

Java Programming

Introduction

소개

Computer Programming

: 간단한 이해

- ▶ 컴퓨터가 이해하는 언어는 1과 0으로 구성된 코드 (Machine Language: 기계어)
- ▶ 사람은 컴퓨터의 언어를 이해할 수 없고, 컴퓨터는 사람의 언어를 이해할 수 없다
- ▶ 사람의 언어와 기계 언어의 중간정도의 메타 언어가 필요
-> 프로그래밍 언어
- ▶ 저수준 언어 vs 고수준 언어
 - ▶ 성능상의 분류가 아님
 - ▶ 인간이 이해할 수 있는 언어 : 고수준 언어
 - ▶ 기계어에 가까운 언어 : 저수준 언어

High-level Language

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
TEMP = V(K)
V(K) = V(K+1)
V(K+1) = TEMP
```

Assembly Language

```
lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)
```

Machine Language

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

MIPS Assembler

C/Java Compiler

Fortran Compiler

해석 방법에 따른 언어 분류

: Compiler vs Interpreter

- ▶ 컴파일러: 프로그래밍 언어를 미리 기계어로 바꾸어 두고 실행
-> 번역에 가까움



Figure: Compiler

- ▶ 인터프리터: 프로그래밍의 소스코드를 읽고 그 즉시 기계에게 실행을 요청
-> 통역에 가까움

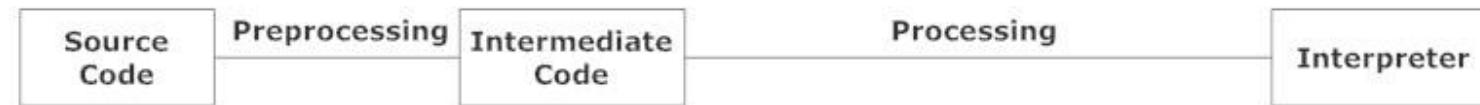


Figure: Interpreter

Java Language

: Birth

- ▶ 자바 언어의 탄생

- ▶ 1991년 미국 Sun Microsystems 사 제임스 고슬링(James Gosling)이 주도한 Oak 언어에서 출발
- ▶ 초기에는 가전제품에서 사용할 목적으로 고안
- ▶ 1995년, Sun World에서 공식 발표
- ▶ 1996년, JDK 1.0 발표
- ▶ 2009년, Oracle이 Sun Microsystems를 인수, 현재에 이름
- ▶ 2017년 현재 90이 가장 최신 버전, 8을 가장 일반적으로 사용



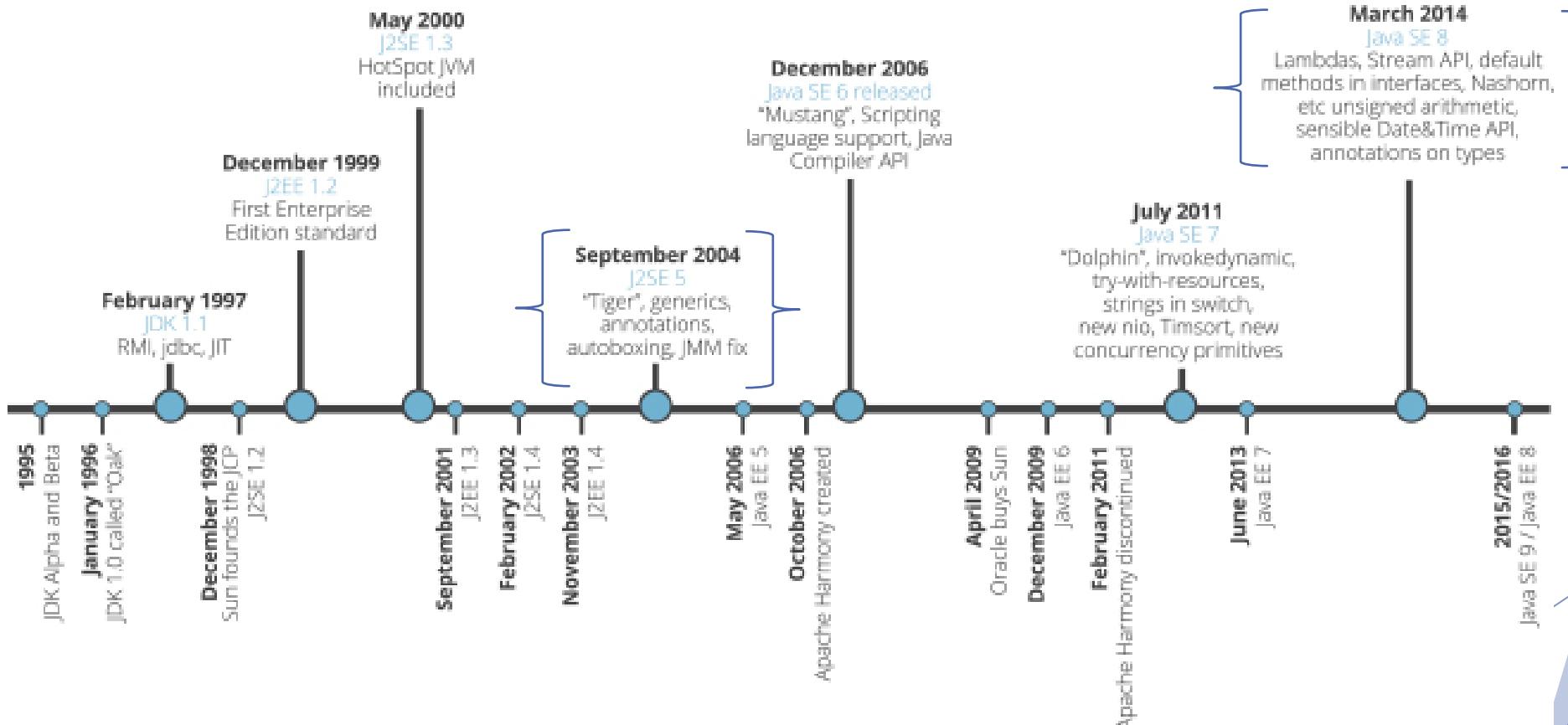
- ▶ Write Once, Run Everywhere

- ▶ 1996년(발표시점) ~ 1999년까지는 윈도 프로그래밍이 주류인 시장에서 열세
- ▶ 1990년대 후반, 월드와이드웹(WWW)이 활성화되면서 웹 애플리케이션 구축 언어로 급부상
- ▶ 다양한 장비와 환경에서 실행되는 애플리케이션을 개발하는 언어로 자리매김



Java Language

: Brief History



Java Language

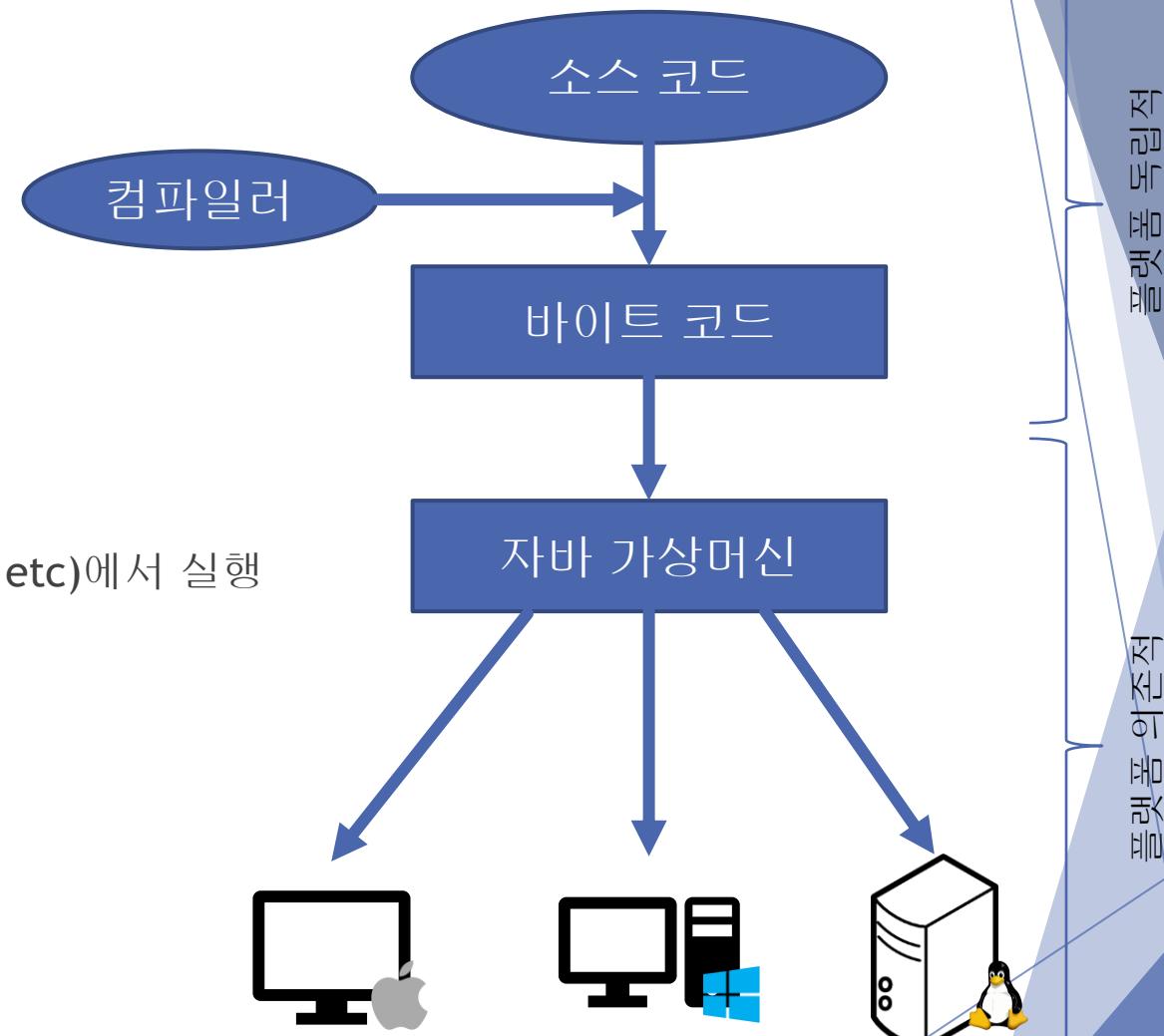
: Keywords

- ▶ Programming Language
- ▶ For Client-Server Programs (Both)
- ▶ Runs on all platforms
 - ▶ Via Java Virtual Machine (JVM)
- ▶ Syntax: derived from C/C++
- ▶ By Sun Microsystems (1991~1995)
- ▶ Object-Oriented
- ▶ Statically Typed

Java 특징 (1)

: 높은 이식성

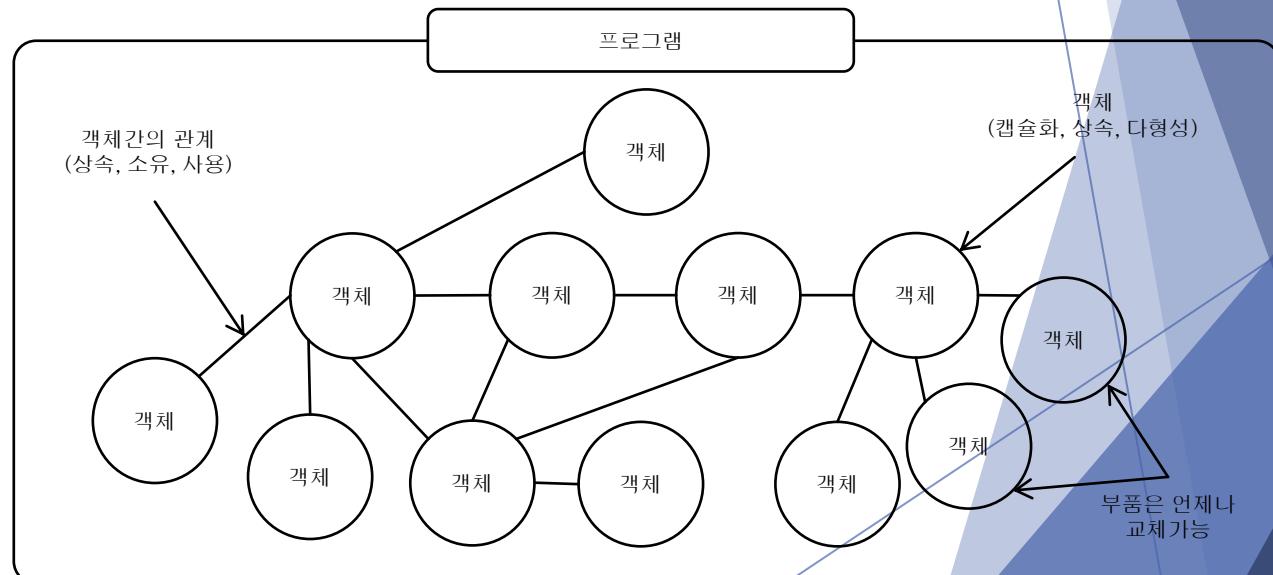
- ▶ Write Once, Run Everywhere
- ▶ JRE (Java Runtime Environment)
 - ▶ 소스 수정, 재컴파일 없이
 - ▶ 다양한 환경(Windows, Linux, Mac, etc)에서 실행



Java 특징 (2)

: 객체지향

- ▶ 100% 객체 지향 언어
- ▶ 객체지향 개발 : 서로 협력하는 다수의 객체들을 조합하는 개발 방식
 - ▶ 부품 - 완제품의 관계
 - ▶ 각 객체는 언제든 교체 가능
 - ▶ 현대 프로그래밍 패러다임 중 하나
- ▶ 캡슐화, 상속, 다형성 지원



Java 특징 (3)

: 단순한 문법과 개발 편의성

- ▶ C/C++ 언어의 구문과 C++ 객체지향에서 영향
 - ▶ 이해하기 어려운 포인터, 다중상속 등은 제외시킴
- ▶ 메모리 관리를 자바가 직접 관리
 - ▶ 개발자가 직접 메모리에 접근할 수 없도록 설계
 - ▶ Garbage Collector가 사용하지 않는 메모리를 제거
- ▶ Dynamic Language, 함수형 프로그래밍의 유용한 요소를 적극적으로 수용
 - ▶ 대용량 데이터의 병렬 처리, 이벤트 지향 프로그래밍 등에 유용 (Lambda 함수, NIO 등)
 - ▶ Java 8 이상부터 지원

Java 특징 (4)

: 그 외의 특징들

- ▶ 동적 로딩 (Dynamic Loading)
 - ▶ 필요한 시점에 객체를 동적 로딩
 - ▶ 변경이 필요한 부분만 수정하면 되므로 유지보수에 용이
- ▶ 풍부한 오픈 소스 라이브러리와 활성화된 개발자 생태계
- ▶ 범용 개발 언어
 - ▶ 다양한 운영체제, 다양한 디바이스에 대응하는 애플리케이션 개발 가능

Java 언어의 핵심 철학

: 5대 핵심 목표

- ▶ 객체 지향 방법론을 사용해야 한다
- ▶ 같은 프로그램(바이트 코드)이 여러 운영체제(마이크로프로세서)에서 실행될 수 있어야 한다
- ▶ 컴퓨터 네트워크 접근 기능이 기본으로 탑재되어 있어야 한다
- ▶ 원격 코드를 안전하게 실행할 수 있어야 한다
- ▶ 다른 객체 지향 언어들의 좋은 부분만 가지고 와서 사용하기 편해야 한다

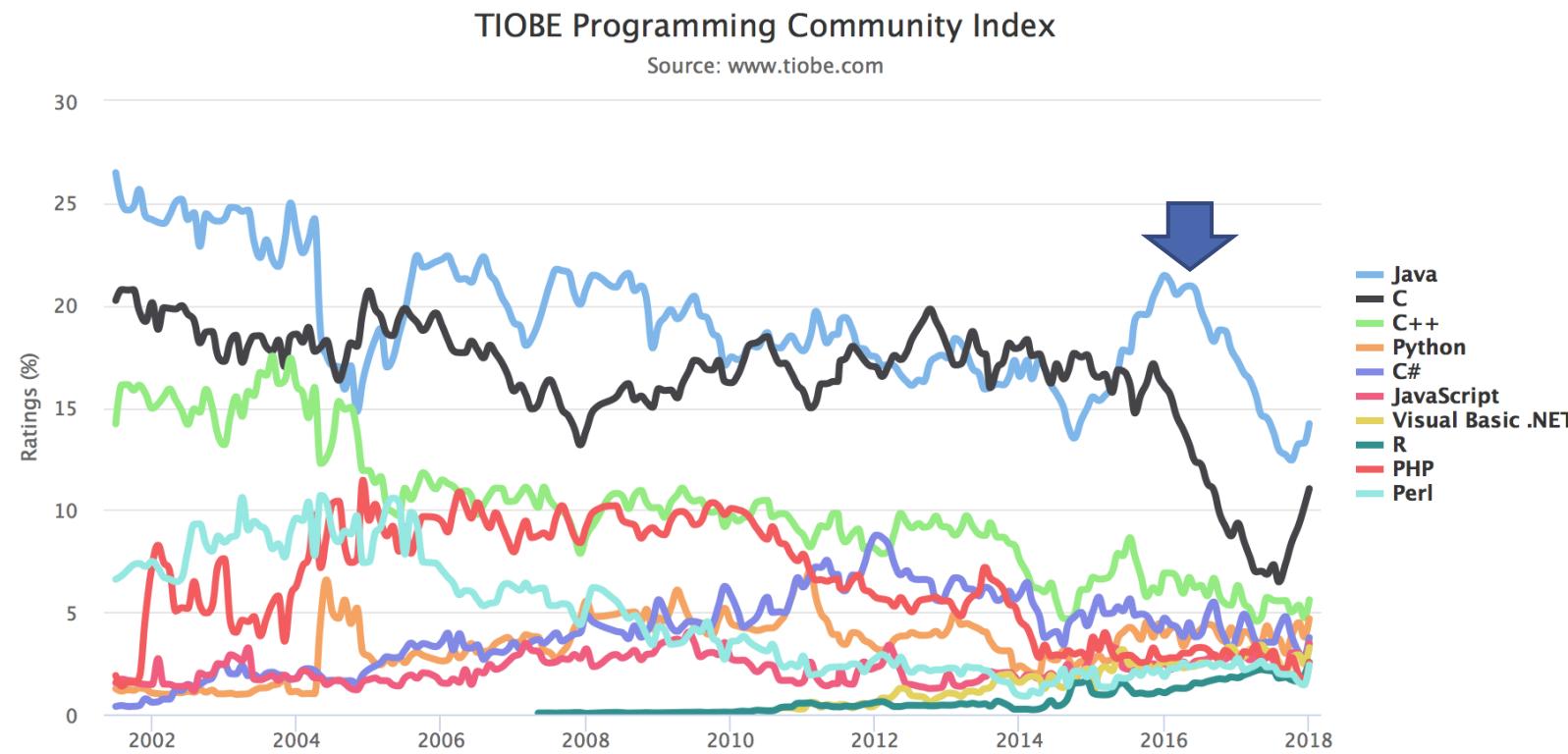
Java 언어의 현재와 미래

: TIOBE Index: 2018 December

Dec 2018	Dec 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.932%	+2.66%
2	2		C	14.282%	+4.12%
3	4	▲	Python	8.376%	+4.60%
