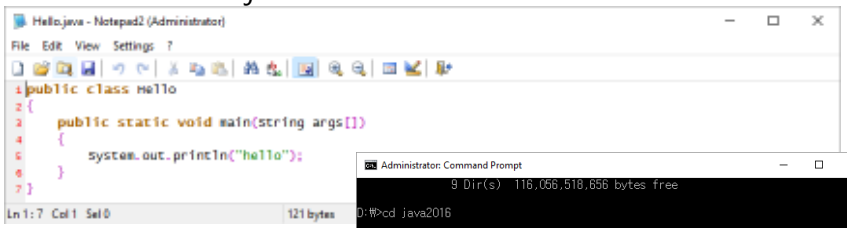


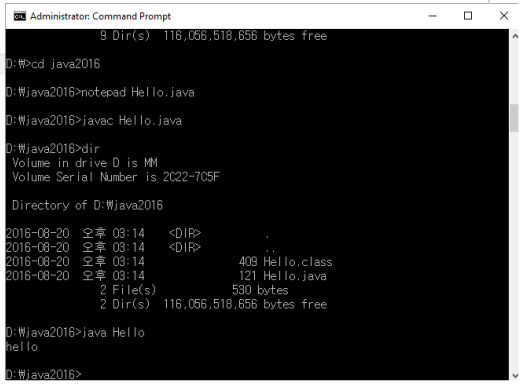
그림 1-12 • 자바 프로그램 개발 단계

자바 프로그램 개발 단계

- 소스코드 Hello.java



- 컴파일 javac Hello.java
- 실행 java Hello



자바 API

- 자바 API(Application Programming Interface)
  - JDK에 포함된 클래스 라이브러리
    - 주요한 기능들을 미리 구현한 코드(클래스 라이브러리)의 집합
  - 개발자는 API를 이용하여 쉽고 빠르게 자바 프로그램 개발
    - API에서 정의한 규격에 따라 클래스 사용
- 자바 패키지(Package)
  - 서로 관련된 클래스들을 분류하여 묶어 놓은 것
  - 계층구조로 되어 있음
    - 클래스의 이름에 패키지 이름도 포함
    - 다른 패키지에 동일한 이름의 클래스 존재 가능
  - 자바 API(클래스 라이브러리)는 JDK에 패키지 형태로 제공됨
    - 필요한 클래스가 속한 패키지만 import하여 사용
  - 개발자 자신의 패키지 생성 가능

자바 IDE 소개와 설치

- 자바 IDE (Integrated Development Environment)
  - 통합 개발 환경
  - 편집, 컴파일, 디버깅을 한번에 할 수 있는 통합된 개발 환경
- 이클립스(Eclipse)
  - 자바 응용 프로그램 개발을 위한 통합 개발 환경
  - IBM에 의해 개발된 오픈 소스 프로젝트
  - http://www.eclipse.org/downloads/ 에서 다운로드
- 넷빈즈(Netbeans)
  - https://netbeans.org/
- IntelliJ Idea
  - http://www.jetbrains.com/idea/download/

Eclipse 설치하기

- 이클립스 Neon (eclipse-java-neon-R-win32.zip) https://www.eclipse.org/downloads/ 다운로드
- eclipse.exe 실행

자바 프로그램 구조

```
/*
* 소스 파일 : Hello.java
*/
public class Hello {

  // main() 메소드에서 실행 시작
  public static void main(String[] args) {
     // "Hello World!" 문자열 화면 출력
     System.out.println("Hello World!");
  }
}
```

주석문

주석문

클래스

메소드

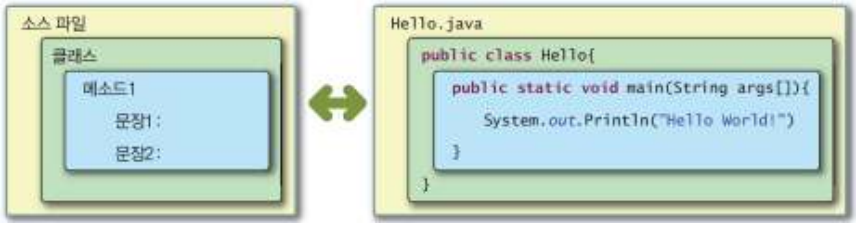
```
Hello World!
```

주석문 (Comments)