4	3	▼	C++	7.562%	+2.84%
5	7	▲	Visual Basic .NET	7.127%	+4.66%
6	5	▼	C#	3.455%	+0.63%
7	6	▼	JavaScript	3.063%	+0.59%
8	9	▲	PHP	2.442%	+0.85%
9	-	▲	SQL	2.184%	+2.18%
10	12	▲	Objective-C	1.477%	-0.02%
11	16	▲	Delphi/Object Pascal	1.396%	+0.00%
12	13	▲	Assembly language	1.371%	-0.10%
13	10	▼	MATLAB	1.283%	-0.29%
14	11	▼	Swift	1.220%	-0.35%
15	17	▲	Go	1.189%	-0.20%
16	8	▼	R	1.111%	-0.80%
17	15	▼	Ruby	1.109%	-0.32%
18	14	▼	Perl	1.013%	-0.42%
19	20	▲	Visual Basic	0.979%	-0.37%
20	19	▼	PL/SQL	0.844%	-0.52%

Is Java Dead?

: 자바의 위기



Is Java Dead?

: 위기의 원인과 모던 자바의 역습

- ▶ Java의 위기

- ▶ 다양해진 플랫폼과 다양한 언어의 등장
- ▶ 경쟁 언어들에 비해 더딘 프로그래밍 패러다임 도입 및 언어적 개선
- ▶ JVM 언어의 약진
: Scala, Groovy, Kotlin 등 자바 언어보다 쉬우면서 최신 프로그래밍 패러다임을 도입하면서도 JVM의 성능을 그대로 활용할 수 있는 언어들의 도래

- ▶ 모던 Java의 역습

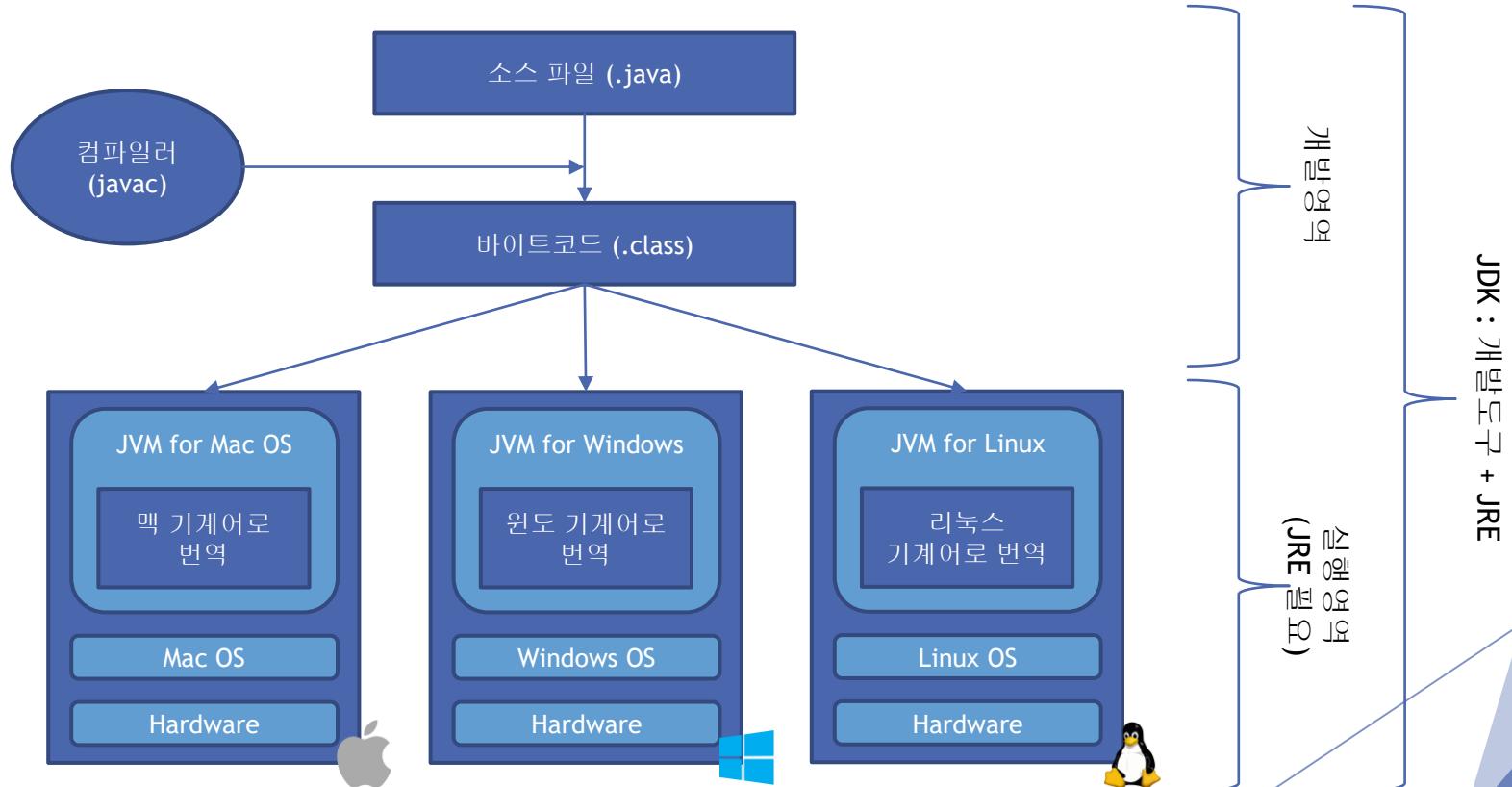
- ▶ Java 8, 9을 거치며, 언어적 개선과 최신 프로그래밍 패러다임을 적극적으로 수용
- ▶ 2018년 Java 11 LTS가 최신 버전

Java Virtual Machine

- ▶ Java는 직접 기계어로 컴파일하지 않고, 중간단계의 바이트 코드로 컴파일
- ▶ 바이트 코드를 읽고 수행할 수 있는 가상의 운영체제가 필요
 - ▶ JVM이 자바 프로그램과 실 운영체제를 중계
 - ▶ 한번의 작성으로 여러 운영체제와 상관 없는 프로그램 작성이 가능
- ▶ 이런 이유로 Java는 Compiler 언어와 Interpreter 언어의 혼합된 형태
 - ▶ 컴파일 단계에서 완전한 기계어로 번역되는 타 언어보다는 다소 느리다는 단점
 - ▶ JVM내 JIT(Just-In-Time) 컴파일러의 도입으로 속도의 격차는 많이 줄어듦

Java Programming

: 동작 방식



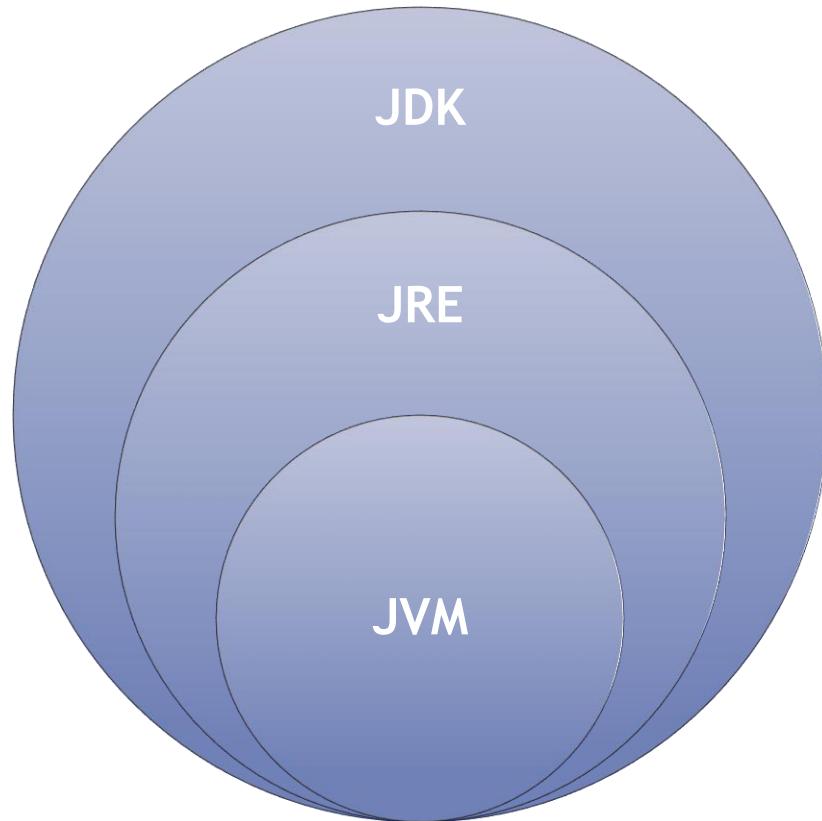
Java Programming

Development Environment

개발환경 구축

Java Development Environment

: JVM, JRE and JDK

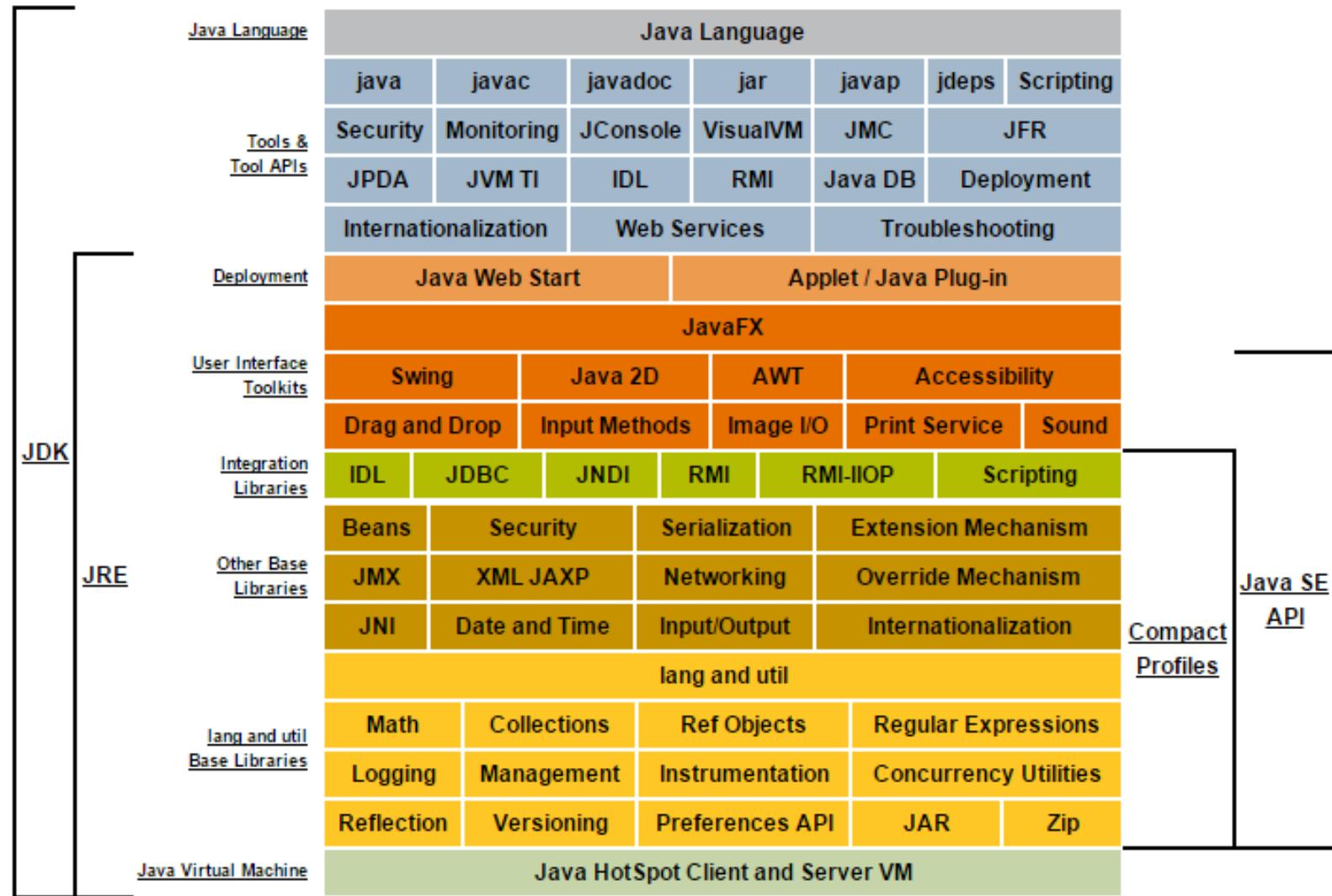


- ▶ **JVM (Java Virtual Machine)**
 - ▶ 자바 가상 머신
- ▶ **JRE (Java Runtime Environment)**
 - ▶ 자바 실행 환경
 - ▶ JVM + 표준 라이브러리
- ▶ **JDK (Java Development Kit)**
 - ▶ 자바 개발 키트
 - ▶ JRE + 라이브러리 API + 컴파일러 등
의 개발 도구

Java Development Environment

: Java Conceptual Diagram

Description of Java Conceptual Diagram

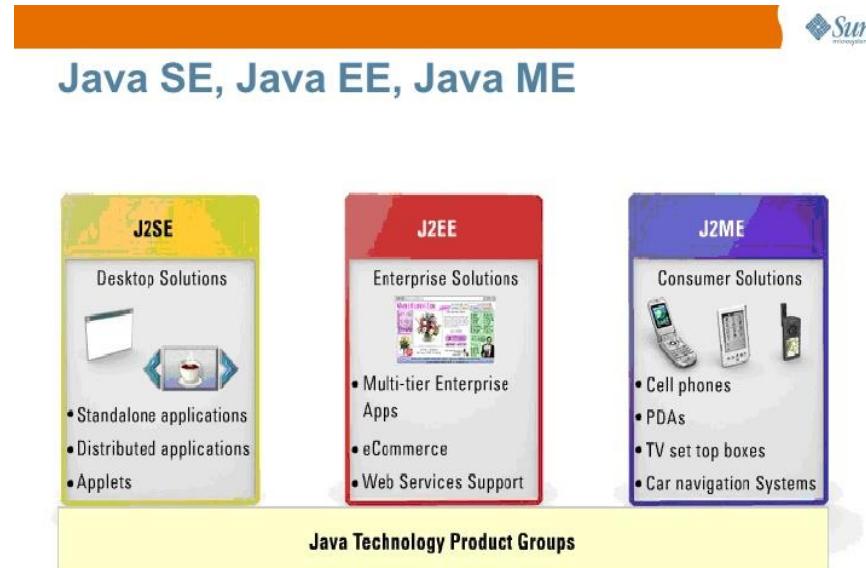


Java Development Environment

: Editions

JAVA의 분류

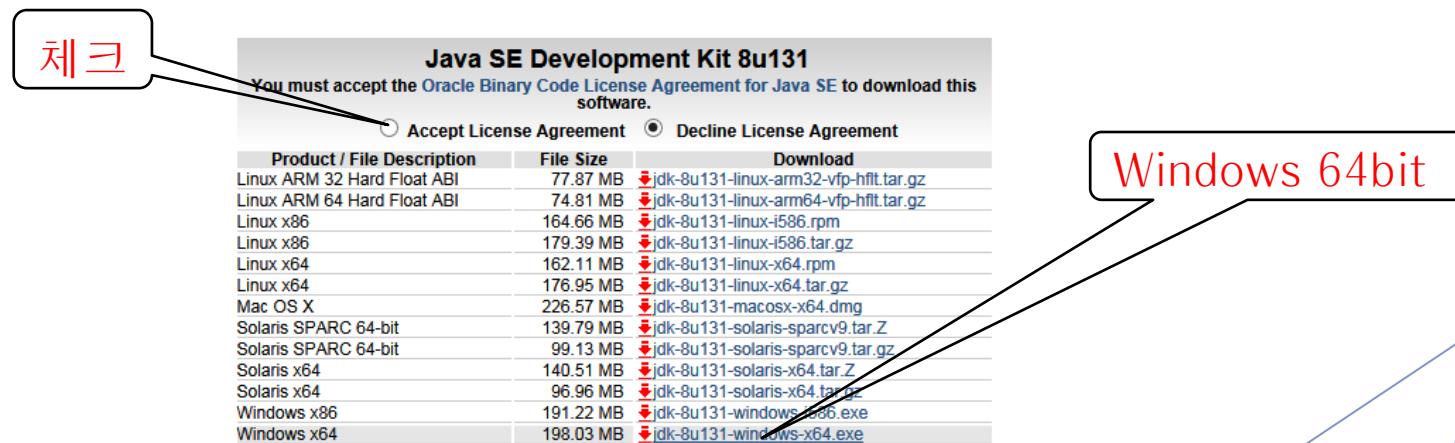
- ▶ SE (Standard Edition)
 - ▶ 일반 응용프로그램 개발용
- ▶ EE (Enterprise Edition)
 - ▶ 엔터프라이즈 응용프로그램 개발용
- ▶ ME (Micro Edition)
 - ▶ 소형 디바이스 개발용



Java Development Environment

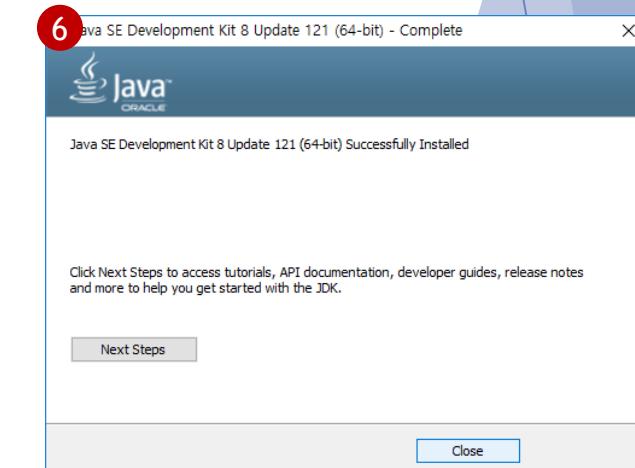
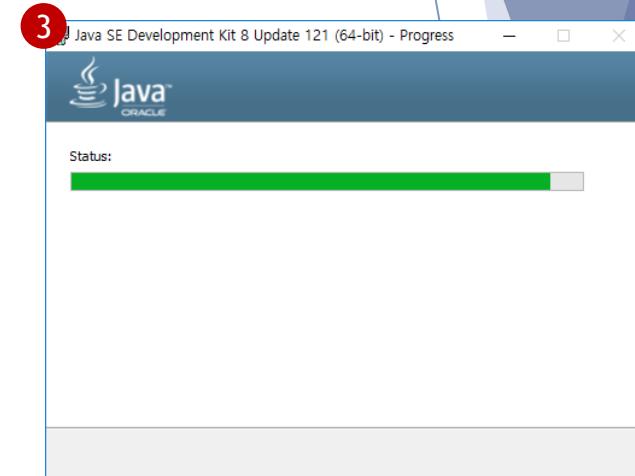
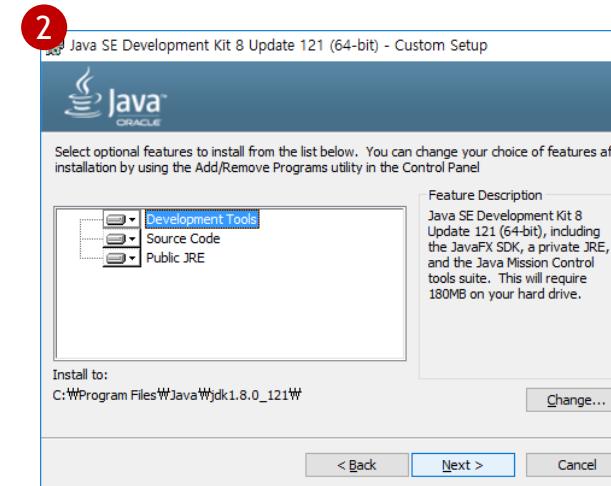
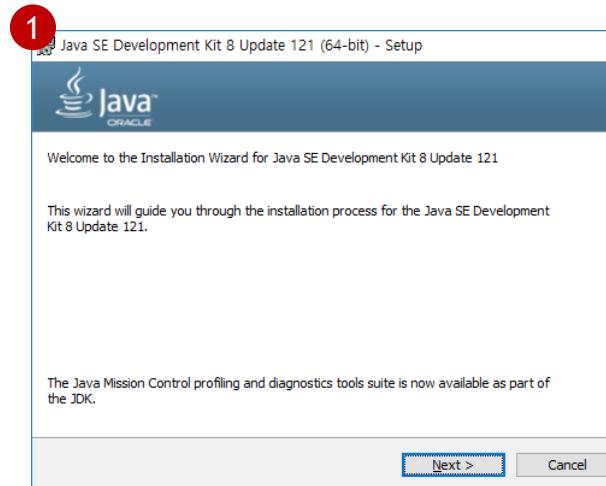
: Download JDK

- ▶ <http://java.oracle.com/>에서 다운로드
 - ▶ Downloads > Java SE > Downloads (탭)
 - ▶ License Agreement에 동의하고, 운영체제에 맞는 설치 파일을 다운로드



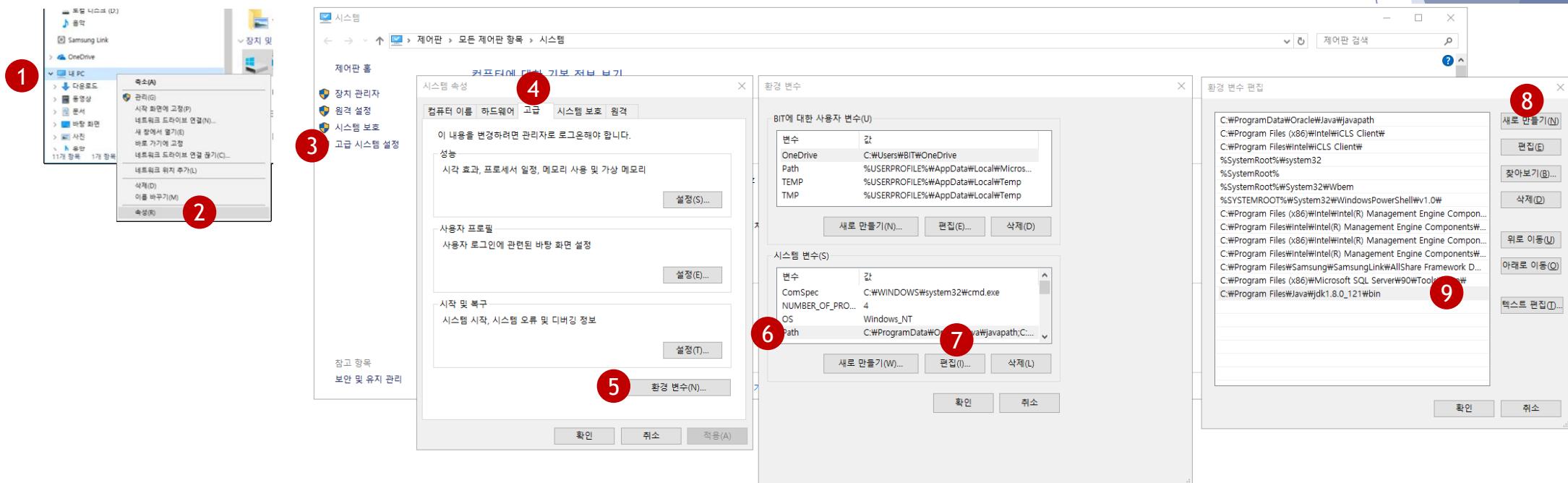
Java Development Environment

: Install JDK



Java Development Environment

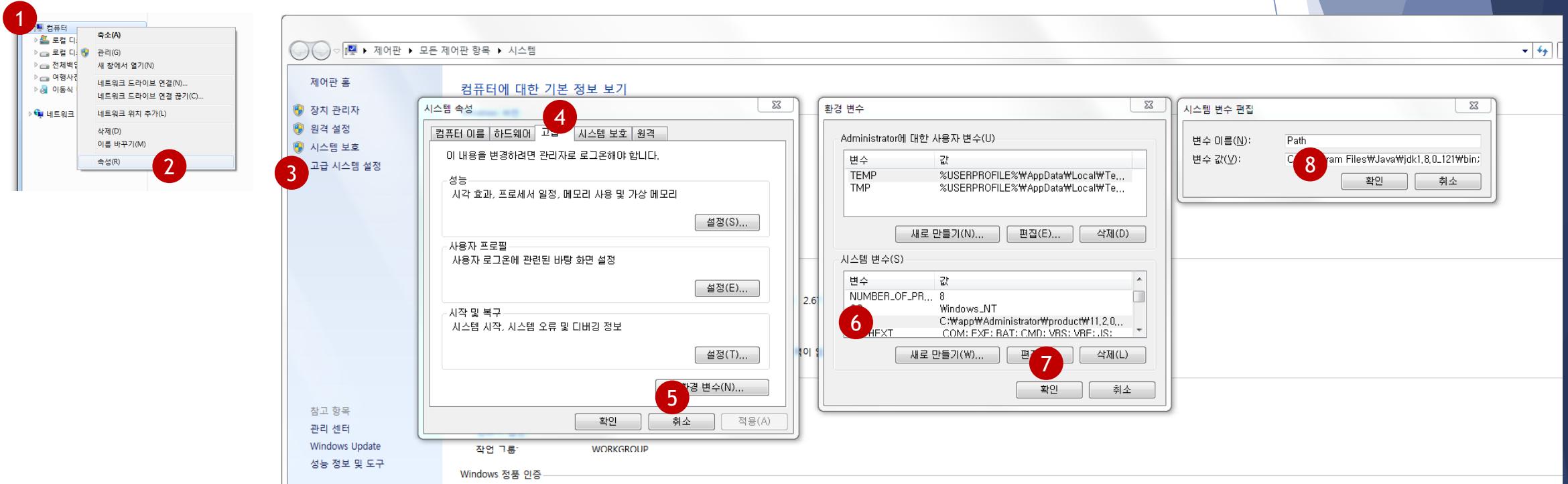
: 환경변수 설정 - Windows 8, 10



- 환경 변수에 JAVA_HOME 설정 : ex) C:\Program Files\Java\jdk1.8.0_131
- Path에 %JAVA_HOME%\bin 설정

Java Development Environment

: 환경변수 설정 - Windows 7



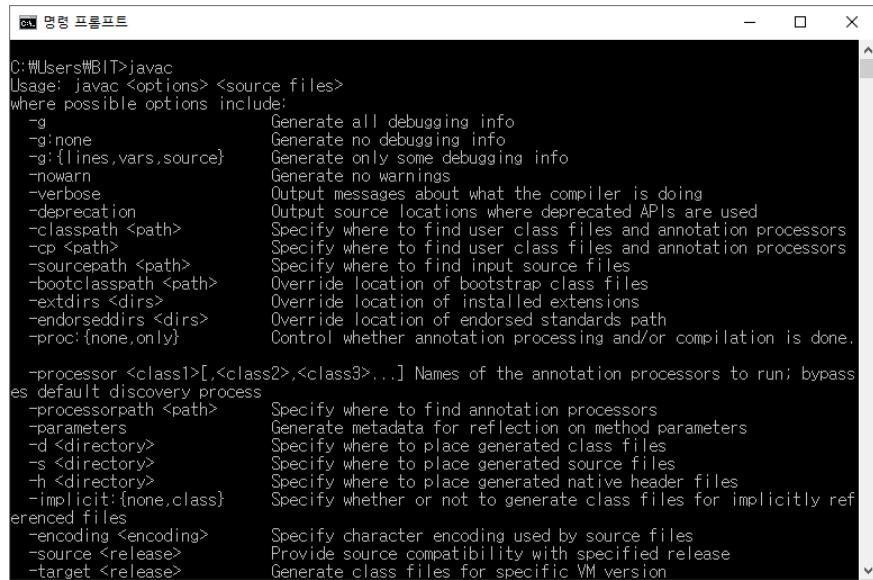
- 환경 변수에 JAVA_HOME 설정 : ex) C:\Program Files\Java\jdk1.8.0_131
- Path에 %JAVA_HOME%\bin 추가 :
ex) %JAVA_HOME%\bin; C:\Program Files (x86)\ATI Technologies\ATI.ACE\Core-Static

Java Development Environment

: 환경변수 설정 확인

▶ Win + R > cmd

javac 입력

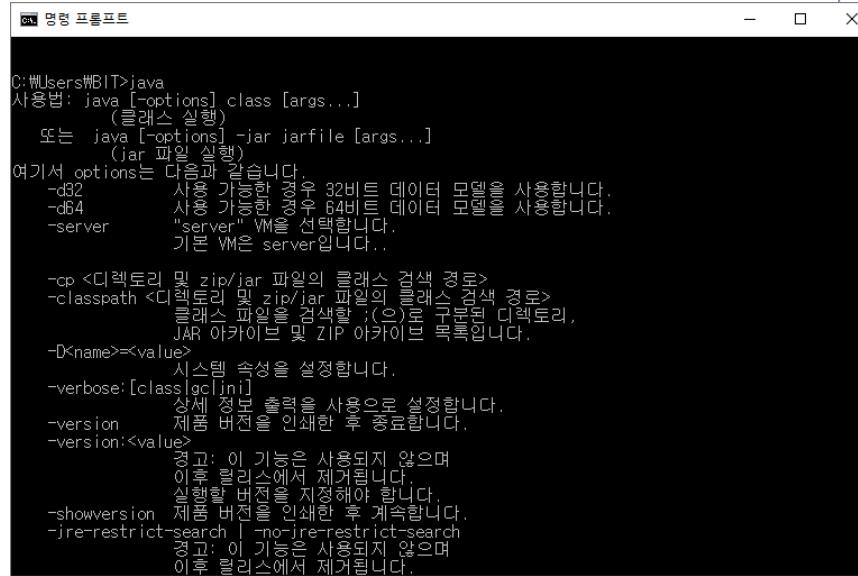


```
C:\#Userst\BIT>javac
Usage: javac <options> <source files>
where possible options include:
  -g                                     Generate all debugging info
  -g:none                                Generate no debugging info
  -g:{lines,vars,source}                   Generate only some debugging info
  -nowarn                                Generate no warnings
  -verbose                               Output messages about what the compiler is doing
  -deprecation                          Output source locations where deprecated APIs are used
  -classpath <path>                      Specify where to find user class files and annotation processors
  -cp <path>                             Specify where to find user class files and annotation 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 compilation is done.

  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; bypass
es default discovery process
  -processorpath <path>                  Specify where to find annotation processors
  -parameters                           Generate metadata for reflection on method parameters
  -d <directory>                        Specify where to place generated class files
  -s <directory>                        Specify where to place generated source files
  -h <directory>                        Specify where to place generated native header files
  -implied:{none,class}
  -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
```

정상적으로 실행되지 않으면 PATH 설정을 재확인

java 입력



```
C:\#Userst\BIT>java
사용법: java [<options>] class [args...]
          (클래스 실행)
  또는  java [-options] -jar jarfile [args...]
          (jar 파일 실행)
여기서 options는 다음과 같습니다.
  -d32    사용 가능한 경우 32비트 데이터 모델을 사용합니다.
  -d64    사용 가능한 경우 64비트 데이터 모델을 사용합니다.
  -server  "server" VM을 선택합니다.
           기본 VM은 server입니다..
  -cp <디렉토리 및 zip/jar 파일의 클래스 검색 경로>
  -classpath <디렉토리 및 zip/jar 파일의 클래스 검색 경로>
            클래스 파일을 검색할 ;(으)로 구분된 디렉토리,
            JAR 아카이브 및 ZIP 아카이브 복록입니다.
  -D<name>=<value>
            시스템 속성을 설정합니다.
  -verbose:[class|gc|ini]
            상세 정보 출력을 사용으로 설정합니다.
  -version  제품 버전을 인쇄한 후 종료합니다.
  -version:<value>
            경고: 이 기능은 사용되지 않으며
            이후 릴리스에서 제거됩니다.
            실행할 버전을 지정해야 합니다.
  -showversion 제품 버전을 인쇄한 후 계속합니다.
  -jre-restrict-search | -no-jre-restrict-search
            경고: 이 기능은 사용되지 않으며
            이후 릴리스에서 제거됩니다.
```

Java Development Environment

: JDK Document

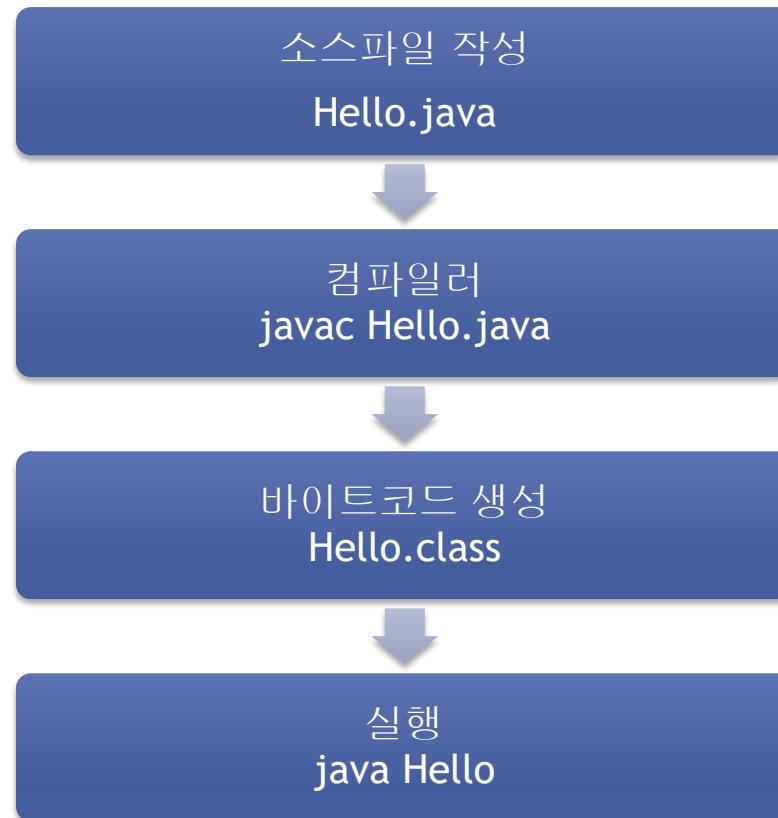
The screenshot shows a web browser window displaying the Java Platform Standard Edition 8 API Specification. The title bar reads "Overview (Java Platform Standard Ed. 8)" and the address bar shows "docs.oracle.com/javase/8/docs/api/index.html". The main content area is titled "Java™ Platform, Standard Edition 8 API Specification". It includes sections for "Profiles" (with items compact1, compact2, compact3) and "Packages". The "Packages" section is currently selected, showing three entries: "java.awt" (Description: Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context), "java.awt.color" (Description: Provides classes for color spaces), and "java.awt.image" (Description: Contains all of the classes for creating user interfaces and for painting graphics and images).

Package	Description
java.awt	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt.color	Provides classes for color spaces.
java.awt.image	Contains all of the classes for creating user interfaces and for painting graphics and images.

► <http://docs.oracle.com/javase/8/docs/api/index.html>

Java Development Environment

: 자바 개발 순서



Java Development Environment

: 첫 번째 자바 프로그램

- ▶ 소스 코드 작성
 - ▶ 편집기를 이용, 우측의 코드를 작성하고 “HelloWorld.java”라는 이름으로 저장
- ▶ 컴파일
 - ▶ 명령 프롬프트 실행
 - ▶ Hello.java 파일이 저장된 디렉토리로 이동하여 컴파일

```
javac HelloWorld.java
```

- ▶ 바이트코드 생성 확인
 - ▶ 컴파일한 디렉토리 내에서 class 파일 확인
- ▶ 실행하여 결과 확인

```
java HelloWorld
```

반드시 일치시켜야 함

HelloWorld.java

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

Java Development Environment

: 주석문, 실행문과 세미콜론(;)

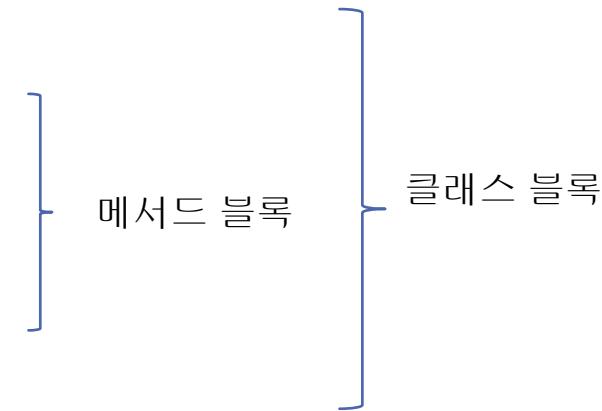
기호	설명
//	// 표시 지점부터 행 끝까지 주석으로 처리 (행 주석)
/* ~ */	/* 와 */ 사이 범위에 있는 내용을 주석으로 처리 (범위 주석)
/** ~ */	API 도큐먼트를 생성하는데 사용 (도큐먼트 주석)

- ▶ 실행문은 변수 선언, 값 저장, 메서드 호출에 해당하는 코드
- ▶ 실행문 맨 마지막에는 반드시 세미콜론(;)을 붙여야 함
 - ▶ 실행문이 끝났음을 자바에게 알려주는 표시

Java Development Environment

: 첫 번째 자바 프로그램 - 들여다보기

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



- ▶ 클래스 : 필드 또는 메서드를 포함하는 블록
- ▶ 메서드 : 어떤 일을 처리하는 실행문들을 모아 놓은 블록
- ▶ public static void main(String[] args) -> 익숙해질 때까지 연습

Java Development Environment

: Integrated Development Environment (IDE)



- ▶ 프로그램 개발에 관련된 모든 작업을 하나의 프로그램 안에서 처리하는 환경을 제공하는 소프트웨어
 - ▶ 코딩, 디버그, 컴파일, 배포 등
- ▶ 대표적인 Java IDE
 - ▶ Eclipse
 - ▶ Netbeans
 - ▶ IntelliJ IDEA

Java Development Environment

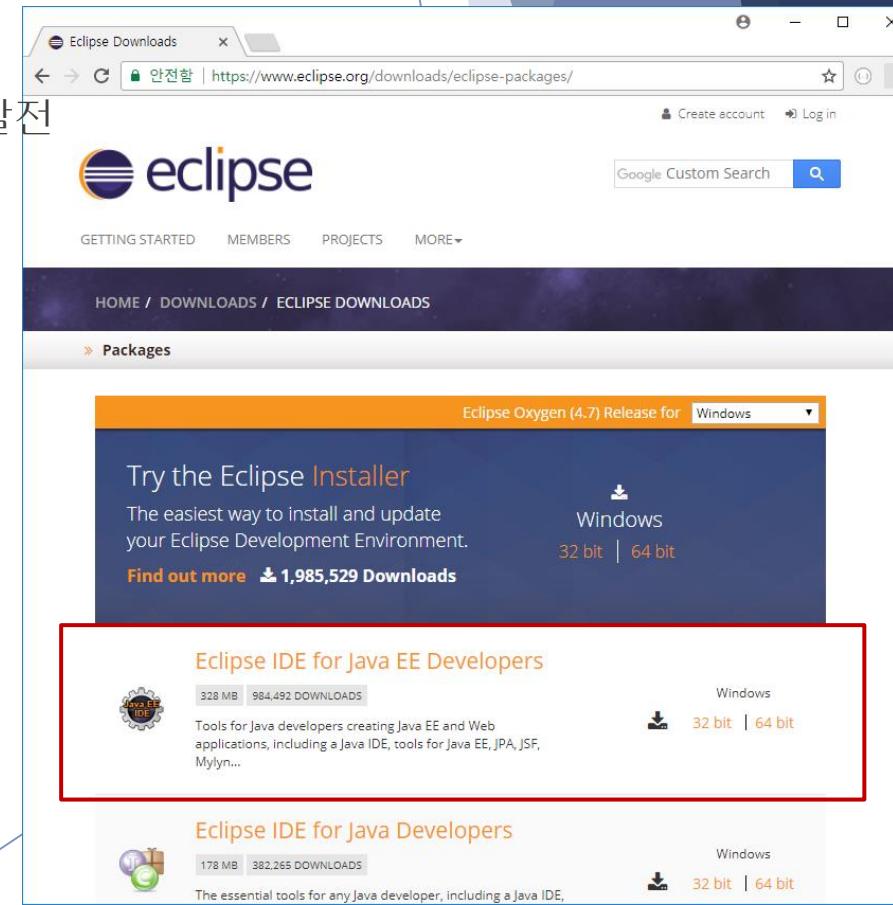
: Eclipse

▶ 특징

- ▶ 강력한 기능과 깔끔한 인터페이스
- ▶ IBM, 래쇼널 소프트웨어, 레드햇 등 여러 업체의 컨소시엄으로 개발, 발전
- ▶ 개발 목적에 맞는 다양한 버전과 변형이 있음
- ▶ 개발 편의를 위한 다양한 플러그인 제공
- ▶ 2017년 현재, 4.7 (Oxygen이 최신 버전)

▶ 설치

- ▶ <https://www.eclipse.org/>에서 다운로드
- ▶ 개발 PC의 OS에 맞는 버전을 다운로드
- ▶ 다운로드 받은 파일을 원하는 위치에 압축 해제

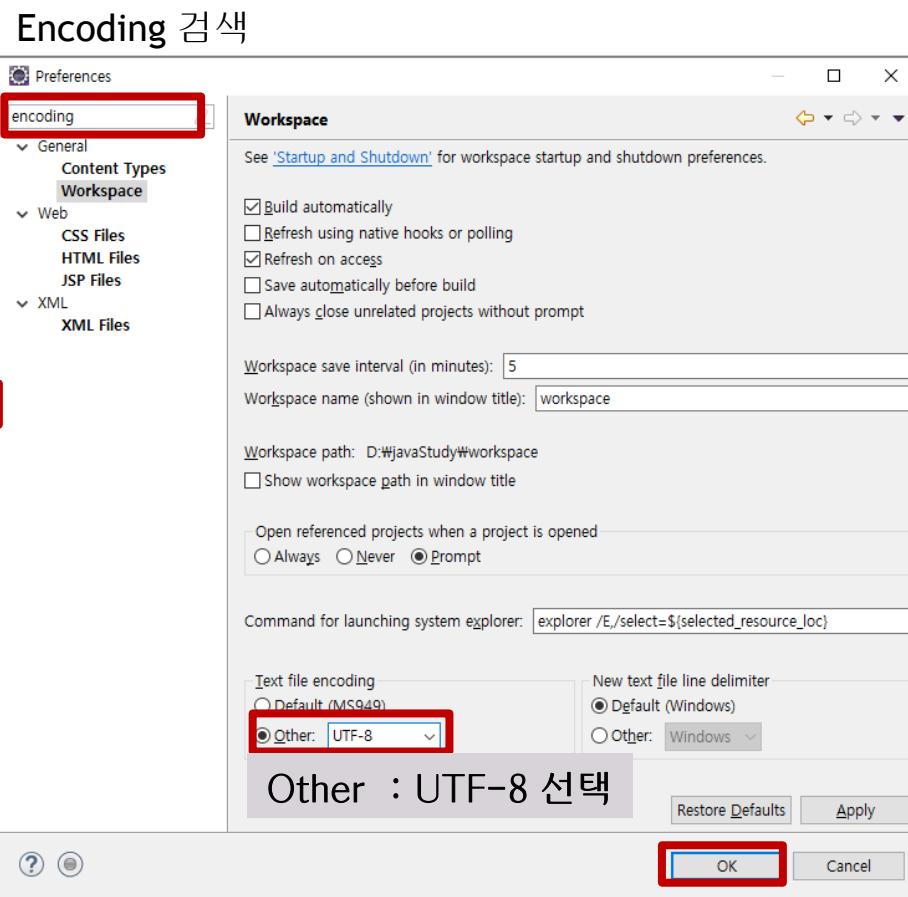
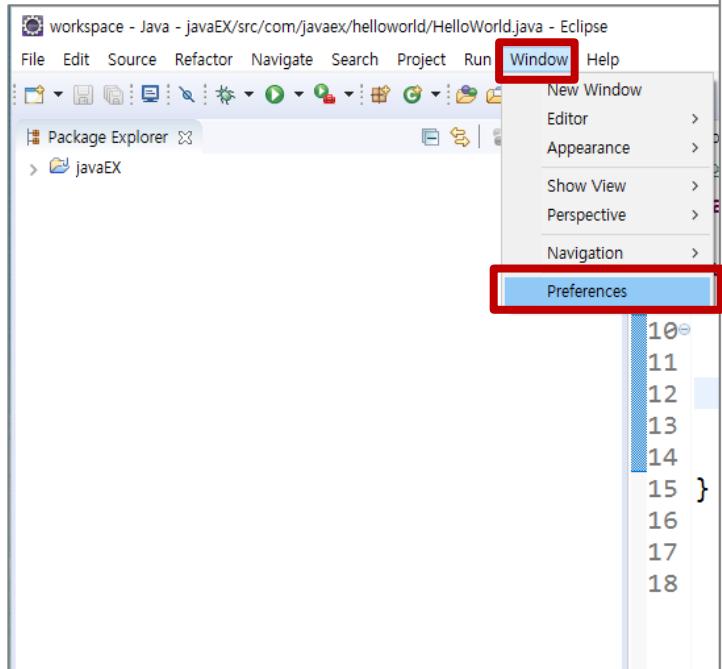


Java Development Environment

: Eclipse 설정

▶ Encoding 변경

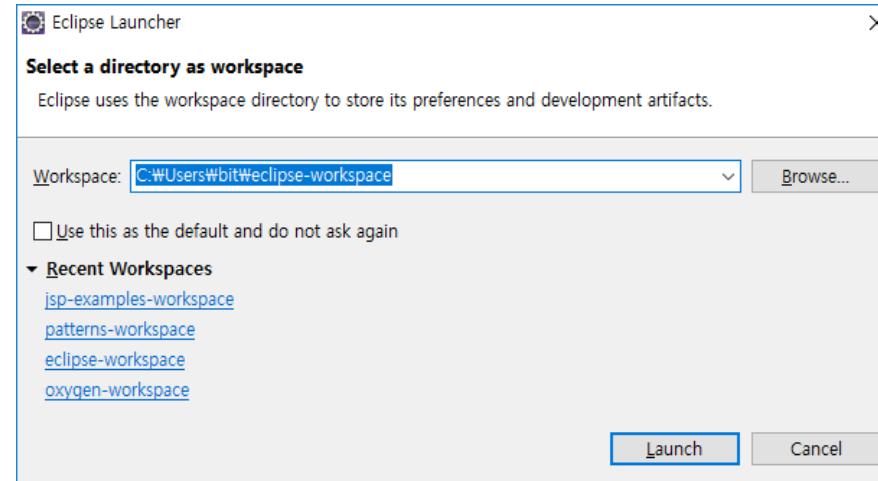
Windows > Preferences



Java Development Environment

: Eclipse 살펴보기

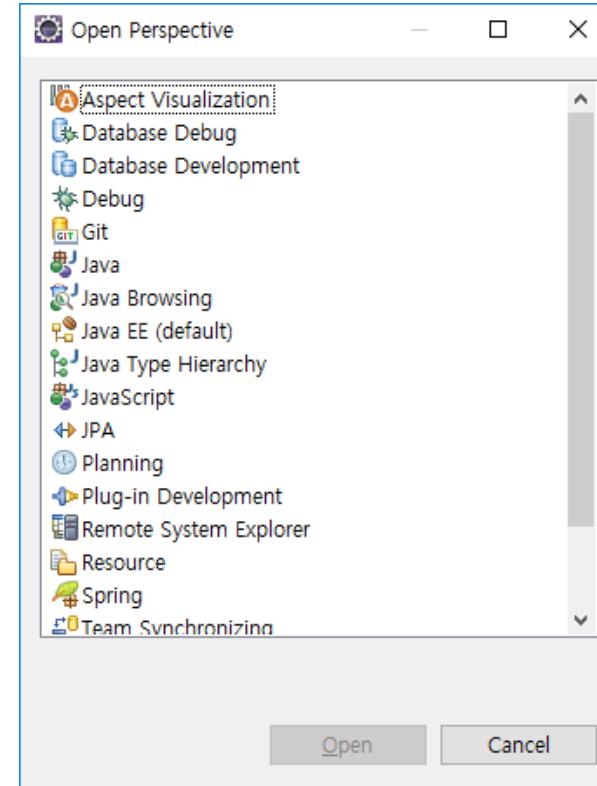
- ▶ 워크스페이스 (Workspace)
 - ▶ 이클립스에서 작성한 프로젝트가 저장되는 디렉토리
 - ▶ 이클립스를 사용하면서 변경되는 속성값들은 하위에 .metadata 디렉토리에서 관리
 - ▶ 원하면 다른 워크스페이스로 전환 할 수 있다.



Java Development Environment

: Eclipse 살펴보기

- ▶ 퍼스펙티브 (Perspective)
 - ▶ 프로젝트 개발시 유용하게 사용하는 뷰 (View)들을 묶어 놓은 것
 - ▶ 개발하고자 하는 응용프로그램에 따라 적합한 뷰 배치를 해 둔 것

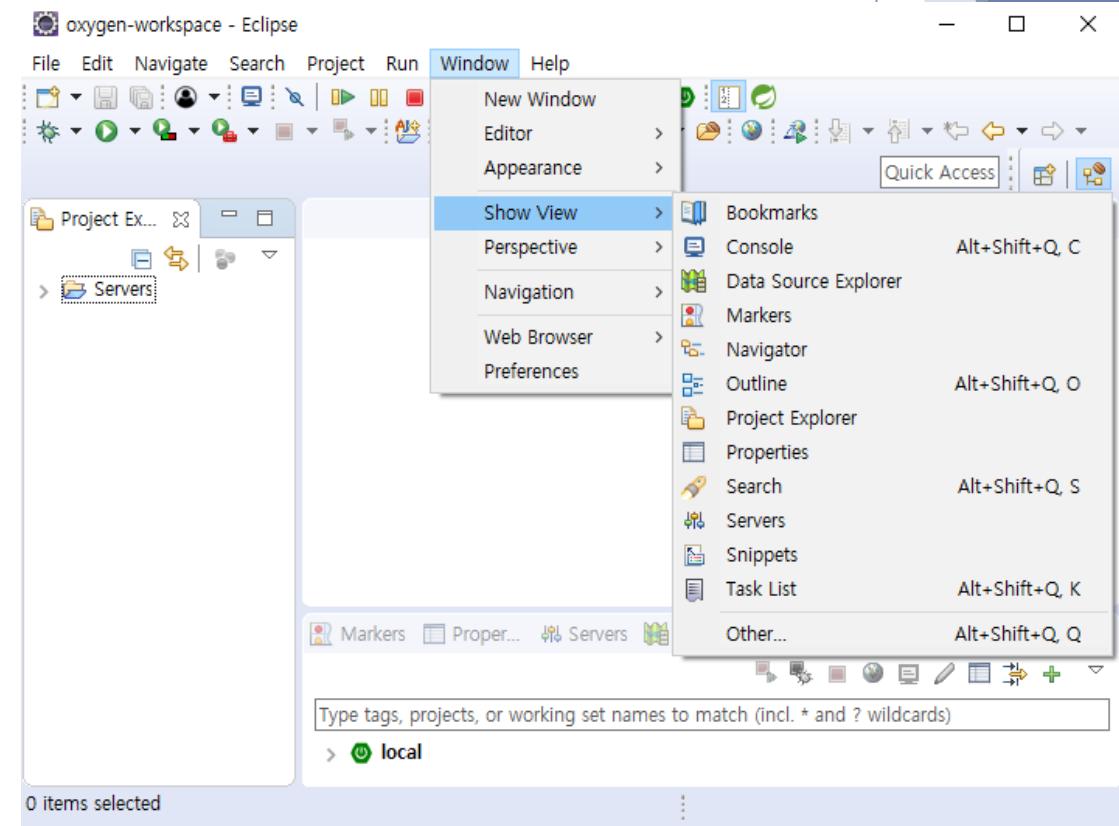


Java Development Environment

: Eclipse 살펴보기

▶ 뷰 (View)

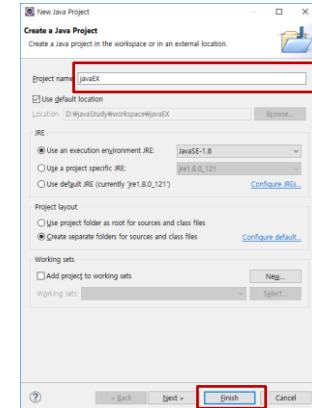
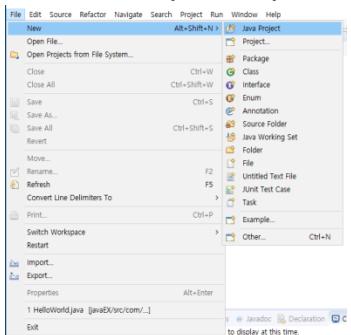
- ▶ 이클립스 내부에서 사용되는 용도별 창
- ▶ 원하는 뷰가 보이지 않는다면, Window > Show View 메뉴에서 찾는다



Java Development Environment

: 첫 번째 Eclipse 프로젝트

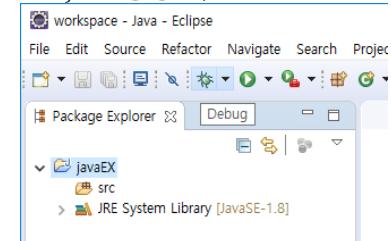
File > New > java Project 선택



finish 버튼 클릭

javaEx 입력

Project 생성 확인



프로젝트 만들기

패키지 만들기

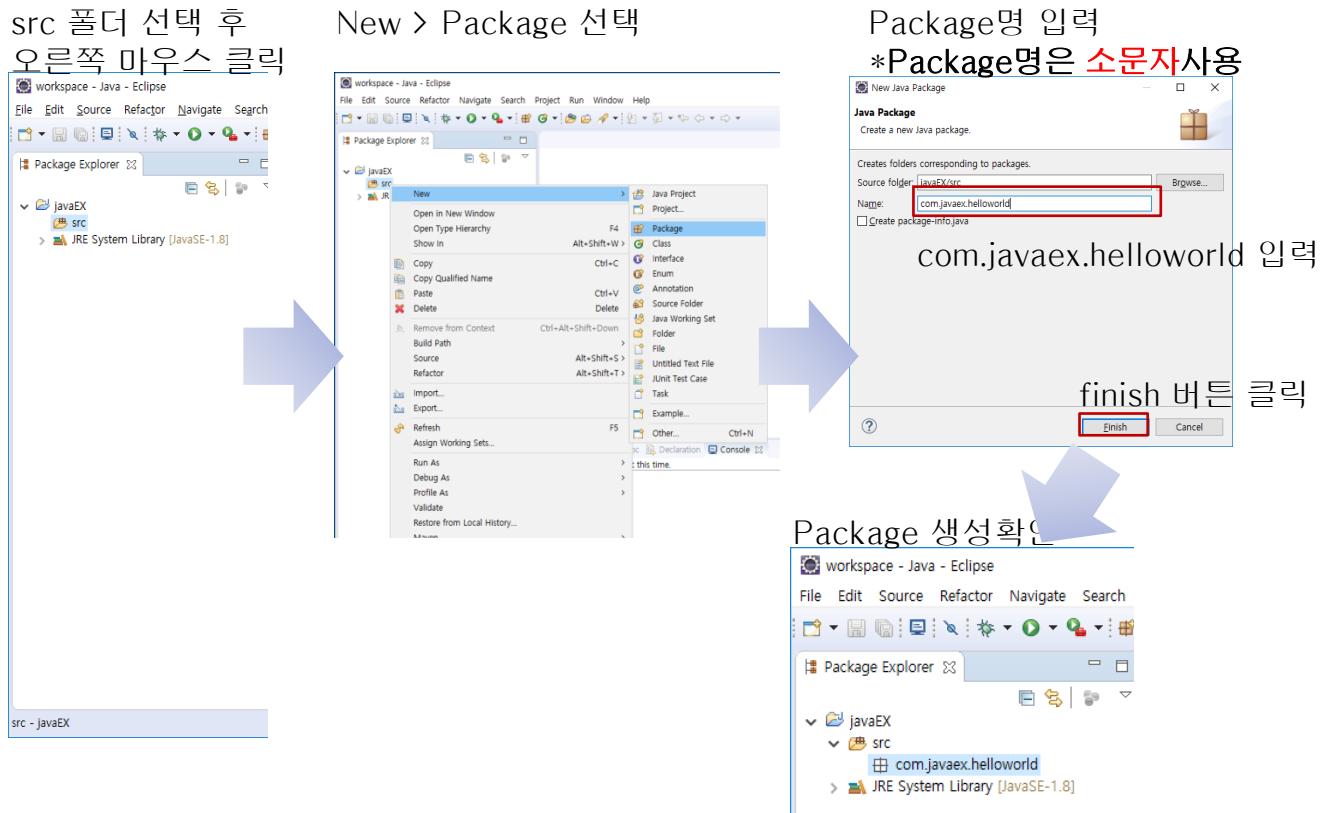
클래스 만들기

소스코드 작성

실행

Java Development Environment

: 첫 번째 Eclipse 프로젝트



프로젝트 만들기

패키지 만들기

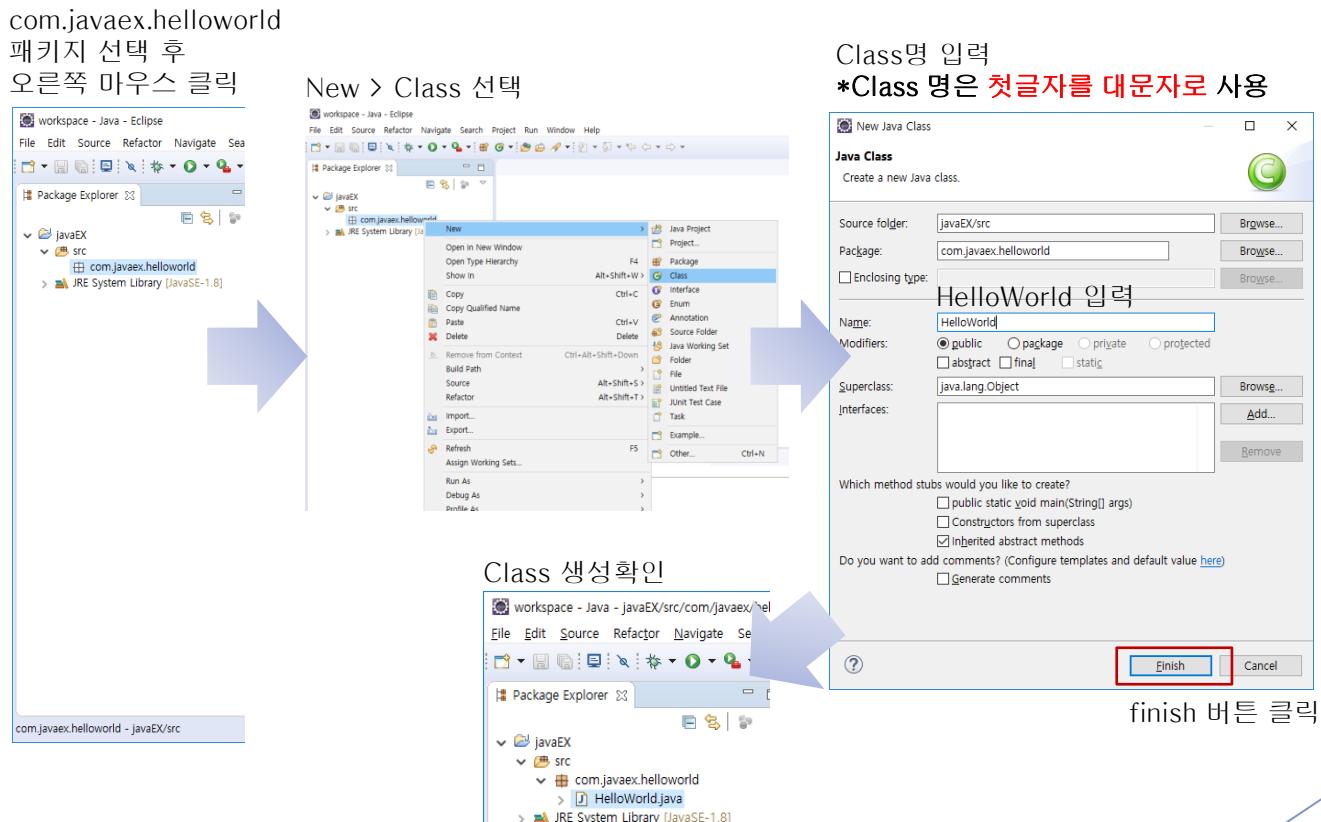
클래스 만들기

소스코드 작성

실행

Java Development Environment

: 첫 번째 Eclipse 프로젝트



프로젝트 만들기

패키지 만들기

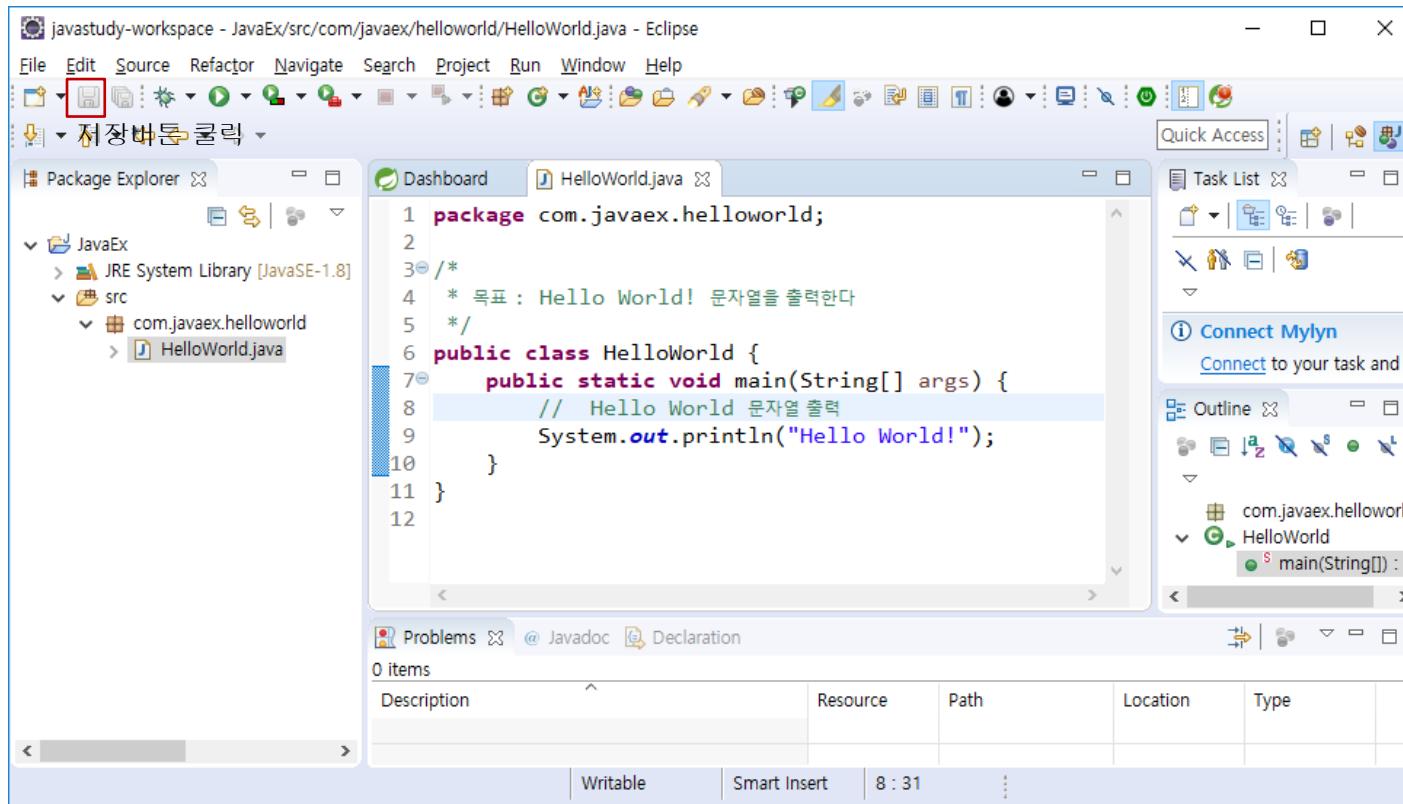
클래스 만들기

소스코드 작성

실행

Java Development Environment

: 첫 번째 Eclipse 프로젝트



프로젝트 만들기

패키지 만들기

클래스 만들기

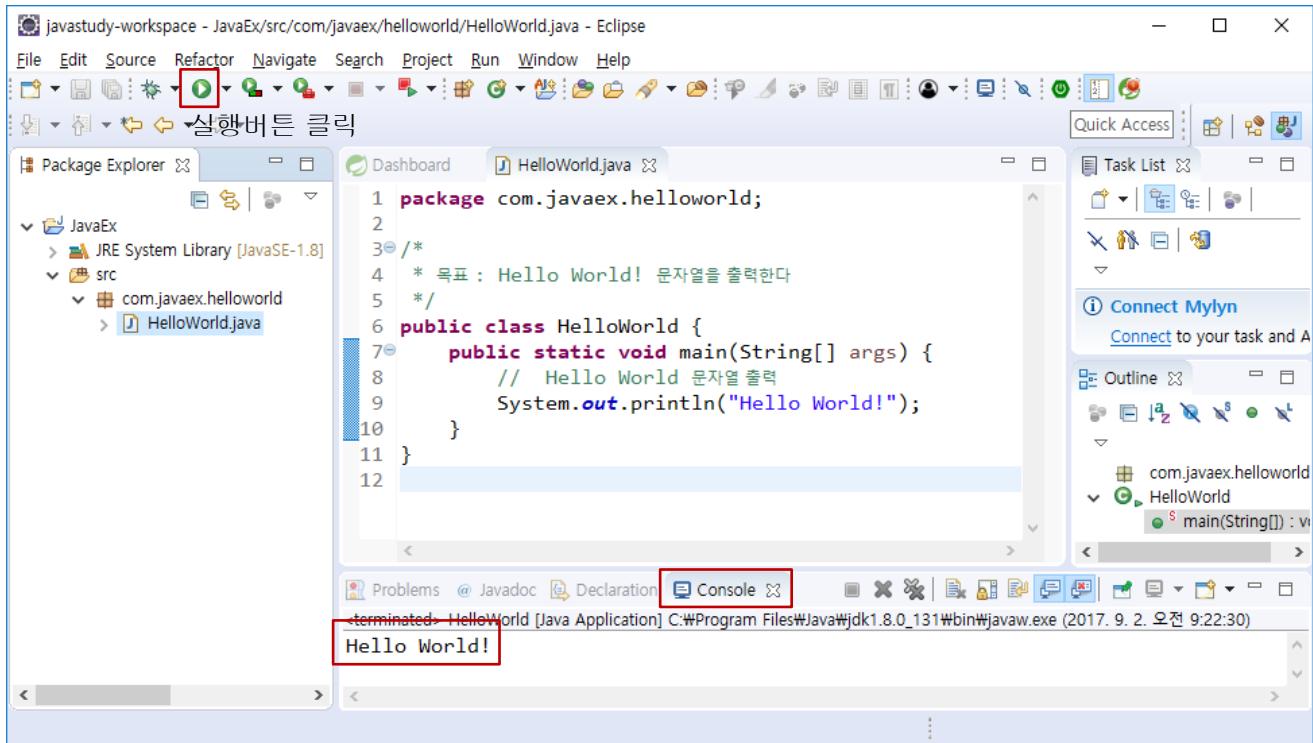
소스코드 작성

실행

Java Development Environment

: 첫 번째 Eclipse 프로젝트

실행버튼 클릭 후 console창 결과 확인



프로젝트 만들기

패키지 만들기

클래스 만들기

소스코드 작성

실행

Java Development Environment

: 코드 구조 살펴보기