- 주석문
  - 프로그램에 대한 설명을 하기 위해 활용되는 코드의 일부로 컴파일 시에는 무시되고 사용되지 않음
- 세 가지 형태의 주석문
  - /* 설명 */
    - /*에서 */ 까지가 주석으로서 컴파일 시에 무시된다
    - 여러 줄에 걸쳐서 사용 가능
  - // 설명
    - //에서 줄의 끝까지 무시된다
  - /** 설명 */
    - /* 설명 */ 형태의 주석문이지만, 주로 선언문 앞에 사용되어 JDK에 포함된 Javadoc 프로그램을 이용해서 HTML문서를 만드는데 활용되는 주석문

자바 프로그램의 구조

- 클래스 (class): 객체(object)를 만드는 설계도 (템플릿)
- 자바 프로그램은 기본적으로 클래스로 구성됨



- **public** 키워드는 Hello 클래스가 다른 클래스에서도 사용 가능함을 나타냄
- 하나의 클래스 안에는 여러 개의 메소드가 포함될 수 있음
- 하나의 메소드 안에는 여러 개의 문장이 포함될 수 있음

Code Block

- 여러 명령문을 논리적으로 결합해야 할 때 중괄호 ({ })를 사용하여 명령문 그룹을 만들어 표현 - 이러한 명령문 그룹을 코드 블록(code block)이라고 함
- 코드 블록 안에는 변수를 선언할 수 있고, 다른 코드 블록을 포함할 수도 있음

```
public class Example {
    public static void main(String[] args) {
        int outer;
        {
            int inner;
            outer = 1;
            inner = 2;
        }
        outer = 5;
        //inner = 10;          // 오류
    }
}
```

내부 코드블록

main() 메소드 코드블록

Example 클래스 코드블록

Identifier

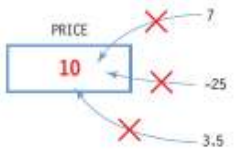
- 식별자란?
  - 클래스, 변수, 상수, 메소드 등에 붙이는 이름
- 식별자의 원칙
  - '@', '#', '!'와 같은 특수 문자, 공백 또는 탭은 식별자로 사용할 수 없으나 '_', '$'는 사용 가능
  - 유니코드 문자 사용 가능. 한글 사용 가능
  - 자바 언어의 키워드는 식별자로 사용불가
  - 식별자의 첫 번째 문자로 숫자는 사용불가
  - ' ' 또는 '$'를 식별자 첫 번째 문자로 사용할 수 있으나 일반적으로 잘 사용하지 않는다.
  - 불린 리터럴 (true, false)과 널 리터럴(null)은 식별자로 사용불가
  - 길이 제한 없음
- 대소문자 구별
  - Test와 test는 별개의 식별자

Keywords

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Constant

- 상수 (Constant)
  - **final** 키워드 사용
  - 변하지 않는 문자나 숫자 값. 값 변경 불가
  - 선언 시 초기값 지정



- 상수 선언 사례

```
final double PI = 3.141592;
final int LENGTH = 20;
```

Variable

- 변수(Variable)
  - 프로그램 실행 중에 값을 임시 저장하기 위한 공간
    - 변수 값은 프로그램 수행 중 변경될 수 있음
  - 데이터 타입에서 정한 크기의 메모리 할당
  - 반드시 변수 선언과 값을 초기화 후 사용
- 변수 선언과 초기화

```
int radius;
char c1, c2, c3; // 3 개의 변수를 한 번에 선언
double weight;
```

```
int radius = 10;
char c1 = 'a', c2 = 'b', c3 = 'c';
double weight = 75.56;
```

```
radius = 10 * 5;
c1 = 'r';
weight = weight + 5.0;
```

Data Type

- 자바의 자료형 (Data Type)
  - 기초형 (Primitive Type)
    - boolean *(1 Byte, true or false)*
    - char *(2 Bytes, Unicode)*
    - byte *(1 Byte, -128 ~ 127)*
    - short *(2 Bytes, -32768 ~ 32767)*
    - int *(4 Bytes, -2^{31}~2^{31}-1)*
    - long *(8 Bytes, -2^{63}~2^{63}-1)*
    - float *(4 Bytes, -3.4E38~3.4E38)*
    - double *(8 Bytes, -1.7E308~1.7E308)*
  - 참조형 (Reference Type)
    - class
    - interface
    - array