```
① package com.javaex.helloworld;  
  
/* ②  
 * 목표 : Hello World! 문자열을 출력한다  
 */  
  
public class HelloWorld {  
    public static void main(String[] args) {  
        // Hello World 문자열 출력 ⑤  
        ⑥ System.out.println("Hello World!"); ⑦  
    }  
}
```

- (1) 패키지 경로, 패키지명
클래스를 체계적으로 관리하기 위한 묶음
소문자로 작성
- (2) 범위 주석
코드를 설명하기 위한 문장
프로그램 수행에는 아무 영향 없음
- (3) 클래스 블록
클래스명의 첫글자는 대문자로 작성
파일명과 클래스명은 일치
- (4) 메서드 블록
여러 실행문의 묶음
내부의 실행문을 절차적으로 실행
- (5) 행 주석
- (6) 실행문
- (7) 문장의 끝은 세미콜론(;)으로 표시

Java Programming

Language

언어

예약어와 식별자

Java Language

: 예약어와 식별자

▶ 예약어 (Keyword)

- ▶ 프로그래밍 언어에 미리 정의된 단어
- ▶ 식별자로 사용하지 않음

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

분류	예약어
기본 데이터 타입	boolean, byte, char, short, int, long, float, double
접근 지정자	private, protected, public
클래스 관련	class, abstract, interface, extends, implements, enum
객체 관련	new, instanceof, this, super, null
메서드 관련	void, return
제어문 관련	if, else, switch, case, default, for, do, while, break, continue
논리 리터럴	true, false
예외 처리 관련	try, catch, finally, throw, throws
기타	transient, volatile, package, import, synchronized, native, final, static, strictfp, assert

Java Language

: 예약어와 식별자

▶ 식별자 (Identifier)

- ▶ 프로그래머가 직접 만들어주는 이름
- ▶ 변수명, 클래스명, 메서드명 등
- ▶ 명명 규칙(Naming Convention)에 따라 지정

작성 규칙

문자, \$, _로 시작해야 함

숫자로 시작할 수 없음

대소문자가 구분됨

예약어는 사용할 수 없음

[식별자 사용 예시]

```
/*
작성자:남승균
작성일:2017.04.03
설 명:Hello World 출력하기
*/
package com.javaex.helloworld;

public class HelloWorld {

    public static void main(String[] args) {
        //Hello World 를 출력합니다.
        System.out.println("Hello World!");
    }
}
```

변수와 자료형

Java Language

: 변수와 자료형

▶ 변수 (Variable)

- ▶ 값(데이터)을 저장하기 위한 메모리 공간
- ▶ 하나의 변수는 하나의 자료형만 지정할 수 있음
- ▶ 값을 저장하고 조회하고 변경할 수 있음

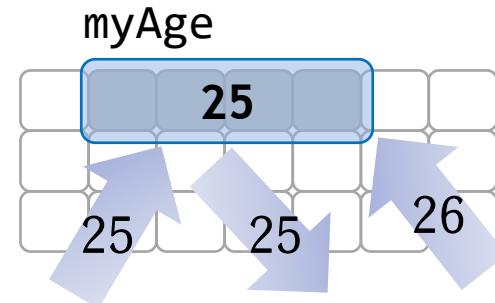
[변수 사용 예시]

-개념-

내 나이를 저장할래
내 나이는 25
내 나이는?

나이를 26 으로 변경

-메모리-



-코드-

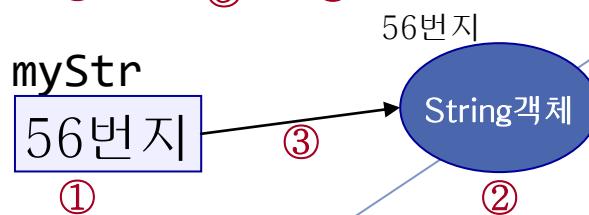
```
int myAge; // 선언  
myAge = 25; // 초기화  
System.out.println(myAge);  
// 조회  
myAge = 26; // 변경
```

Java Language

: 변수와 자료형

▶ 자료형 (Type)

- ▶ 자료형에 따라 저장할 수 있는 값을 종류와 범위가 결정
- ▶ 변수를 사용하는 도중 변경할 수 없음

기본자료형 (primitive data types)	참조자료형 (reference data types)
<ul style="list-style-type: none">✓ 최소 단위의 자료형 다른 자료형으로 분해되지 않음✓ 메소드 없이 값만 가짐✓ int, float, double, char 등	<ul style="list-style-type: none">✓ 여러 자료형들의 집합 구성된 클래스의 객체를 참조✓ 데이터와 메소드를 가짐✓ String, Integer, ArrayList 등
<code>int myInt = 19;</code>  myInt 19	<code>String myStr = new String();</code>  myStr 56번지 56번지 → String객체 ① ③ ②

Java Language

: 변수와 자료형

▶ 기본 자료형(primitive data types)

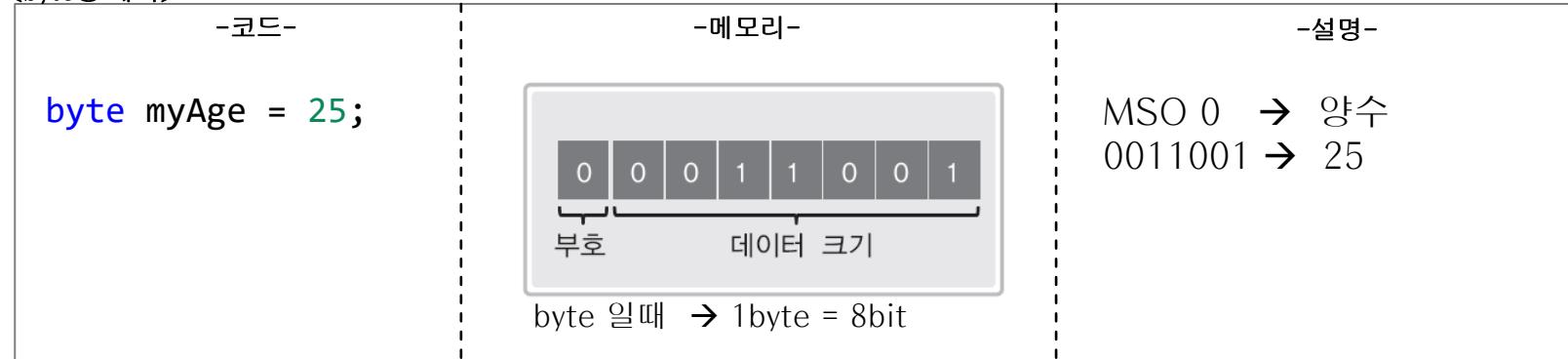
자료형	키워드	크기	표현 범위	사용 예
논리형	boolean	1byte	true OR false(0과 1이 아니다)	<code>boolean isFun = true;</code>
문자형	char	2byte	모든 유니코드 문자 (\u0000~\uFFFF, 0~65535)	<code>char c = 'f';</code>
정수형	byte	1byte	-128 ~ 127	<code>byte b = 89;</code>
	short	2byte	-32,768 ~ 32,767	<code>short s = 32760;</code>
	int	4byte	-2,147,483,648 ~ 2,147,483,647	<code>int x = 59;</code>
	long	8byte	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	<code>long big = 345678912345L;</code>
실수형	float	4byte	-3.4E38 ~ 3.4E38	<code>float f = 32.5F;</code>
	double	8byte	-1.7E308 ~ 1.7E308	<code>double d = 2.3e10;</code>

Java Language

: 기본 자료형 - short, int, long (정수형)

- ▶ 가장 왼쪽 비트(MSB: Most Significant Bit)는 부호를 표시하는 비트
- ▶ MSB를 제외한 나머지는 데이터의 크기를 나타냄
- ▶ 처리할 수 있는 수의 범위에 따라 short < int < long으로 구분

[byte형 예시]



[int형 예시]

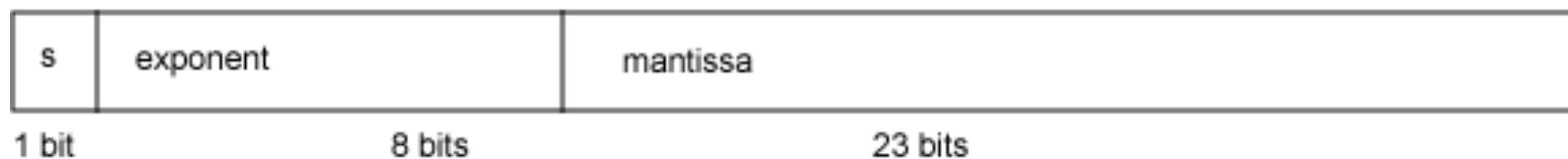


Java Language

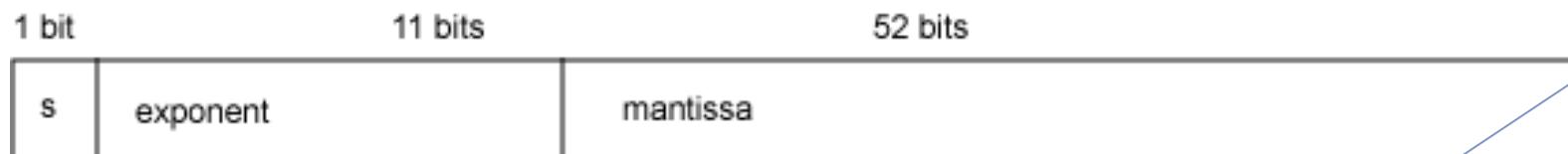
: 기본 자료형 - float, double (실수형)

- ▶ 정수형 자료형과 값 저장방식이 다름 : 더 큰 값을 저장할 수 있음
- ▶ 실수형 저장을 위한 기본 식
- ▶ $(+/-) m * 10^n$

IEEE Floating Point Representation



IEEE Double Precision Floating Point Representation



Java Language

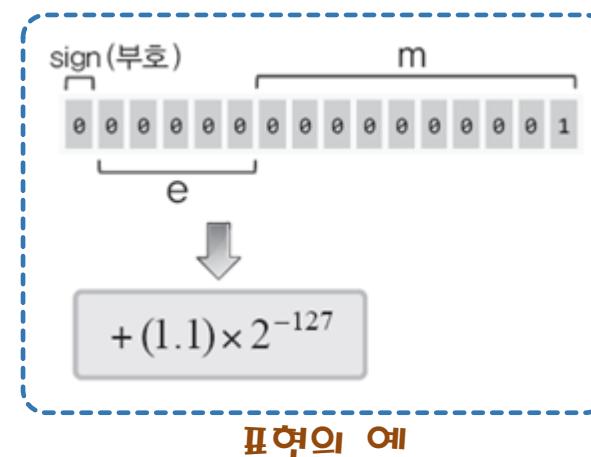
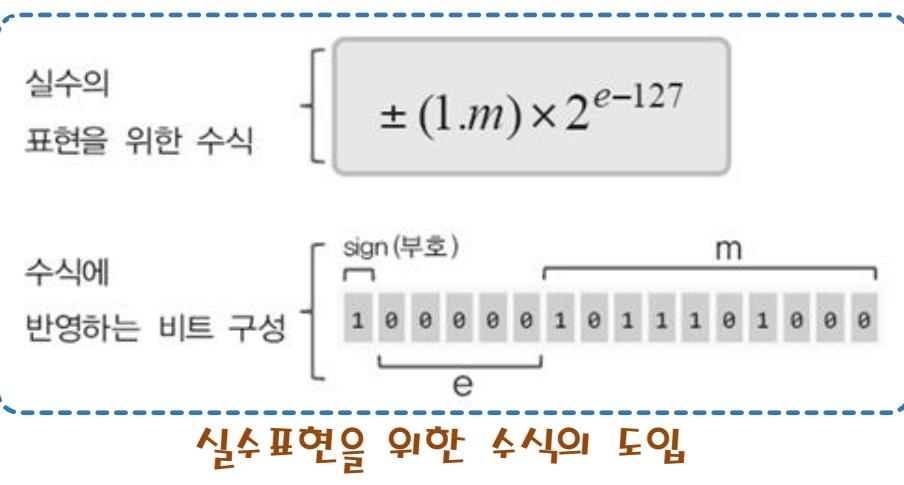
: 기본 자료형 - float, double (실수형)

- ▶ 실수 표현의 문제

- ▶ 0과 1사이의 실수만 해도 그 수가 무한대
- ▶ 바이트 추가로 모든 수의 표현은 불가능

- ▶ Solution:

- ▶ 정밀도를 포기하고 표현할 수 있는 값의 범위를 넓히자



Java Language

: 기본 자료형 - boolean (논리형)

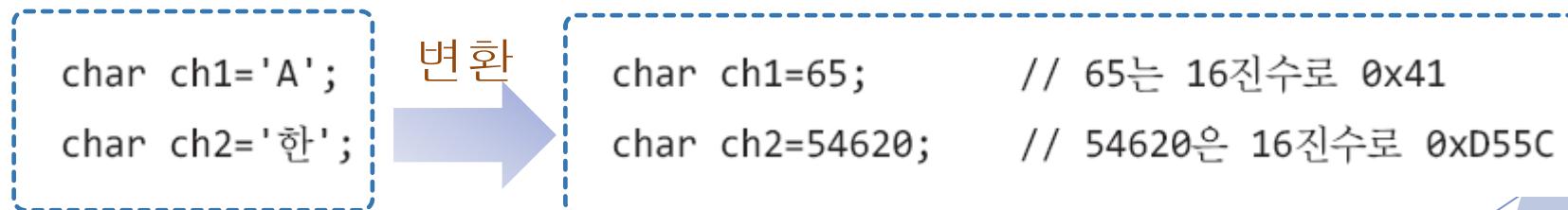
- ▶ 논리값(true/false)을 저장하는 자료형
- ▶ 값에 따라 조건문과 제어문의 흐름을 변경할 때 사용
 - ▶ true : ‘참’을 의미하는 키워드 혹은 조건을 만족함
 - ▶ false : ‘거짓’을 의미하는 키워드 혹은 조건을 만족하지 않음
- ▶ 키워드 true와 false의 이해
 - ▶ 숫자의 관점에서 이해하지 말자
 - ▶ Java에서 true/false는 그 자체로 저장이 가능한 데이터

Java Language

: 기본 자료형 - char (문자형)

▶ 문자형

- ▶ 문자 하나를 유니코드 기반으로 표현
- ▶ 유니코드 : 전 세계의 문자를 표현할 수 있는 국제표준 코드 집합
- ▶ 문자는 작은 따옴표(')로 묶어서 지정
- ▶ 문자는 **char**형 변수에 저장한다. 저장시 실제는 유니코드 값 저장



Java Language

: 기본 자료형 - char(문자형)

	00	01	02	03	04	05	06
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACI</u> 000
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0011
20	<u>SP</u> 0020	!	"	#	\$	%	&
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 003
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 004
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 005
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076

AC0	AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8
가	감	갠	갰	갈	깥	깯	거	검
AC00	AC10	AC20	AC30	AC40	AC50	AC60	AC70	AC80
각	갑	갭	갱	갉	같	깕	걱	겁
AC01	AC11	AC21	AC31	AC41	AC51	AC61	AC71	AC81
깎	깎	깎	깎	깎	깎	깎	깎	깎
AC02	AC12	AC22	AC32	AC42	AC52	AC62	AC72	AC82
갚	갓	갠	겻	갚	갛	깃	겄	겄
AC03	AC13	AC23	AC33	AC43	AC53	AC63	AC73	AC83
₩	₩	₩	₩	₩	₩	₩	₩	₩
0067	0068	0069	006A	006B	006C	006D	006E	006F
₩	₩	₩	₩	{		}	~	DEL
0077	0078	0079	007A	007B	007C	007D	007E	007F

연습문제

: 변수와 자료형

- ▶ 괄호 안에 적절한 데이터 타입을 기술하시오

```
(    ) number;      // 학번  
(    ) name;        // 이름  
(    ) isEnrolled; // 등록 여부  
(    ) grade;       // 평점  
(    ) address;     // 주소  
(    ) major;        // 전공  
(    ) unit;        // 이수 학점  
(    ) haveMinor;   // 부전공 여부  
(    ) juminNo;     // 주민번호(-없이 13자리숫자)  
(    ) cellphone;   // 핸드폰 번호(-포함한 숫자)  
(    ) age;          // 나이  
(    ) email;        // 이메일주소
```

상수와 형변환

Java Language

: 상수 (Constant)

- ▶ 변경할 수 없는 고정된 데이터
 - ▶ 할당, 조회는 되나 변경은 할 수 없음
- ▶ 코드의 이해와 변경이 쉬움
- ▶ 정의
 - ▶ static final로 선언
 - ▶ 대문자로 표현 (관례)
 - ▶ 여러 단어가 겹칠 경우 _로 구분

상수 선언 예제

```
static final double PI = 3.14159;  
static final int MAXIMUM_SPEED = 110;
```

Java Language

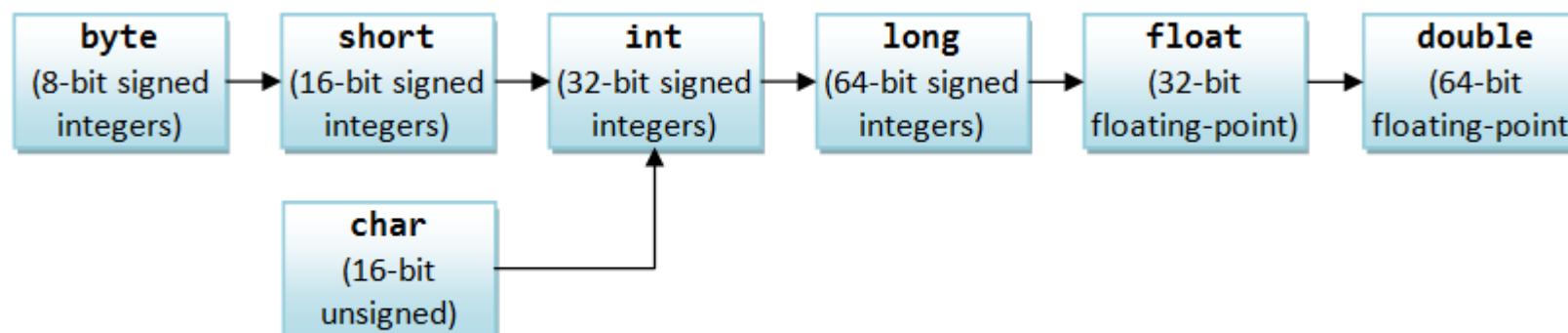
: 형 변환 (Type Casting)

▶ 암묵적 형 변환 (자동 타입 변환: Promotion)

- ▶ 자료의 범위가 좁은 자료형에서 넓은 자료형으로의 변환은 시스템이 자동으로 행함

암묵적 형변환

```
int num1 = 2;  
float num2 = 1.2F;  
float multiply = num1 * num2;
```



Orders of Implicit Type-Casting for Primitives

Java Language

: 형 변환 (Type Casting)

▶ 명시적 형 변환 (강제 타입 변환: Casting)

- ▶ 자료의 범위가 넓은 자료형에서 좁은 자료형으로 변환은 프로그래머가 강제로 변환해야 함
- ▶ 이때, 자료의 유실이 일어날 수 있으므로 주의해야 함

[정상변환X]

```
int intValue = 103029770;  
byte byteValue = (byte)intValue; //강제타입변환
```

00000110 00100100 00011100 00001010 → 10

[정상변환]

```
int intValue = 10;  
byte byteValue = (byte)intValue; //강제타입변환
```

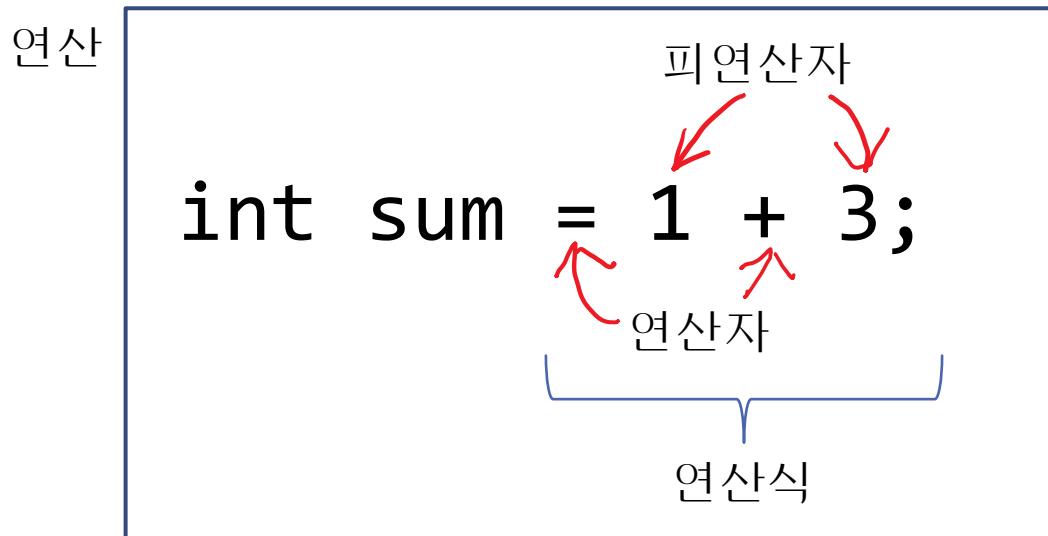
00000000 00000000 00000000 00001010 → 10

연산자

Java Language

: 연산자 (Operator)

- ▶ 연산 (Operation) : 데이터를 처리하여 결과를 산출하는 것
- ▶ 연산자 (Operator) : 연산에 사용되는 표시나 기호 (데이터를 처리하는 기능을 수행)
- ▶ 피연산자 (Operand) : 연산 되는 데이터
- ▶ 연산식 (Expression) : 연산자와 피연산자를 이용, 연산 과정을 기술한 것

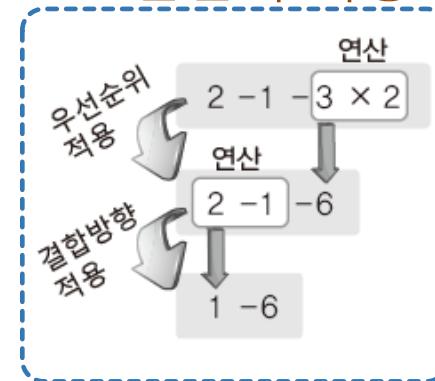


Java Language

: 연산자 우선순위

연산기호	결합방향	우선순위
[], .	→	1(높음)
expr++, expr--	←	2
++expr, --expr, +expr, -expr, ~, !, (type)	←	3
*, /, %	→	4
+, -	→	5
<<, >>, >>>	→	6
<, >, <=, >=, instanceof	→	7
==, !=	→	8
&	→	9
^	→	10
	→	11
&&	→	12
	→	13
? expr : expr	←	14
=, +=, -=, *=, /=, %=, &=, ^=, =, <<=, >>=	←	15(낮음)

연산의 과정



Java Language

: 대입연산자(=)와 산술연산자(+, -, *, /, %)

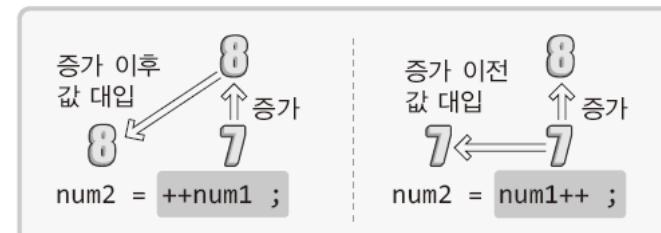
연산자	연산자의 기능	결합방향
=	연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입한다. 예) val = 20;	◀
+	두 피연산자의 값을 더한다. 예) val = 4 + 3;	→
-	왼쪽의 피연산자 값에서 오른쪽의 피연산자 값을 뺀다. 예) val = 4 - 3;	→
*	두 피연산자의 값을 곱한다. 예) val = 4 * 3;	→
/	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눈다. 예) val = 7 / 3;	→
%	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눴을 때 얻게 되는 나머지를 반환한다. 예) val = 7 % 3	→

Java Language

: 증가, 감소 연산자

값의 변경 순서에 유의하자

연산자	연산자의 기능	결합방향
++ (prefix)	피연산자에 저장된 값을 1 증가 예) val = ++n;	←
-- (prefix)	피연산자에 저장된 값을 1 감소 예) val = --n;	←
연산자	연산자의 기능	결합방향
++ (postfix)	피연산자에 저장된 값을 1 증가 예) val = n++;	←
-- (postfix)	피연산자에 저장된 값을 1 감소 예) val = n--;	←



증가 이후
값 대입
8
7
증가
num2 = **++num1** ;

증가 이전
값 대입
8
7
증가
num2 = num1**++** ;

Java Language

: 관계 연산자

연산자	연산자의 기능	결합방향
<	예) $n1 < n2$ $n1$ 이 $n2$ 보다 작은가?	→
>	예) $n1 > n2$ $n1$ 이 $n2$ 보다 큰가?	→
\leq	예) $n1 \leq n2$ $n1$ 이 $n2$ 보다 같거나 작은가?	→
\geq	예) $n1 \geq n2$ $n1$ 이 $n2$ 보다 같거나 큰가?	→
$==$	예) $n1 == n2$ $n1$ 과 $n2$ 가 같은가?	→
$!=$	예) $n1 != n2$ $n1$ 과 $n2$ 가 다른가?	→

Java Language

: 논리 연산자

- ▶ AND (`&&`), OR (`||`), NOT (`!`)
- ▶ 결과는 boolean 타입

a	b	<code>a&&b</code>	<code>a b</code>	<code>!a</code>
False	False	False	False	True
False	True	False	True	
True	False	False	True	False
True	True	True	True	

Java Language

: 비트 연산자

- 정수형 데이터를 비트 단위로 개별 조작

비트 A	비트 B	비트 A & 비트 B
1	1	1
1	0	0
0	1	0
0	0	0

비트 A	비트 B	비트 A 비트 B
1	1	1
1	0	1
0	1	1
0	0	0

비트 A	비트 B	비트 A ^ 비트 B
1	1	0
1	0	1
0	1	1
0	0	0

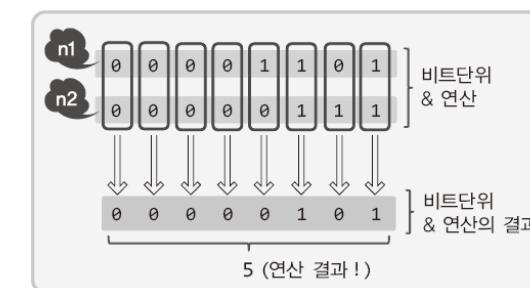
비트	~비트
1	0
0	1

~(NOT)

&(and)

|(OR)

반만true
^(XOR)



Java Language

: 비트 시프트(Shift) 연산자

연산자	연산자의 기능	결합방향
<code><<</code>	<ul style="list-style-type: none">피연산자의 비트 열을 왼쪽으로 이동이동에 따른 빈 공간은 0으로 채움예) <code>n << 2;</code> → n의 비트 열을 두 칸 왼쪽으로 이동 시킨 결과 반환	→
<code>>></code>	<ul style="list-style-type: none">피연산자의 비트 열을 오른쪽으로 이동이동에 따른 빈 공간은 음수의 경우 1, 양수의 경우 0으로 채움예) <code>n >> 2;</code> → n의 비트 열을 두 칸 오른쪽으로 이동 시킨 결과 반환	→
<code>>>></code>	<ul style="list-style-type: none">피연산자의 비트 열을 오른쪽으로 이동이동에 따른 빈 공간은 0으로 채움예) <code>n >>> 2;</code> → n의 비트 열을 두 칸 오른쪽으로 이동 시킨 결과 반환	→

✓ 비트연산의 특징

- 왼쪽으로의 비트 열 이동은 2의 배수의 곱
- 오른쪽으로의 비트 열 이동은 2의 배수의 나눗셈

- 정수 2 → 00000010 → 정수 2
- $2 << 1 \rightarrow 00000100 \rightarrow$ 정수 4
- $2 << 2 \rightarrow 00001000 \rightarrow$ 정수 8
- $2 << 3 \rightarrow 00010000 \rightarrow$ 정수 16

Java Language

: 3항 연산자(Conditional Operator)

- ▶ 세 개의 피연산자를 필요로 하는 연산자
- ▶ 조건식에 따라 **true**면 : 앞쪽의 연산식을, **false**면 : 뒤쪽의 피연산자를 선택한다

조건식 ? 값 또는 연산식 : 값 또는 연산식

조건식이 **true** 일 때 조건식이 **false** 일 때



Java Programming

Input and Output

콘솔 입출력

Java Input and Output

: 콘솔 출력

- ▶ System.out.println 메서드 : 출력 후 개행을 함
- ▶ System.out.print 메서드 : 출력 후 개행을 하지 않음

```
System.out.print("안녕");
System.out.println("하세요"); //안녕하세요
```

```
System.out.println("안녕하세요"); //안녕하세요
```

- ▶ 이스케이프 시퀀스 : 문자열 내 특별한 의미로 해석되는 문자 (\로 시작)

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

Java Input and Output

: 콘솔 입력 - Scanner

- ▶ Scanner 클래스 : 다양한 리소스를 대상으로 입력을 받을 수 있도록 정의된 클래스

```
import java.util.Scanner;  
// ... 중략  
Scanner scanner = new Scanner(System.in);  
int value = scanner.nextInt();  
System.out.println(value);  
scanner.close();
```

입력받을 값에 맞는 메서드 선택

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

✓ Scanner 클래스생성자

```
Scanner(File source)  
Scanner(InputStream source)  
Scanner(String source)  
Scanner(System.in)
```

Scanner 클래스는 단순히 키보드의 입력만을 목적으로 디자인된 클래스가 아니다.

연습문제

: 콘솔 입출력

[문제]

이름을 입력받아 출력하는 프로그램을 작성하세요

[문제]

이름과 나이를 입력받아 출력하는 프로그램을 작성하세요

```
<terminated> Console01 [Java Application] /Library/Java/JavaVi
이름을 입력해 주세요
이름:홍길동
당신의 이름은 홍길동입니다.
```

```
<terminated> Console01 [Java Application] /Library/Java/JavaVi
이름을 입력해 주세요
이름:홍길동
나이:23
당신의 이름은 홍길동, 나이는 23입니다.
```

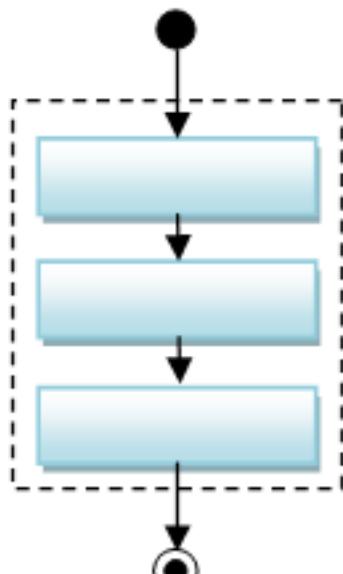
Java Programming

Java Conditionals

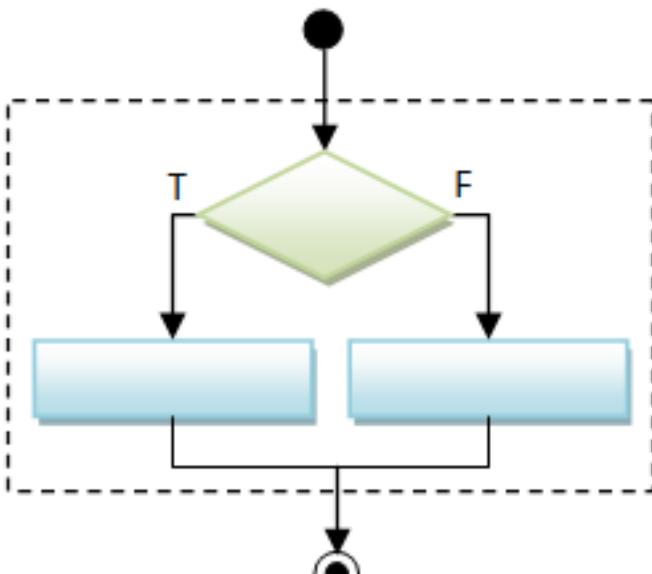
조건문

Flow Control

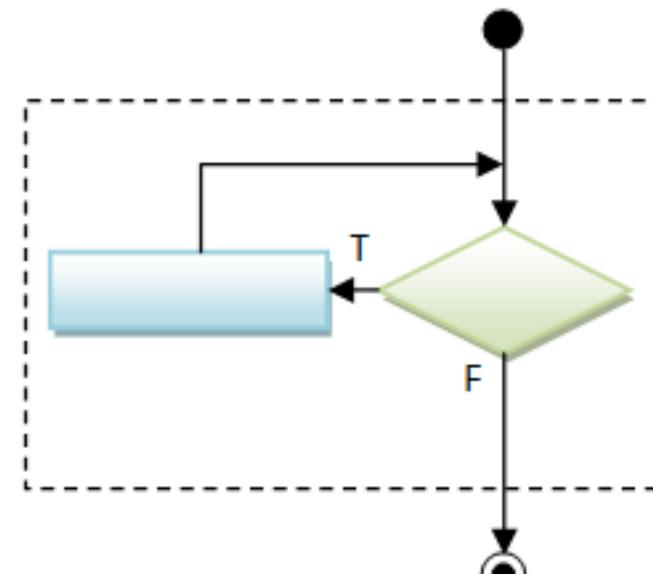
: Sequential vs Conditional vs Loop



Sequential



Conditional (Decision)



Loop (Iteration)

Java Conditionals

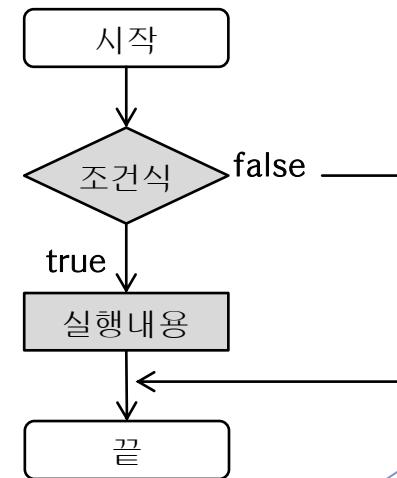
: if 조건문

```
if ( 조건식 ) {  
    /* 조건식이 true 이면 실행되는 영역 */  
}
```

어떤 조건에 의해 흐름을 결정하는 것이어서
Decision Making이라 부르기도 한다

[예제]

점수를 입력받아
점수가 60점 이상이면 “합격입니다.” 를 출력하세요



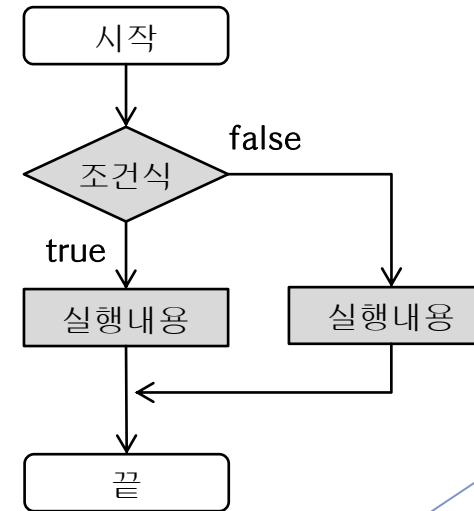
Java Conditionals

: if ~ else 조건문

```
if ( 조건식 ) {  
    /* 조건식이 true 이면 실행되는 영역 */  
} else {  
    /* 조건식이 false 이면 실행되는 영역 */  
}
```

[예제]

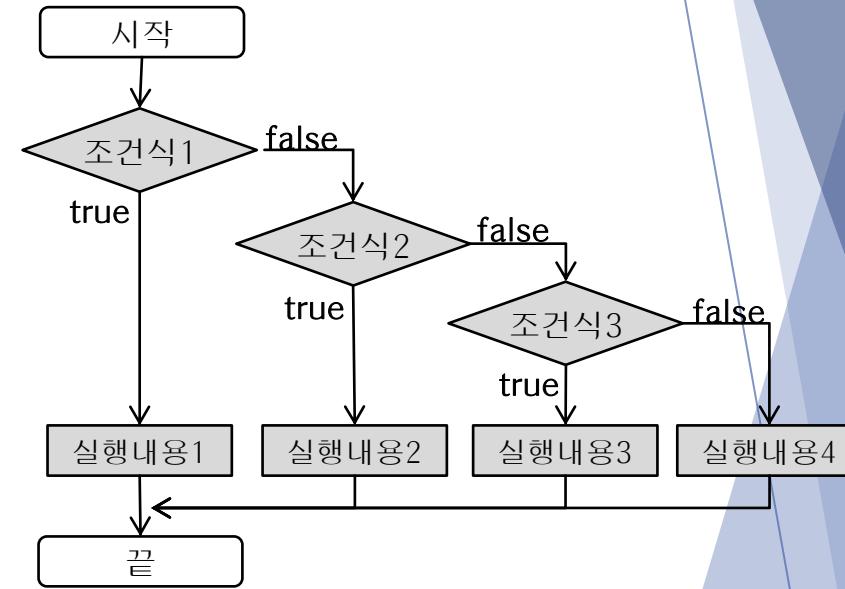
점수를 입력 받아
60점 이상이면 “합격입니다.”
60점 미만이면 “불합격입니다.”
를 출력하세요



Java Conditionals

: if ~ else if ~ else 조건문