Data Type	Default Value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'₩u0000'
String (or any object)	null
boolean	false

String

- 문자열 (String)
  - 이중 인용부호("")로 묶어서 표현 – 예시 "Good", "Morning", "자바", "3.19", "26", "a"
  - 자바에서 문자열은 객체이므로 기본 타입이 아님
  - 자바에서 문자열**(String)**은 문자들의 모임이다. 예를 들어서 문자열 "Hello"는 H, e, l, l, o 등의 5개의 유니코드 문자로 구성됨
  - 문자열 리터럴은 String 객체로 생성됨
- **String 클래스**가 제공됨

```
String str1 = “Welcome”;
String str2 = “java”;
String str = str1 + str2;
System.out.println(str);
```

String

- + 연산자로 문자열(String) 연결
  - + 연산의 피연산자에 문자열이 있는 경우
  - + 연산에 객체가 포함되어 있는 경우
    - 객체.toString()을 호출하여 객체를 문자열로 변환한 후 문자열 연결
  - 기본 타입 값은 문자열로 변환된 후에 연결

```
System.out.print("abcd" + 1 + true + 3.13e-2 + 'E'+ "fgh" );
// abcd1true0.0313Efgh  출력
```

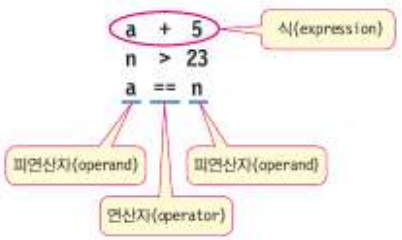
- String concat(String str)를 이용한 문자열 연결

```
"abcd".concat("efgh");
// “abcdefg” 리턴
```

  - 기존 String 객체에 연결되지 않고 새로운 스트링 객체 생성 리턴

Operator

- 연산 - 주어진 식을 계산하여 결과를 얻어내는 과정



연산자의 종류	연산자
증감	++ --
산술	+ - * / %
시프트	>> << >>>
비교	> < >= <= == !=
비트	& | ^ ~
논리	&& || ! ^
조건	? :
대입	= *= /= += -= &= ^= |= <<= >>= >>>=

Control Statement

- 제어문의 종류
  - 제어문이란 프로그램을 실행할 이러한 문장의 논리적인 흐름
  - 조건문 – 조건식의 값에 따라 수행한다. 예) if 문, switch 문
  - 반복문 – 조건이 만족하는 동안 특정 명령문을 반복적으로 수행한다. 예) while 문, do 문, for 문, foreach 문
  - 점프문 – 제어권을 이동시킬 때 점프문을 사용한다. 예) label 문, break 문, continue 문

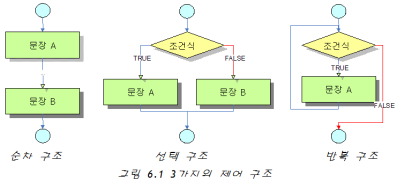
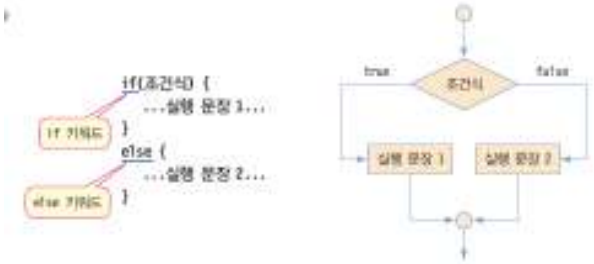


그림 6.1 3가지의 제어 구조

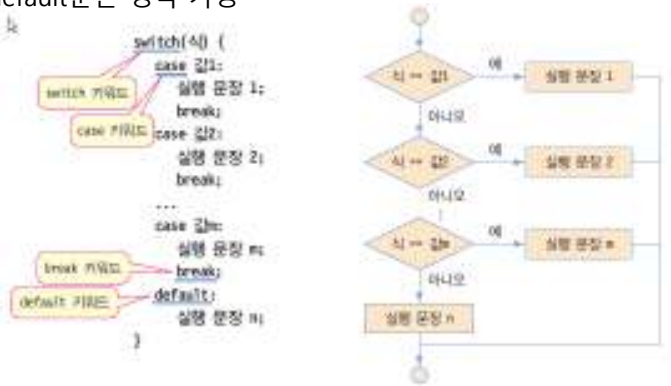
If-Else

- If-else 문
  - 조건식이 true면 실행문장1 실행 후 if-else문을 벗어남
  - false인 경우에 실행문장2 실행후, if-else문을 벗어남