```
if ( 조건식1 ) {  
    /* 조건식1 이 true 이면 실행되는 영역 */  
} else if (조건식2) {  
    /* 조건식2 가 true 이면 실행되는 영역 */  
} else if (조건식3) {  
    /* 조건식3 이 true 이면 실행되는 영역 */  
} else {  
    /* 위의 조건이 모두 해당되지 않으면 실행되는 영역 */  
}
```



[예제]

숫자를 입력받아
수가 0보다 크면 “양수” 영보다 작으면 “음수” 0일때는 “0입니다.”을 출력하세요

연습문제

: 조건문 연습

[문제]

숫자를 입력받아 아래와 같이 출력되는
프로그램을 작성하세요

입력받은 수가

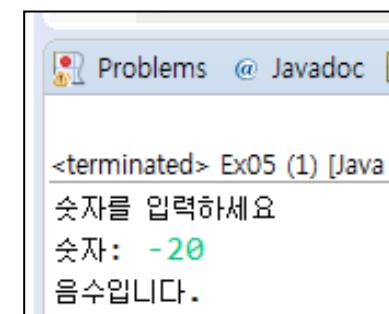
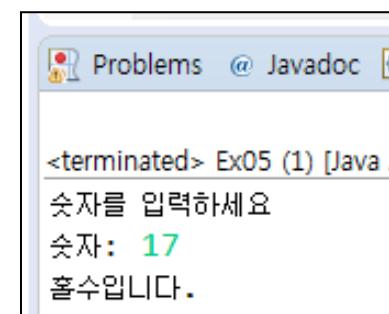
양수일때

짝수이면 “짝수” 출력

홀수 이면 “홀수” 출력

음수이면 “음수” 라고 출력

0 이면 “0” 으로 출력



연습문제

: if ~ else if ~ else 연습

[문제]

과목번호를 입력받아 강의실 번호를 출력하는 프로그램을 작성하세요

과목 code값이

1이면 "R101호 입니다."

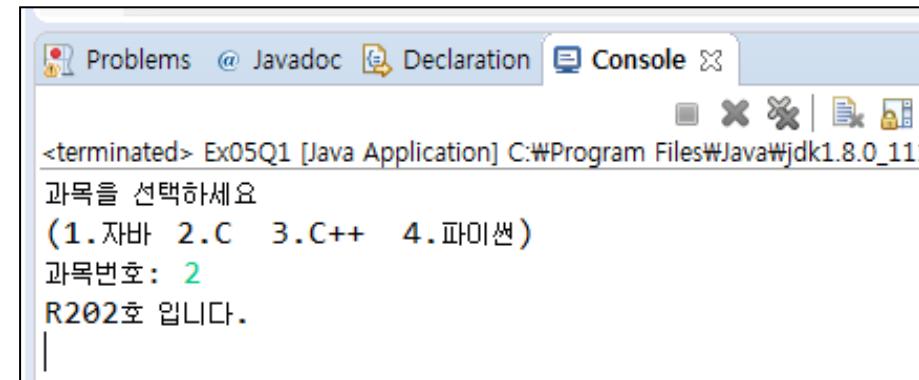
2이면 "R202호 입니다."

3이면 "R303호 입니다."

4이면 "R404호 입니다."

나머지는 "상담원에게 문의하세요"

를 출력하세요

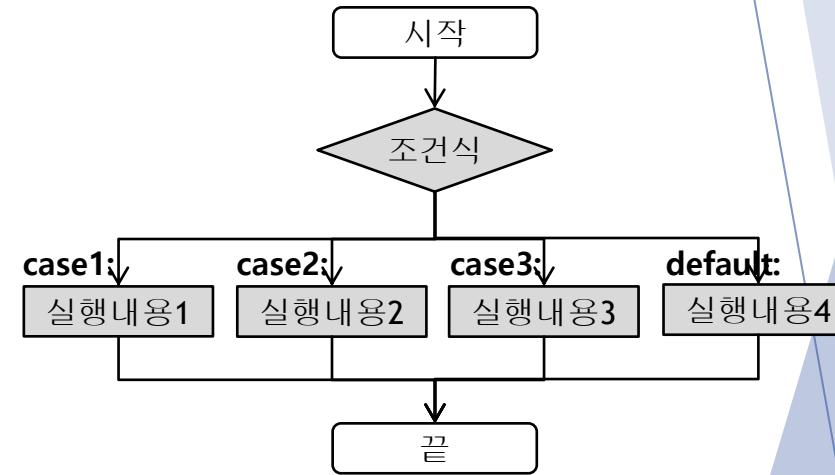


The screenshot shows a Java application window titled 'Ex05Q1 [Java Application]'. The console tab is active, displaying the following text:
<terminated> Ex05Q1 [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin
과목을 선택하세요
(1. 자바 2. C 3. C++ 4. 파이썬)
과목번호: 2
R202호 입니다.

Java Conditionals

: switch ~ case 조건문

```
switch (변수) {  
    case 값1 :  
        /* 변수 값이 값1일 때 실행내용*/  
        break;  
    case 값2 :  
        /* 변수 값이 값2일 때 실행내용*/  
        break;  
    case 값3 :  
        /* 변수 값이 값3일 때 실행내용*/  
        break;  
    default :  
        /* 해당 내용이 없을 때 실행내용*/  
        break;  
}
```



case의 값에는 수치형 뿐 아니라, char, String 도 올 수 있다

연습문제

: switch ~ case 연습

- ▶ if 연습문제와 같은 문제지만 switch ~ case를 이용하여 풀어 봅시다

[문제]

과목번호를 입력받아 강의실 번호를 출력하는 프로그램을 작성하세요

과목 code값이
1이면 "R101호 입니다."
2이면 "R202호 입니다."
3이면 "R303호 입니다."
4이면 "R404호 입니다."
나머지는 "상담원에게 문의하세요"
를 출력하세요

```
<terminated> Ex05Q1 [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin
과목을 선택하세요
(1.자바 2.C 3.C++ 4.파이썬)
과목번호: 2
R202호 입니다.
```

if-else 문의 조건식 부분이 모두 == 로 표현될 때 switch문을 사용

연습문제

: switch ~ case 연습

[문제]

월을 입력받아 해당월의 일수를 출력하는
프로그램을 작성하세요

1월이면 "31일"
2월이면 "28일"
3월이면 "31일"
4월이면 "30일"
5월이면 "31일"
6월이면 "30일"
7월이면 "31일"
8월이면 "31일"
9월이면 "30일"
10월이면 "31일"
11월이면 "30일"
12월이면 "31일"

```
Problems @ Javadoc Declaration Console X
<terminated> Ex06 [Java Application] C:\Program Files\Java\j...
월을 입력하세요
2
28일 입니다.
```

```
Problems @ Javadoc Declaration Console X
<terminated> Ex06 [Java Application] C:\Program Files\Java\j...
월을 입력하세요
9
30일 입니다.
```

```
Problems @ Javadoc Declaration Console X
<terminated> Ex06 [Java Application] C:\Program Files\Java\j...
월을 입력하세요
5
31일 입니다.
```

연습문제

: 조건문 연습

[문제]

점수를 입력받아 입력된 수가 3의 배수인지 판별하는 프로그램을 작성하세요

```
Problems @ Javadoc Declaration Console 
<terminated> Ex01 (3) [Java Application] C:\Program Files\Java\jdk1.8.0_111\
점수를 입력하세요
15
3의 배수입니다.
```

```
Problems @ Javadoc Declaration Console 
<terminated> Ex01 (3) [Java Application] C:\Program Files\Java\jdk1.8.0_111\
숫자를 입력하세요
55
55은 3의 배수가 아닙니다.
```

[문제]

점수를 입력받아 등급을 표시하는 프로그램을 작성하세요

90점 이상 이면 "A등급"
80점 이상~89점 이면 "B등급"
70점 이상~79점 이면 "C등급"
60점 이상~69점 이면 "D등급"
60점 미만이면 "F등급"

```
Problems @ Javadoc Declaration Console 
<terminated> Ex07Q [Java Application] C:\Program Files\Java\jdk1\
점수를 입력하세요 : 94
A등급
```

```
Problems @ Javadoc Declaration Console 
<terminated> Ex07Q [Java Application] C:\Program Files\Java\jdk1\
점수를 입력하세요 : 78
C등급
```

if문과 switch문 비교

```
switch (month) {  
    case 2:  
        days = 28;  
        break;  
  
    case 4:  
    case 6:  
    case 9:  
    case 11:  
        days = 30;  
        break;  
  
    default:  
        days = 31;  
        break;  
}
```

```
if (month == 2) {  
    days = 28;  
} else if ((month == 4) || (month == 6) || (month == 9)  
|| (month == 11)) {  
    days = 30;  
} else {  
    days = 31;  
}
```

break 없는 switch-case문 짐작하지 않음

Java Programming

Java Loop

반복문

Java Loop

: while 반복문

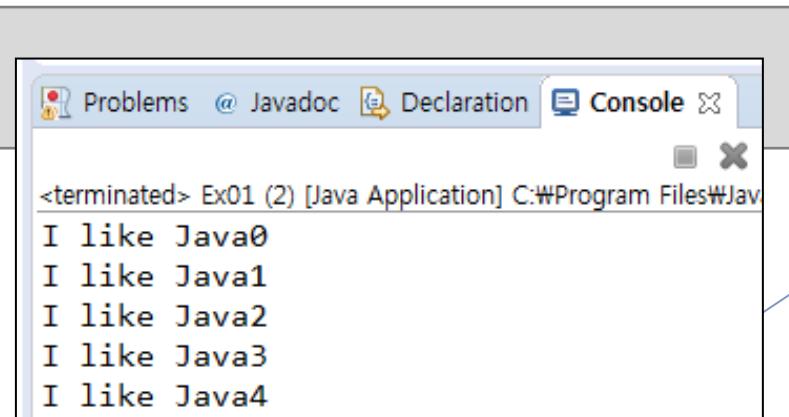
▶ 기본 구문 : `while (condition) { statements }`

- ▶ `condition` 값이 참인 동안 내부 블록을 반복하여 실행함
- ▶ 초기 `condition` 값이 `false` 이면 내부 블록은 실행되지 않음
 - ▶ Condition 값이 계속 `true` 면 무한 반복
 - ▶ 무한 반복은 주의해야 하지만, 의도적으로 무한 반복을 실행하는 경우도 있음

[예제]

다음과 같이 출력되는 프로그램을 작성하세요

```
// 1에서 10까지 숫자를 출력함  
int x = 1;  
while (x <= 10){  
    System.out.println(x);  
    x = x + 1;  
}
```

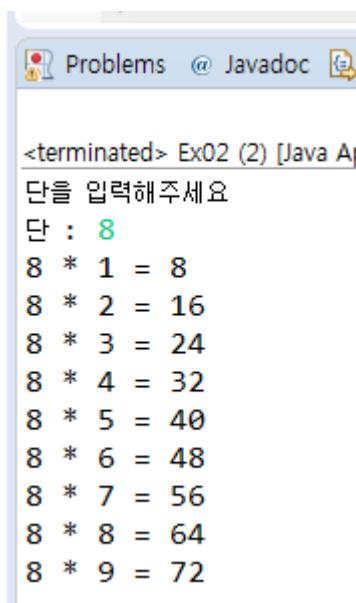


연습문제

: while 반복문 구구단

[문제]

숫자를 입력받아 입력한 숫자(단)의
구구단을 출력하세요



The screenshot shows an IDE interface with a 'Problems' tab selected. Below it, the code for 'Ex02.java' is displayed. The code prompts the user for a number and then prints its multiplication table from 1 to 9.

```
Problems @ Javadoc

<terminated> Ex02 (2) [Java App]

단을 입력해주세요
단 : 8
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
```

```
public class Ex02 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int dan ;
        int i = 1;

        //메세지출력, dan값 입력
        while(//조건문){
            //구구단 출력 코드
        }
        sc.close();
    }
}
```

Java Loop

: do ~ while 반복문

- ▶ do 블록을 우선 한번 수행하고 반복 여부를 while에서 확인
 - ▶ 적어도 한 번은 수행되어야 하는 반복 구문에 사용
 - ▶ while 문에서 확인할 조건이 반복 구문 내에서 할당되는 경우

[문제]

사용자의 숫자를 입력받아 더하는 프로그램을 작성하세요(0이면 종료)

The image shows two side-by-side Java application windows. Both windows have the title 'Problems @ Javadoc Declarations' and status 'terminated'.

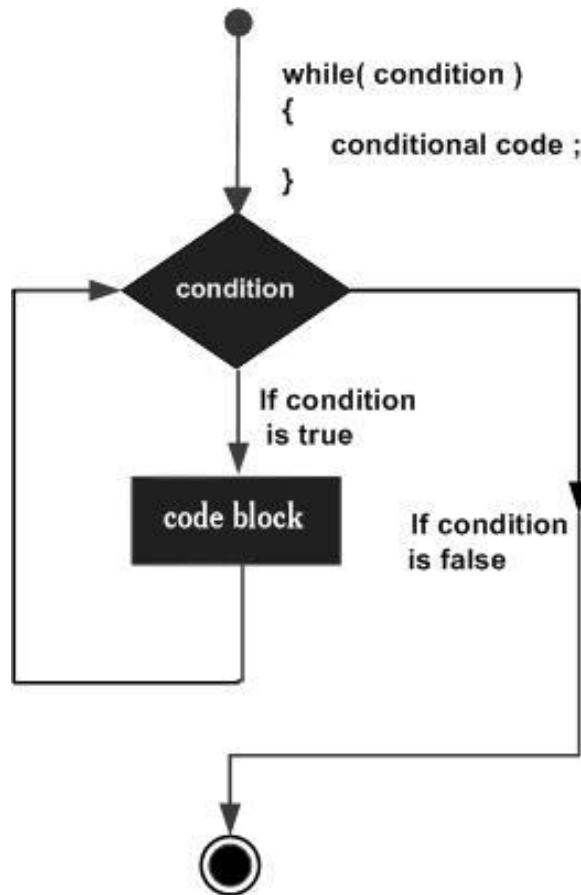
Left Window:
Prompt: 숫자를 입력하세요. [0이면 종료]
Input: 4
Output: 합계: 4
Input: 5
Output: 합계: 9
Input: 6
Output: 합계: 15
Input: 0
Output: 합계: 15
Output: 종료

Right Window:
Prompt: 숫자를 입력하세요. [0이면 종료]
Input: 0
Output: 합계: 0
Output: 종료

```
int total = 0;
int value = 0;
do {
    System.out.print("숫자입력[ 0 to quit ] >");
    value = scanner.nextInt();
    total += value;
} while(value != 0);
```

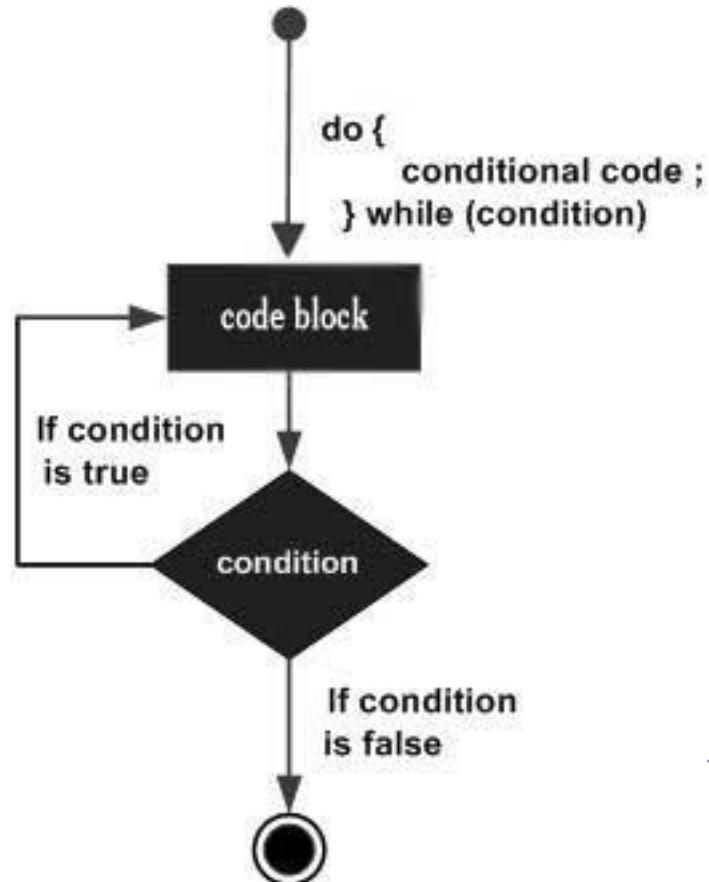
Java Loop

: while vs do ~ while



while loop

루프를 빠져나가는 것은 오롯이
개발자의 몫!
의도한 것이 아니라면 무한루프는 피하도록 한다



do ~ while loop

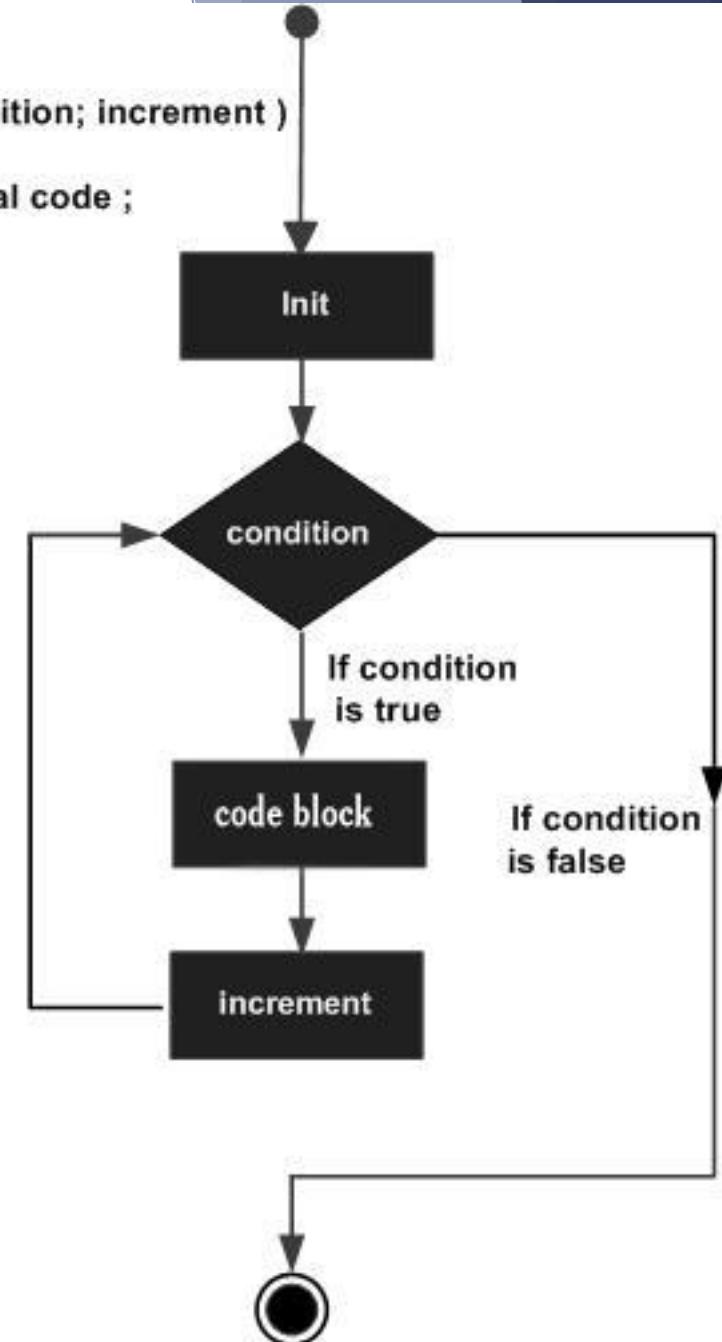
Java Loop

: for 반복문

```
for ( 초기화; 조건검사; 증감연산 ) {  
    statement1;  
    statement2;  
    .....  
}
```

```
for ( int i=0; i<10; i++ ) {  
    a = a + 1;  
}
```

```
for( init; condition; increment )  
{  
    conditional code ;  
}
```



Java Loop

: for vs while

```
int num=0; ①  
while( num<5 ) ②  
{  
    System.out.println( num );  
    num++; ③  
}
```

```
① for( int num=0 ; num<5 ; num++ )  
{  
    System.out.println( num );  
}
```

```
for( 초기화; 조건식; 증감식 ){  
    조건식이 맞을때 실행내용  
}
```

① → 반복의 횟수를 세기 위한 변수 (**초기화**)

② → 반복의 조건 (**조건식**)

③ → 반복의 조건을 무너뜨리기 위한 연산 (**증감식 : 탈출조건**)

Java Loop

: for 반복문 실행 흐름

첫 번째 루프의 흐름

1 → 2 → 3 → 4 [i=1]

두 번째 루프의 흐름

2 → 3 → 4 [i=2]

세 번째 루프의 흐름

2 → 3 → 4 [i=3]

네 번째 루프의 흐름

2 [i=3] 따라서 탈출!

```
1. int i=0 ; 2. i<3 ; 3. i++ )  
{   System.out.println("...");  
}
```

I like Java 0

I like Java 1

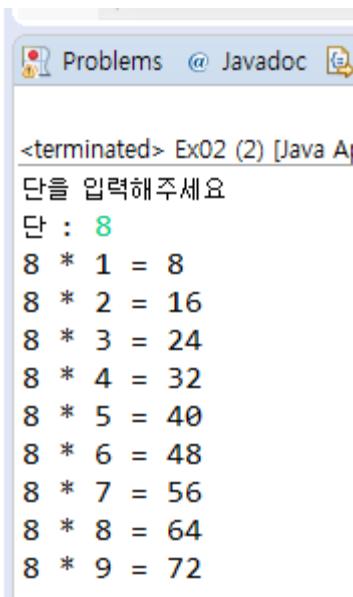
I like Java 2

연습문제

: for 반복문 구구단

[문제]

숫자를 입력받아 입력한 숫자(단)의
구구단을 출력하세요 (for문으로작성)



The screenshot shows an IDE interface with a 'Problems' tab selected. Below it, the code for 'Ex02.java' is displayed. The code prompts the user to input a number (단), initializes a variable 'dan' to that value, and then uses a for loop to print the multiplication table for that number. The output window shows the program running and printing the table for the number 8.

```
<terminated> Ex02 (2) [Java App]
단을 입력해주세요
단 : 8
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
```

```
public class Ex02 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int dan ;
        int i = 1;

        //메시지 출력, 단 입력
        for(//조건문){
            //구구단 출력 코드
        }
        sc.close();
    }
}
```

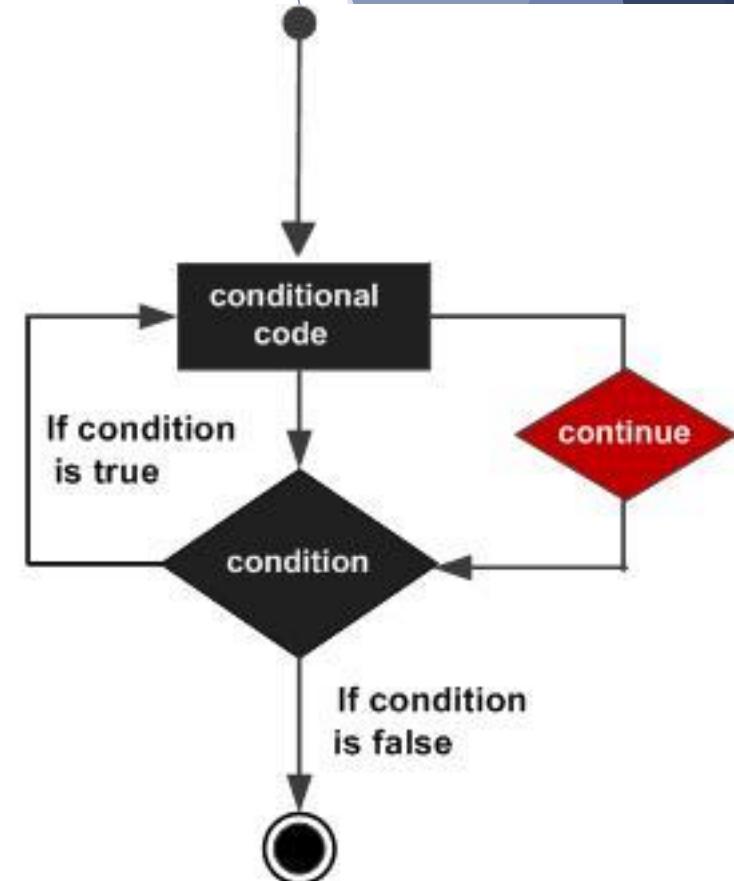
Java Loop

: continue

▶ 흐름 제어 : continue

- ▶ 반복문 내에서 `continue`를 만나면 이후 문장은 수행하지 않음
- ▶ 반복 블록을 벗어나지 않고 블록의 가장 마지막으로 이동, 다시 반복 조건을 검사

```
for ( int i = 0; i < 20; i++ ) {  
    if( i % 2 == 0 || i % 3 == 0 ) {  
        continue;  
    }  
  
    System.out.println( i );  
}
```

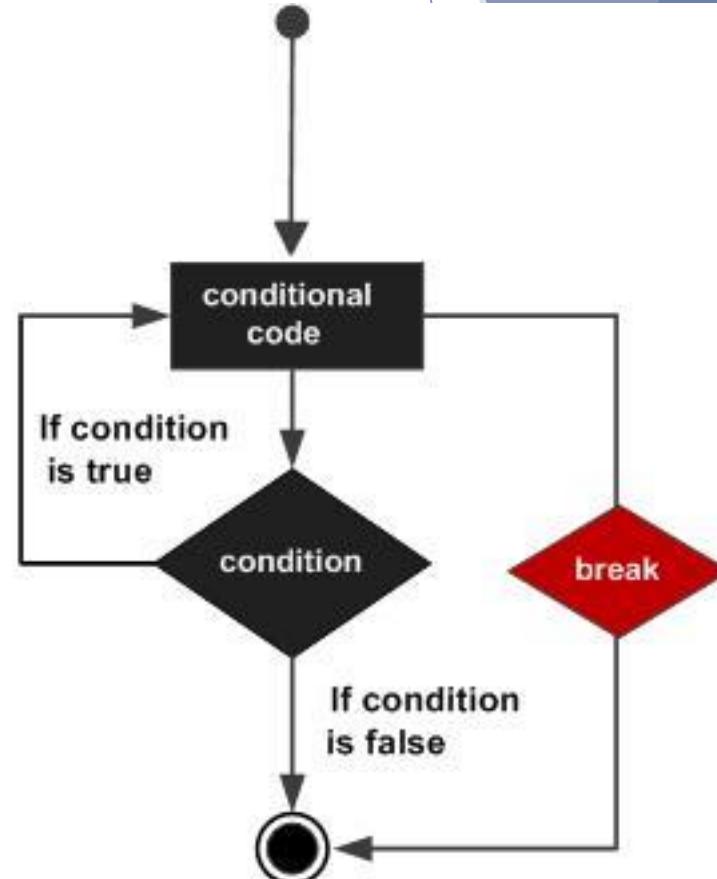


Java Loop

: break

- ▶ 흐름 제어 : break
 - ▶ 반복문 수행을 중단하고 반복 블록 다음 문장을 실행
 - ▶ 중첩된 반복문에서 한 단계씩 반복문을 벗어남

```
while( true ) {  
    sum += num;  
  
    if( sum > 5000 ) {  
        break;  
        num++;  
    }  
}
```

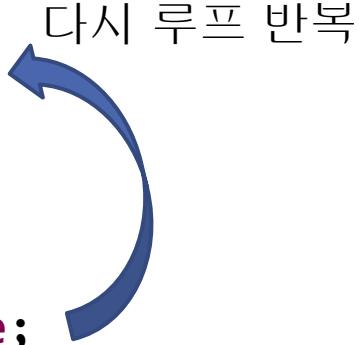


Java Loop

: continue vs break

continue

```
while (true) {  
    //...  
    if (x < 0) {  
        continue;  
    }  
    //...  
}
```



break

```
while( true ) {  
    //...  
    if( x < 0 ) {  
        break;  
    }  
    //...  
}
```



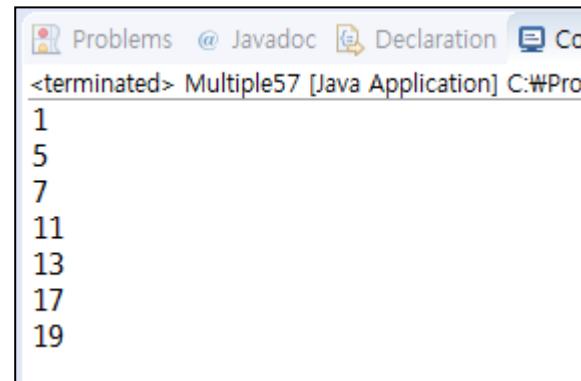
루프 탈출

연습문제

: continue, break

[문제]

1에서 20까지의 수에서 2의배수와 3의배수를 제외한 숫자를 출력하세요
(for문, continue문 사용)



```
Problems @ Javadoc Declaration Corrections
<terminated> Multiple57 [Java Application] C:\#Prog
1
5
7
11
13
17
19
```

연습문제

: 반복문 연습

- ▶ **for** 루프도 좋고, **while** 루프도 좋습니다. 자신있는 루프 방식으로 문제들을 구현해 봅시다

[문제]

다음과 같이 출력되는 구구단을 출력하세요

```
<terminated> Ex06 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_251\bin\java.exe -jar Ex06.jar
2*1 = 2
2*2 = 4
2*3 = 6
[중간생략]
9*7 = 63
9*8 = 72
9*9 = 81
```

[문제]

아래와 같이 출력되는 프로그램을 작성하세요

```
<terminated> Ex08 [Java Application] C:\Program Files\Java\jdk1.8.0_251\bin\java.exe -jar Ex08.jar
*
**
***
****
*****
******
*****
```

생각해 봅시다

: for vs while

- ▶ 아래와 같은 문제가 있다고 가정할 때, for 문이 좋을까요? while 문이 좋을까요?

[문제]

6의 배수이자 14의 배수인 가장 적은 정수 찾기



Hint: 반복의 횟수를 알 수 있을 때 = for
반복의 횟수를 알 수 없을 때 = while

Random

: Math.random()으로 정수값 생성하기

```
int num = (int)(Math.random() *최대값) +최소값;
```

예) 주사위

```
int num = (int)(Math.random()*6)+1
```

- 0.0 이상 1.0 미만인 실수값 반환 (1.0은 포함되지 않는다)
- double 형 값 반환
- double 형을 int 형으로 강제 형변환 하면 소수점 아래가 버림 처리 된다.
 4.175353107765874 → 4
- 정수 난수값 필요시 주로 사용

연습문제

: Math.random()을 이용한 미니 로또

[문제]

1~45 까지의 숫자중 임의의 6개의 숫자를 출력하세요
(중복체크는 하지 말 것)



Problems @ Javadoc Declaration
<terminated> MiniLotto [Java Application] C
6 40 20 39 27 25



Problems @ Javadoc Declaration
<terminated> MiniLotto [Java Application]
15 15 23 29 43 26

for 혹은 while 문을 이용하여 1~45사이의 정수값 여섯 개를 생성한다

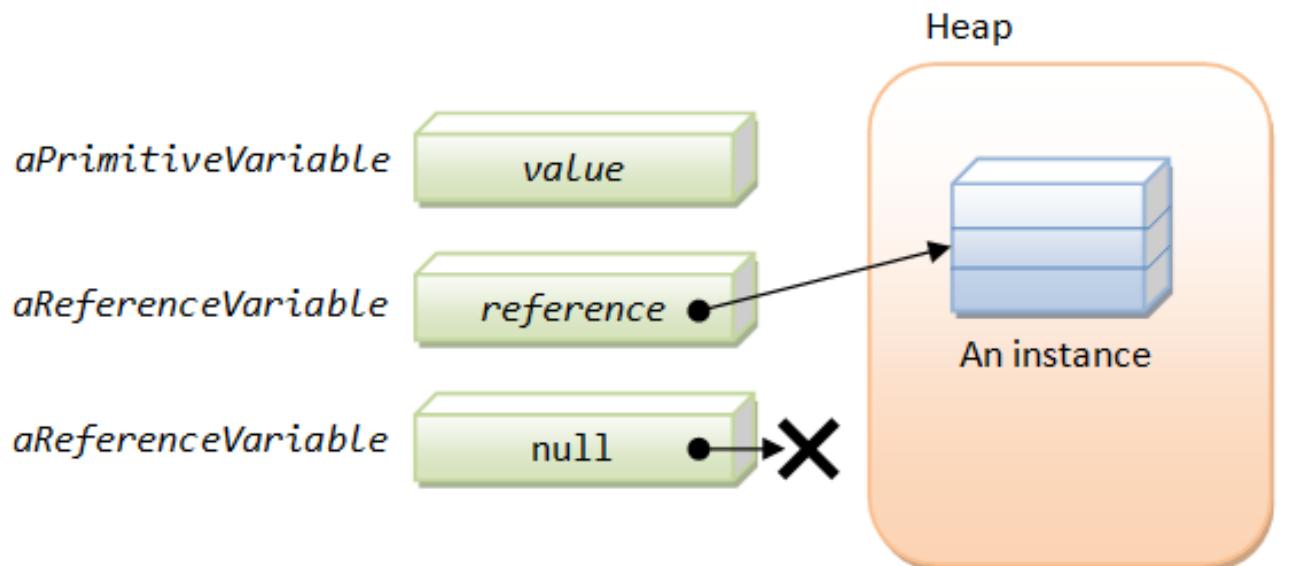
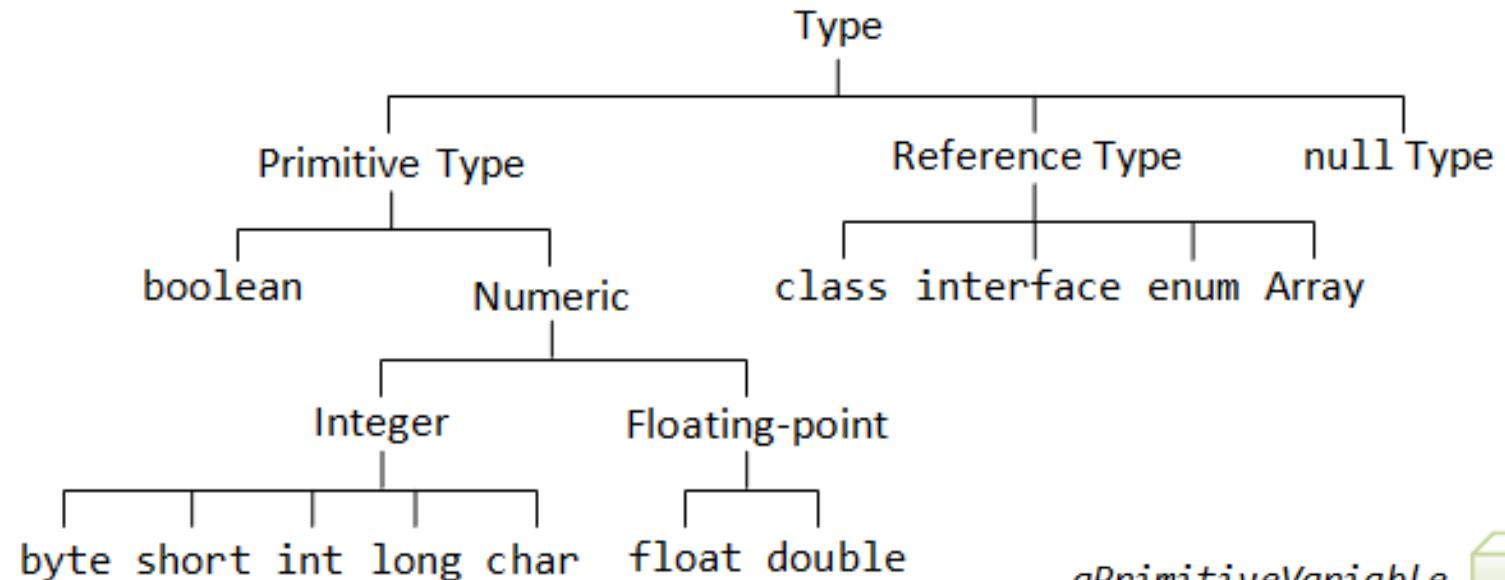
Java Programming

Reference Type

참조 타입

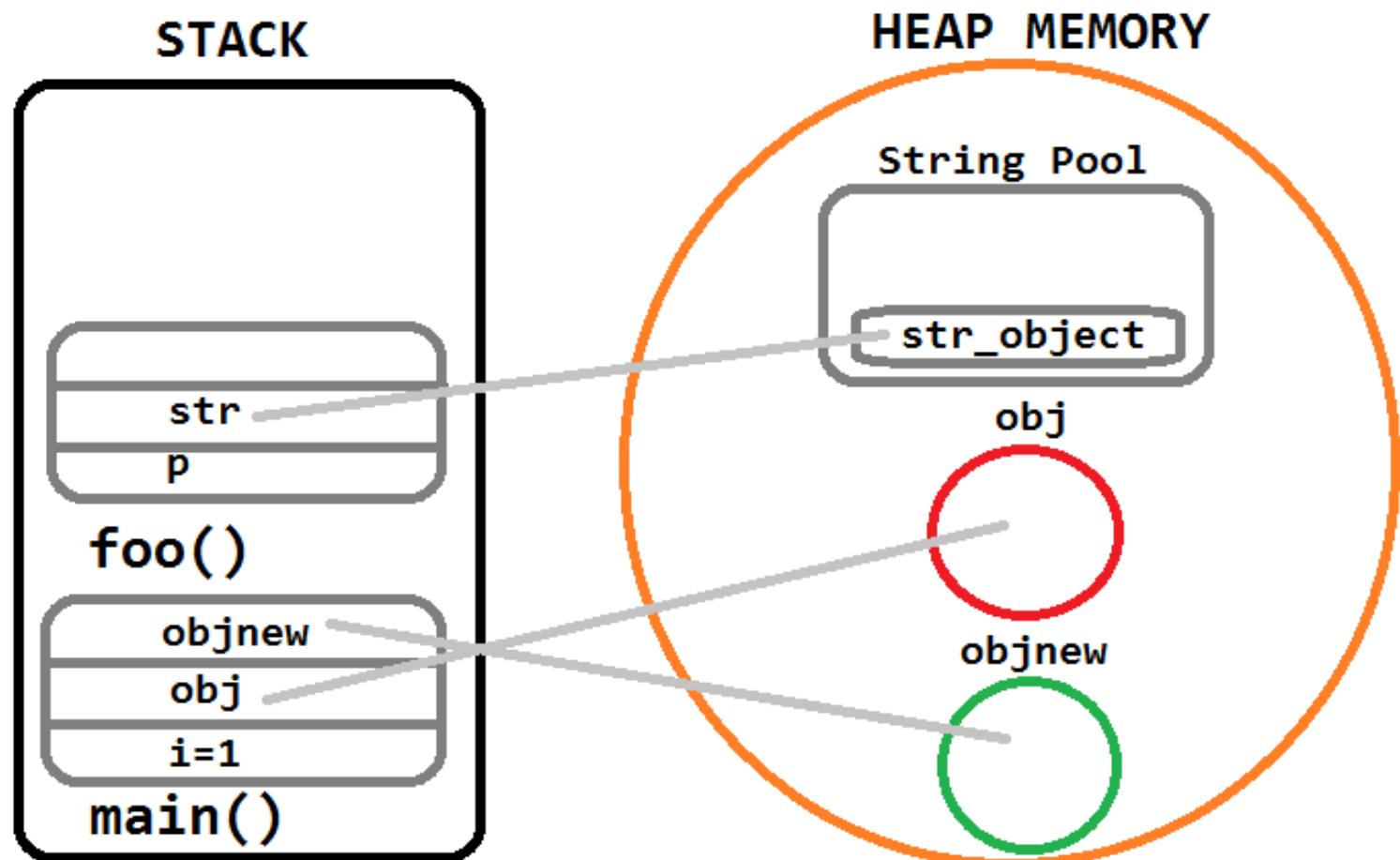
Reference Type

: Primitive Type vs Reference Type



Java의 메모리 사용 구조

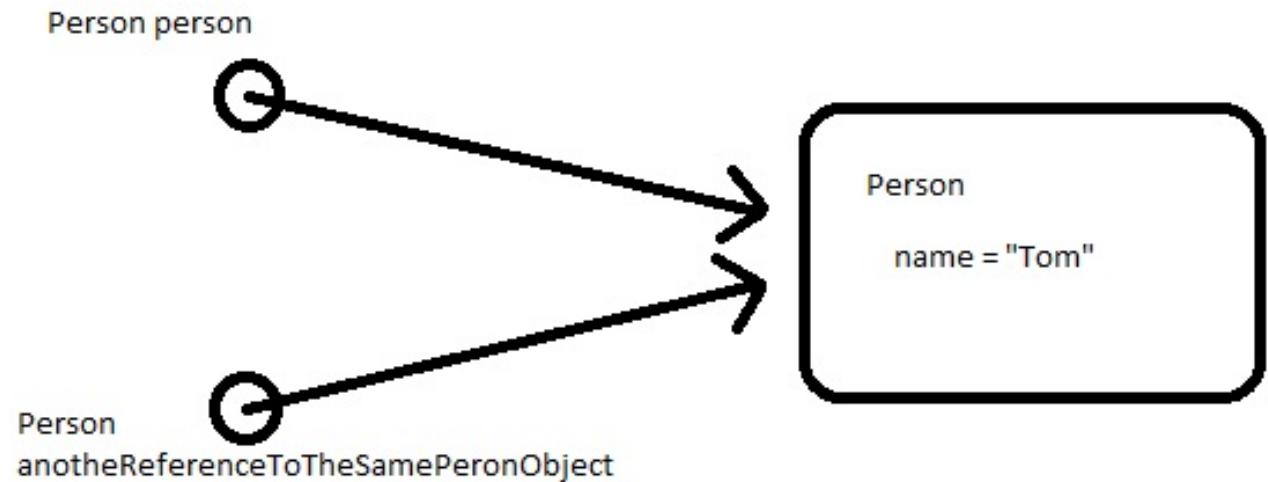
: Stack과 Heap 영역



Reference Type

: 참조 변수의 ==, !=

- ▶ 참조 변수의 ==, != 는 값이 같은지 비교하는 것이 아니라 동일한 객체를 참조하는지 아닌지를 확인하기 위해 사용
 - ▶ refVal1 == refVal2 : refVal1과 refVal2가 같은 객체를 참조하는가 확인
 - ▶ refVal1 != refVal2 : refVal1과 refVal2가 다른 객체를 참조하는가 확인
- ▶ null : 참조하는 객체가 없음을 나타냄
 - ▶ null인 참조 변수를 코드상에서 사용하고자 하면 NullPointerException 발생



Reference Type

: String

- ▶ 문자열을 저장하기 위해서는 String 타입으로 선언

- ▶ `String varName; // 선언`

- ▶ 리터럴: 소스코드상에 고정된 값

- ▶ `varName = "문자열"; // 문자열 할당. "" 사이에 문자열 리터럴 넣음`

- ▶ `String varName = "문자열"; // 선언과 할당을 동시에`

- ▶ `String varName = new String("문자열");`

Reference Type

: String 값 비교

- ▶ String 값이 같은지 비교하려면 .equals 메서드를 사용한다
 - ▶ strVal1.equals(strVal2); // strVal1과 strVal2가 같은 값을 가지고 있는가?

Reference Type

: String의 포맷 - printf

Examples

```
System.out.printf("Hello %s of %s%n", "World", "Java");
String s = String.format("The meaning of the %s is %d", "universe", 42);
```

Format	Description	Format	Description	Format	Description
%b	Boolean data	%c	Charactor	%d	Decimal
%e	Big Deciaml	%f	Float-point	%x	Hex-Decimal
%o	As Octal	%s	String	%n	New Line
%t	Date/time	%%	Percent		

문자열을 보다 풍부하고 편리하게 표현하기 위해 .format 메서드를 이용한다

열거형(enum)

- ▶ 몇 가지로 한정된 값만을 가지는 경우
- ▶ 예) 요일, 계절, 성별 등
- ▶ `public enum 열거타입명 { 열거 상수, ... }`
- ▶ 열거 상수는 모두 대문자로 작성(관례)
- ▶ 여러 단어로 구성될 경우 _로 연결(관례)
 - ▶ 예) RESULT_SUCCESS, RESULT_FAILED

The diagram shows a code snippet for an enum named 'DayOfWeek' with seven members: SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY. A purple bracket on the right side of the code groups the seven members together and is labeled '열거 상수' (enumeration constant). An arrow points from the text '열거 타입명' (enumeration type name) to the word 'enum' in the code. The entire code block is enclosed in a light gray box.

```
public enum DayOfWeek {  
    SUNDAY,  
    MONDAY,  
    TUESDAY,  
    WEDNESDAY,  
    THURSDAY,  
    FRIDAY,  
    SATURDAY;  
}
```

열거 객체의 메서드

Return Type	메서드	
String	name()	열거 타입의 문자열을 리턴
int	ordinal()	열거 객체의 순번을 리턴
int	compareTo(열거 객체)	열거 객체를 비교, 순번차를 리턴
열거 타입	valueOf(String name)	주어진 문자열의 열거 객체를 리턴
열거 타입	values()	모든 열거 객체들을 배열로 리턴

Reference Type

: Array (배열)

30명의 성적을 처리한다고 생각해보자
30개의 변수를 선언할 것인가?

▶ 배열이란?

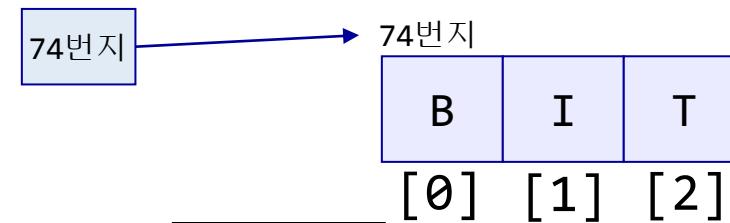
- ▶ 동일한 자료 유형의 여러 값들로 이루어진 객체(Object)
- ▶ new로 생성되는 참조 자료형
- ▶ 배열에 포함된 값들은 기본 자료형(Primitive Type)일 수도 있고 다른 객체를 참조하는 참조 자료형(Reference Type)일 수도 있음

```
char[] c = new char[3];
```

```
c[0] = 'B';  
c[1] = 'I';  
c[2] = 'T';
```

```
System.out.println(c[2]);  
c[2] = 'G';
```

```
char[] c
```



Reference Type

: Array (배열)의 선언

- ▶ 배열을 선언하는 두 가지 방법
 - ▶ 타입[] 변수;
 - ▶ 타입 변수[];

```
int[] intArray; // = int intArray[];  
double[] doubleArray; // = double doubleArray[];  
String[] stringArray; // = String stringArray[];
```

Reference Type

: Array (배열)의 초기화

- ▶ 값 목록이 있다면, {}으로 값의 목록을 지정하여 초기화할 수 있음
- ▶ 타입 배열명[] = { 값0, 값1, 값2, 값3, ...};

```
String[] days = {"월요일", "화요일", "수요일", "목요일", "금요일", "토요일", "일요일"};
```

- ▶ 이 방식은 배열 변수 선언과 동시에 해 주어야 허용된다.
- ▶ 다음과 같은 방식으로 new 연산자 뒤에 나열할 수도 있다.