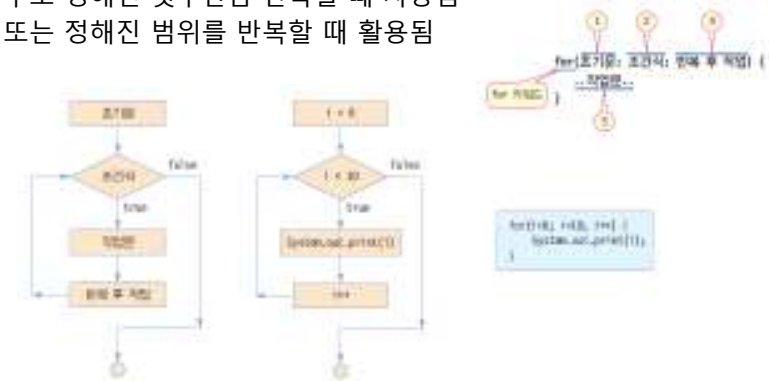
Switch

- switch문은 식과 case 문의 값과 비교
  - case의 비교 값과 일치하면 해당 case문의 실행문장 수행
    - break를 만나면 switch문을 벗어남
  - case의 비교 값과 일치하는 것이 없으면 default 문 실행
- default문은 생략 가능



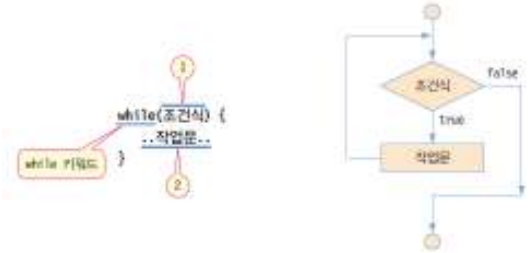
For-Loop

- For 문
  - 주로 정해진 횟수만큼 반복할 때 사용됨
  - 또는 정해진 범위를 반복할 때 활용됨



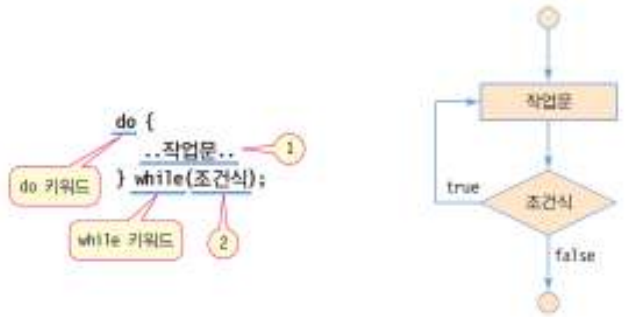
While-Loop

- While 문
  - 반복 조건이 true이면 반복, false 이면 반복 종료
  - 반복 조건이 없으면 컴파일 오류
  - 처음부터 반복 조건을 통과한 후 작업문 수행

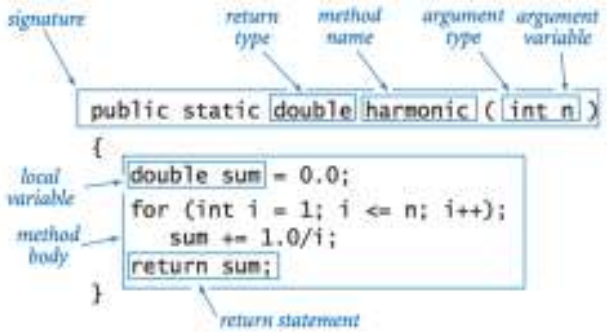


Do-While-Loop

- do-while 문
  - 무조건 최소 한번은 실행
  - 반복 조건이 true이면 반복, false이면 반복 종료
  - 반복 조건이 없으며 컴파일 오류



Method



Method

```
/*
* 소스 파일 : Hello2.java
*/
public class Hello2 {

   public static int sum(int n, int m) {
      return n + m;
   }

   // main() 메소드에서 실행 시작
   public static void main(String[] args) {
      int i = 20;
      int s;
      char a;

      s = sum(i, 10); // sum() 메소드 호출
      a = '?';
      System.out.println(a); // 문자 '?' 화면 출력
      System.out.println("Hello2"); // "Hello2" 문자열 화면 출력
      System.out.println(s); // 정수 s 값 화면 출력
   }
}
```

sum 메소드

main 메소드

```
?
Hello2
30
```

System.out.println

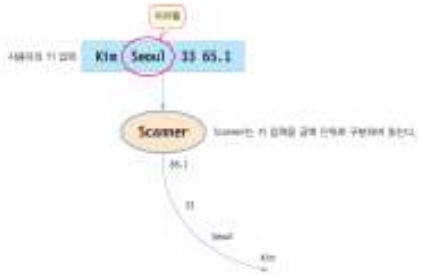
- 화면 출력
  - 표준 출력 스트림에 메시지 출력

```
System.out.println(a); // 문자 ? 화면 출력
System.out.println("Hello2"); // "Hello2" 문자열 화면 출력
System.out.println(s); // 정수 s 값 화면 출력
```

  - 표준 출력 스트림 System.out의 println() 메소드 호출
  - println()은 여러 종류 데이터 타입 출력 가능
  - println()은 출력 후 다음 행으로 커서 이동

Scanner 클래스

```
Scanner scanner = new Scanner(System.in);
String name = scanner.next();                  // "Kim"
String addr = scanner.next();                  // "Seoul"
int age = scanner.nextInt();                   // 33
double weight = scanner.nextDouble();          // 65.1
```



import

- 다른 패키지에 작성된 클래스 사용
  - import를 이용하지 않는 경우
    - 소스 내에서 패키지 이름과 클래스 이름의 전체 경로명을 써주어야 함

```
public class ImportExample {
    public static void main(String[] args) {
        java.util.Scanner scanner =
            new java.util.Scanner(System.in);
    }
}
```

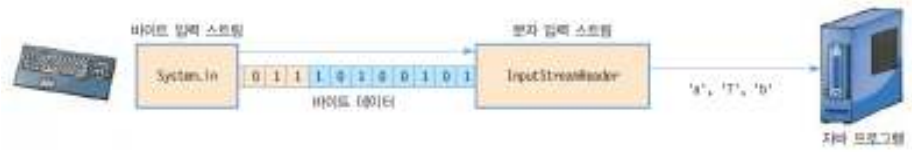
  - import 키워드 이용하는 경우
    - 소스의 시작 부분에 사용하려는 패키지 명시
      - 소스에는 클래스 명만 명시하면 됨
    - 특정 클래스의 경로명만 포함하는 경우
      - import java.util.Scanner;
    - 패키지 내의 모든 클래스를 포함시키는 경우
      - import java.util.*;
      - *는 현재 패키지 내의 클래스만을 의미하며 하위 패키지의 클래스까지 포함하지 않는다.

```
import java.util.Scanner;
public class ImportExample {
    public static void main(String[] args) {
        Scanner scanner =
            new Scanner(System.in);
    }
}
```

```
import java.util.*;
public class ImportExample {
    public static void main(String[] args) {
        Scanner scanner =
            new Scanner(System.in);
    }
}
```

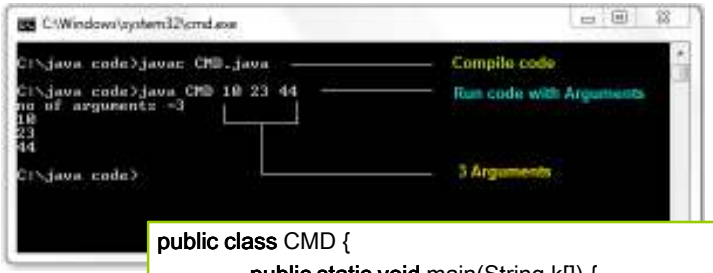
System.in

- 자바에서 키 입력: System.in
  - 자바의 표준 입력 스트림
  - java.io 패키지의 InputStream 클래스
    - System.in은 바이트 스트림으로서 키 값을 바이트로 리턴
    - 문자로 변환하려면 InputStreamReader 클래스를 이용



  - 입력 동안 문제가 발생하면 IOException 발생
    - try-catch를 이용한 예외 처리 필요

Command Line Arguments




```
public class CMD {
        public static void main(String k[]) {
                System.out.println("no. of arguments ="+k.length);
                for(int i=0;i< k.length;i++) {
                        System.out.println(k[i]);
                }
        }
}
```