```
String[] days;
```

```
days = new String[] {"월요일", "화요일", "수요일", "목요일", "금요일", "토요일", "일요일"};
```

Reference Type

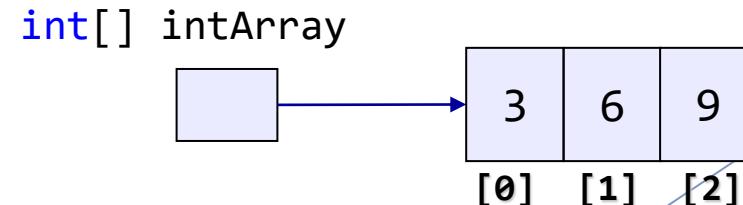
: Array (배열)의 초기화

- ▶ new 연산자로 배열 생성

```
int[] intArray = new int[3];  
intArray[0] = 3;  
intArray[1] = 6;  
intArray[2] = 9;
```

주의: 타입 선언시 배열 크기 지정 안됨

```
int[5] intArray = new int[]; // (x)
```

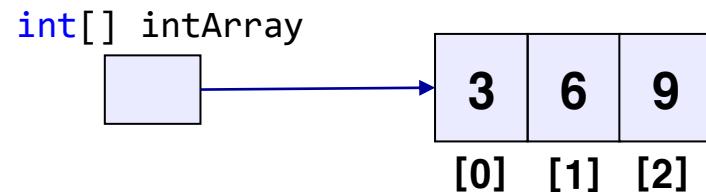


Reference Type

: Array (배열)의 사용

```
int[] intArray = new int[3]  
  
intArray[0] = 3;  
intArray[1] = 6;  
intArray[2] = 9;
```

```
System.out.println(intArray[0]);  
System.out.println(intArray[1]);  
System.out.println(intArray[2]);
```



```
for(int i=0; i<3; i++){  
    System.out.println(intArray[i]);  
}
```

```
for(int i=0; i<intArray.length; i++){  
    System.out.println(intArray[i]);  
}
```

배열의 `length` 멤버는 배열의 길이를 알아낼 때 사용할 수 있다

Reference Type

: Array (배열)의 선언 → 생성 → 초기화

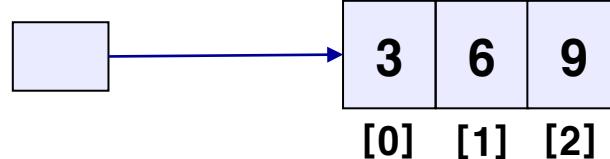
일반적인 인스턴스 배열의 선언과 생성

```
int[] intArray = new int[3];
```

배열의 초기화

```
intArray[0]=3;  
intArray[1]=6;  
intArray[2]=9;
```

int[] intArray



일반적인 인스턴스 배열의 선언과 생성

```
int[] intArray = new int[3];
```

생성과 동시에 초기화, 길이정보 생략가능

```
int[] intArray= new int[3] {1,2,3}; (X)  
int[] intArray= new int[] {1,2,3};
```

줄여서 표현가능

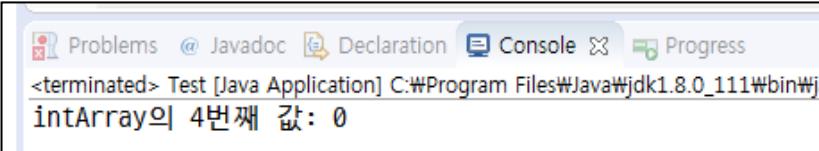
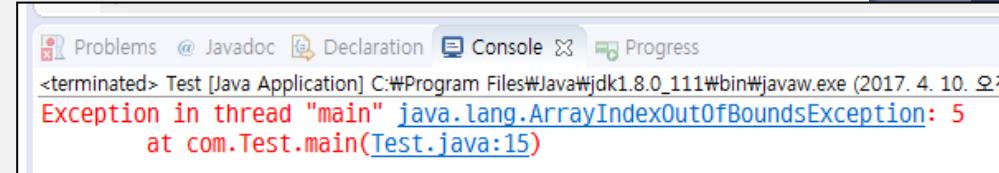
```
int[] intArray= {1,2,3};
```

Reference Type

: Array (배열) `ArrayIndexOutOfBoundsException`

- ▶ 배열의 범위를 벗어난 곳에 접근하면 `ArrayIndexOutOfBoundsException`을 일으킨다
- ▶ 배열의 인덱스는 0부터 시작한다는 점을 잊지 말자
 - ▶ 배열의 인덱스 범위는 0 ~ (`length - 1`)

```
int[] intArray;  
intArray = new int[5];  
  
intArray[0] = 3;  
intArray[1] = 6;  
intArray[2] = 9;  
  
int result = 0;  
for ( int i = 0; i <= intArray.length; i++ ){  
    result = result + intArray[i];  
}  
  
System.out.println("intArray의 4번째 값: " + intArray[3]);
```



```
intArray의 4번째 값: 0
```

Reference Type

: main 함수에서의 배열

배열 형태의 값 목록이 main 함수로 전달된다

```
package com.javaex.helloworld;

public class MainArgs {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println("args [" + i + "] : " + args[i]);
        }
    }
}
```

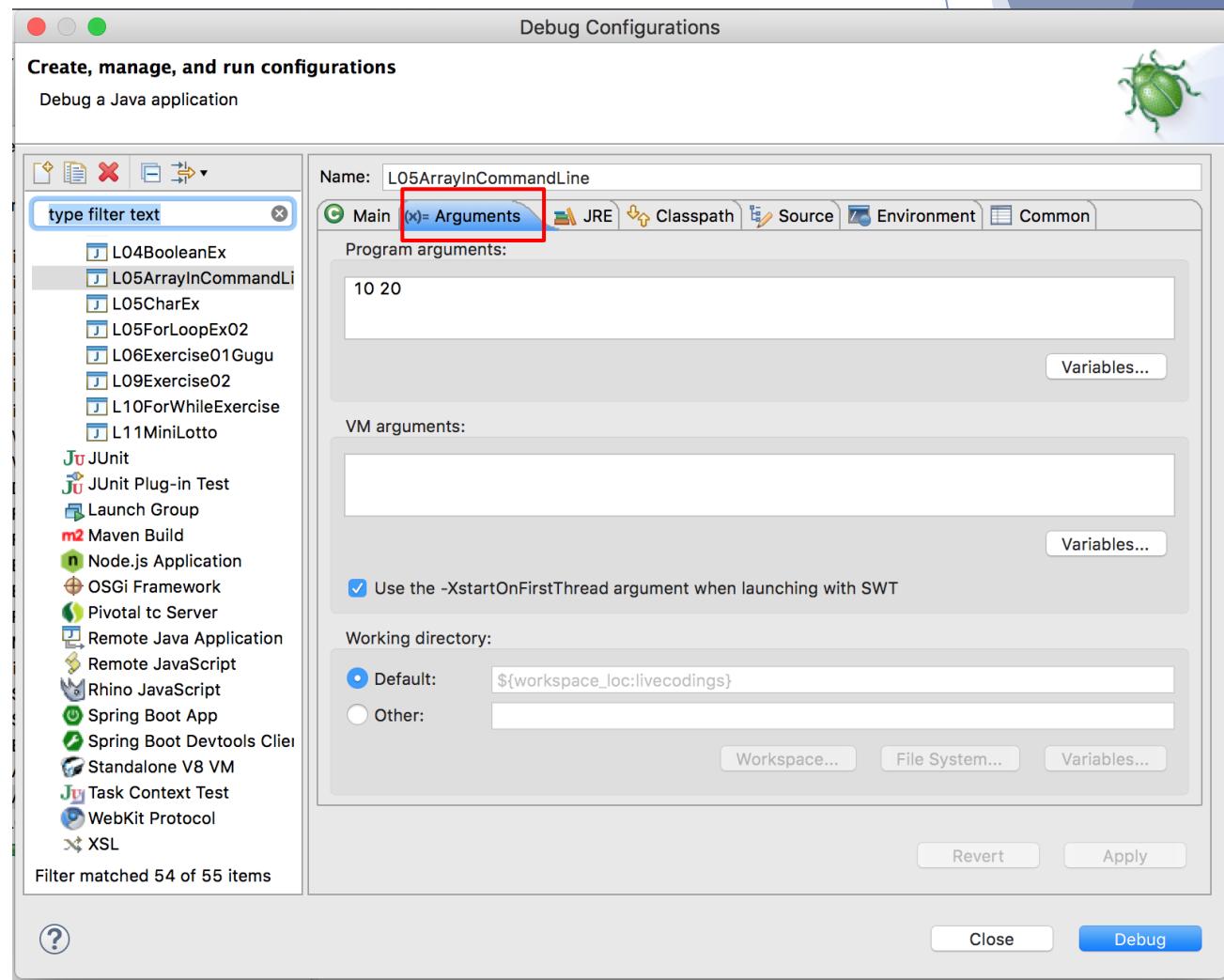
> java MainArgs Java Language is fun

args[0] = Java
args[1] = Language
args[2] = is
args[3] = fun

Reference Type

: Eclipse에서 main args 인수 전달

- ▶ Run > Run Configurations 에서 Arguments 탭을 클릭하면 main 함수로 넘겨줄 인자를 정의할 수 있다

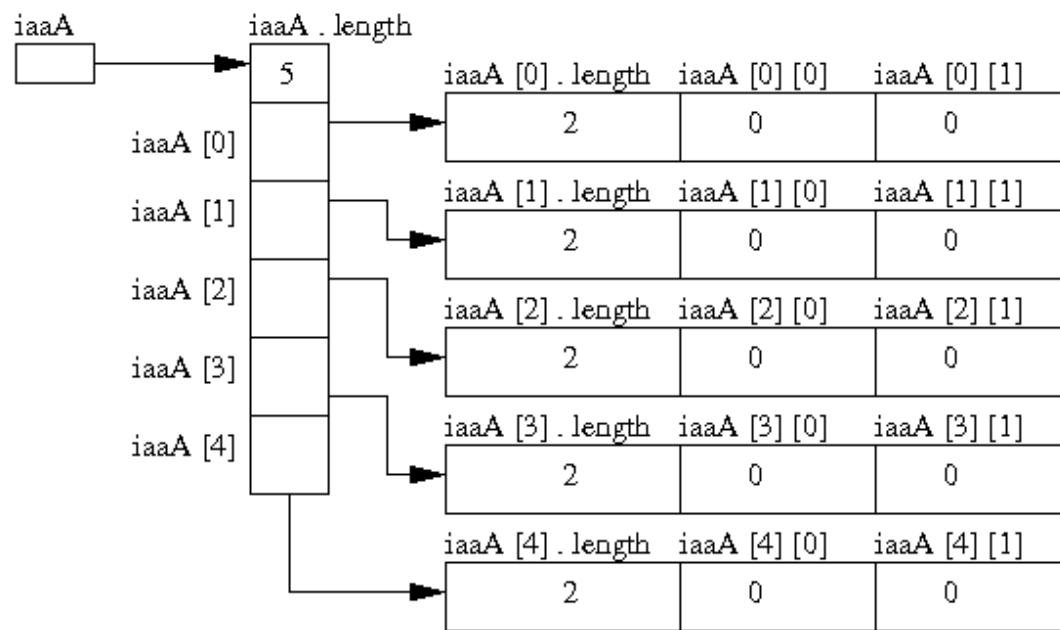


Reference Type

: 다차원 배열

```
int [][] iaaA = new int [5][2];
```

Layout of Memory



▶ 타입[][] 배열명 = new 타입[열_수][행_수];

```
int[][] twoDimens = int[5][2];
```

-> 5행 2열 배열 생성

3차원 이상 배열은 다루기 힘드니 조심해서 사용해야 한다

Reference Type

: 배열의 복사

- ▶ 배열은 한번 생성하면 크기를 변경할 수 없다
- ▶ 추가로 저장 공간이 필요하다면 원래 배열보다 큰 배열을 만들고 이전 배열의 항목값을 복사해야 한다

```
// For 문을 이용한 Copy

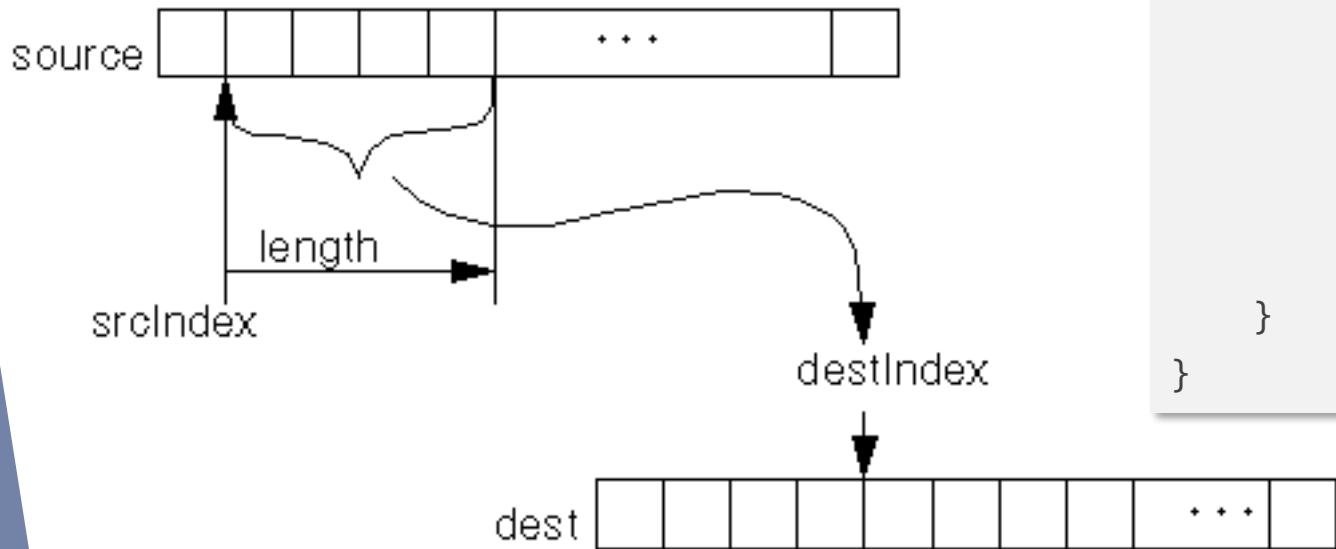
public class ArrayCopyByFor {
    public static void main(String[] args) {
        int[] intArray = { 1, 2, 3 };
        int[] newArray = new int[10];
        for (int i = 0; i < intArray.length; i++) {
            newArray[i] = intArray[i];
        }
        for (int i = 0; i < newArray.length; i++) {
            System.out.print(newArray[i] + " ");
        }
        System.out.println();
    }
}
```

Reference Type

: 배열의 복사

- ▶ System.arraycopy()로 배열을 복사할 수도 있다

- ▶ System.arraycopy(원본배열, 시작인덱스
타겟배열, 시작인덱스, 복사할 길이)



```
// System.arraycopy()를 이용한 배열 복사
public class ArrayCopyByArrayCopy {
    public static void main(String[] args) {
        int[] intArray = { 1, 2, 3 };
        int[] newArray = new int[10];

        System.arraycopy(intArray, 0,
                        newArray, 0,
                        intArray.length);

        for (int i = 0; i < newArray.length; i++) {
            System.out.print(newArray[i] + " ");
        }
        System.out.println();
    }
}
```

Enhanced For Loop

- ▶ 객체를 좀 더 쉽게 처리할 수 있는 향상된 for 문을 제공(Java 5 이상)
- ▶ 카운터 변수, 증감식을 사용하지 않고, 배열 및 컬렉션 항목의 개수만을 반복

```
public class EnhancedFor {  
    public static void main(String[] args) {  
        char vowels[] = { 'a', 'e', 'i', 'o', 'u' };  
        for (char vowel: vowels) {  
            System.out.print(vowel + " ");  
        }  
        System.out.println();  
    }  
}
```

Java 객체지향 프로그래밍

객체지향 이해하기

객체지향 기본개념

: 도서대여점 프로그램의 예



- ▶ 기능이 많아지고 복잡해진다
- ▶ 현실 세계를 시뮬레이션 할 필요의 대두
- ▶ 새로운 개발 방법이 필요해짐



- ✓ 책
- ✓ 책장
- ✓ 고객
- ✓ 고객장부
- ✓ 금고
- ✓ 대여장부
- ✓ ...

} 현실세계의 객체들

✓ **객체지향 프로그래밍**

(OOP: Object Oriented Programming)

객체지향 기본 개념

: 도서대여점 객체지향 설계

- ✓ 책
- ✓ 책장
- ✓ 고객
- ✓ 고객장부
- ✓ 금고
- ✓ 대여장부
- ✓ ...

- 책을 의인화 해봅시다.
- 책이 살아 있다면 책에게 다음과 같은 질문을 할 수 있을 것입니다.
 - 책 제목은 어떻게 되니?
 - 저자는 어떻게 되니?
 - 출판사는 어떻게 되니?
 - 가격은 어떻게 되니?
 - ...

객체지향 기본 개념

: 도서대여점 객체지향 설계

- ✓ 책
- ✓ 책장
- ✓ 고객
- ✓ 고객장부
- ✓ 금고
- ✓ 대여장부
- ✓ ...

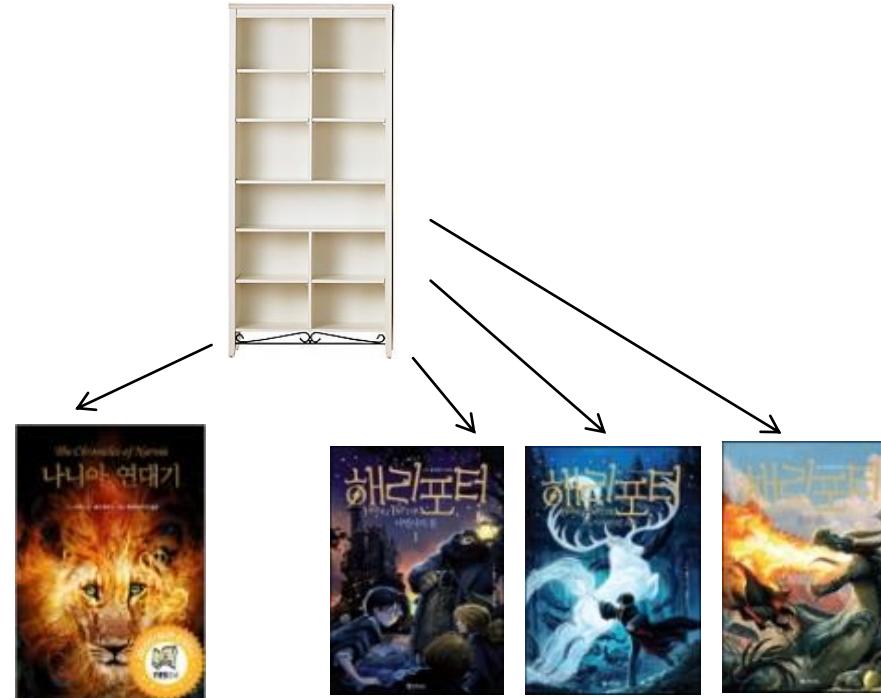
- 책장도 의인화 해봅시다.
- 책장이 살아 있다면 책장에게 다음과 같은 질문을 할 수 있을 것입니다.
 - 가지고 있는 책은 모두 몇권이니?
 - “해리포터”라는 소설을 가지고 있니?
 - C. S. 루이스가 쓴 판타지 소설은 뭐지?
- 책을 가지고 있는 것은 책장입니다.
- 의인화 해보면 책장이 책을 관리하고 있습니다.

객체지향 기본 개념

: 도서대여점 객체지향 설계

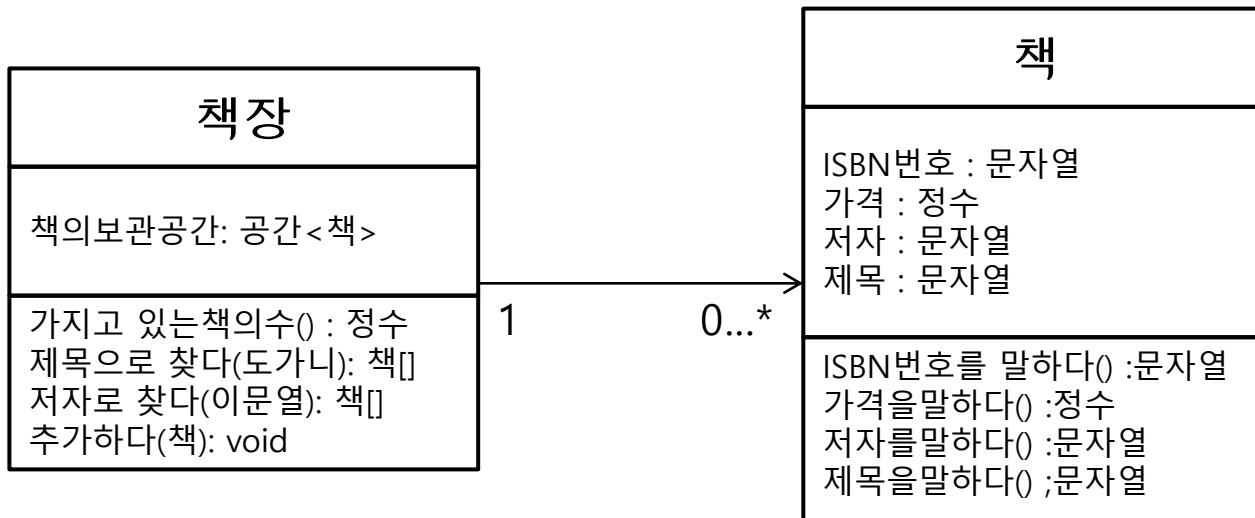
- ✓ 책
- ✓ 책장
- ✓ 고객
- ✓ 고객장부
- ✓ 금고
- ✓ 대여장부
- ✓ ...

• 책과 책장을 도식화 해보면



객체지향 기본 개념

: 도서대여점 객체지향 설계



▶ 관계를 표현해 보면

- ▶ 책장은 책을 저장하는 공간을 가지고 있다(속성)
- ▶ 책은 ISBN 번호, 가격, 저자, 제목을 가지고 있다 (속성)
- ▶ 책장은 자신이 가지고 있는 속성을 이용하는 기능을 가지고 있다
- ▶ 책장은 책을 보유할 수 있다 (관계)

객체지향 기본 개념

3. 개념의 확인

- ✓ 책
 - ✓ 책장
 - ✓ 고객
 - ✓ 고객장부
 - ✓ 금고
 - ✓ 대여장부
 - ✓ ...

• 고객과 고객장부의 관계를 그려보기

객체지향 기본 개념

: 도서대여점 객체지향 설계

- ▶ 도서대여점 고객 vs 신발가게 고객
 - ▶ 도서 대여점의 고객에게 신발 사이즈를 물어본다면?
 - ▶ 도서 대여점의 고객에게 몸무게를 물어본다면?
 - ▶ 신발 가게 고객에게 신발 사이즈를 물어본다면?

- 도서대여점 고객은 이름과 연락처가 중요
- 신발가게는 신발사이즈가 중요

중요한 것은 남기고, 불필요한 것을 없애는 것을 **객체지향**에서
“**추상화**”라고 말한다.

객체지향 기본 개념

: 도서대여점 객체지향 설계

- ▶ 모든 것이 중요한 것은 아니다

- ▶ 도서대여점의 책장은 색이 중요하지 않지만, 가구점에서는 색이 중요하다
- ▶ 어느 관점에서 보느냐에 따라 중요한 속성도 있고, 그렇지 않은 것도 있다.

- 객체지향 프로그래밍은 관점에 따라 객체가 가지는 속성과 기능을 다르게 표현하게 된다.

- ▶ 관점을 어떻게 잡고 어떻게 설계하느냐에 따라 재사용이 어려울 수도 있다

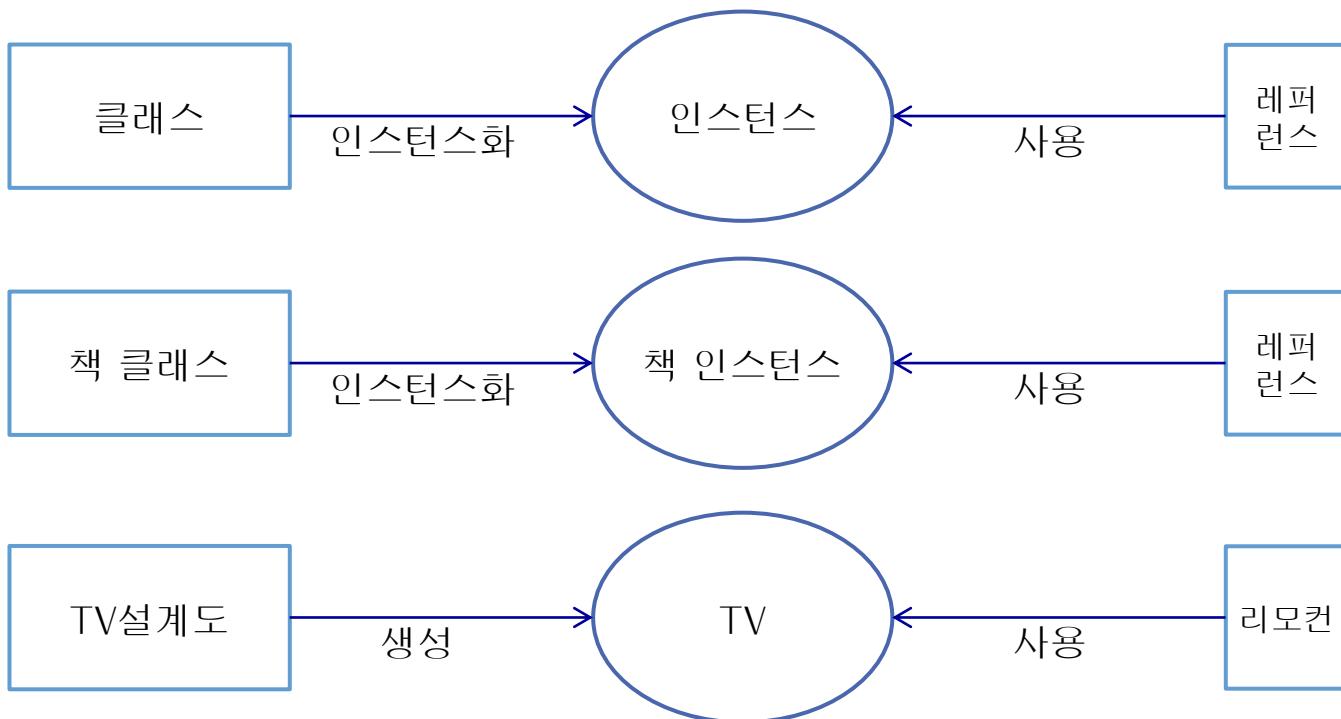
객체 지향 기본 개념

: 객체 지향이 추구하는 것

- ▶ 현실세계를 그대로 옮긴다
 - ▶ 객체들이 가지고 있는 속성과 기능 중 필요한 것만 남기고 불필요한 것은 제거한다
 - ▶ 추상화한다
- ▶ 재사용한다
 - ▶ 관점을 잘 파악하고 그 요구사항에 맞게 설계하면 재사용이 가능해진다
 - ▶ 설계도를 그려두면 양산이 가능해진다

객체 지향 기본 개념

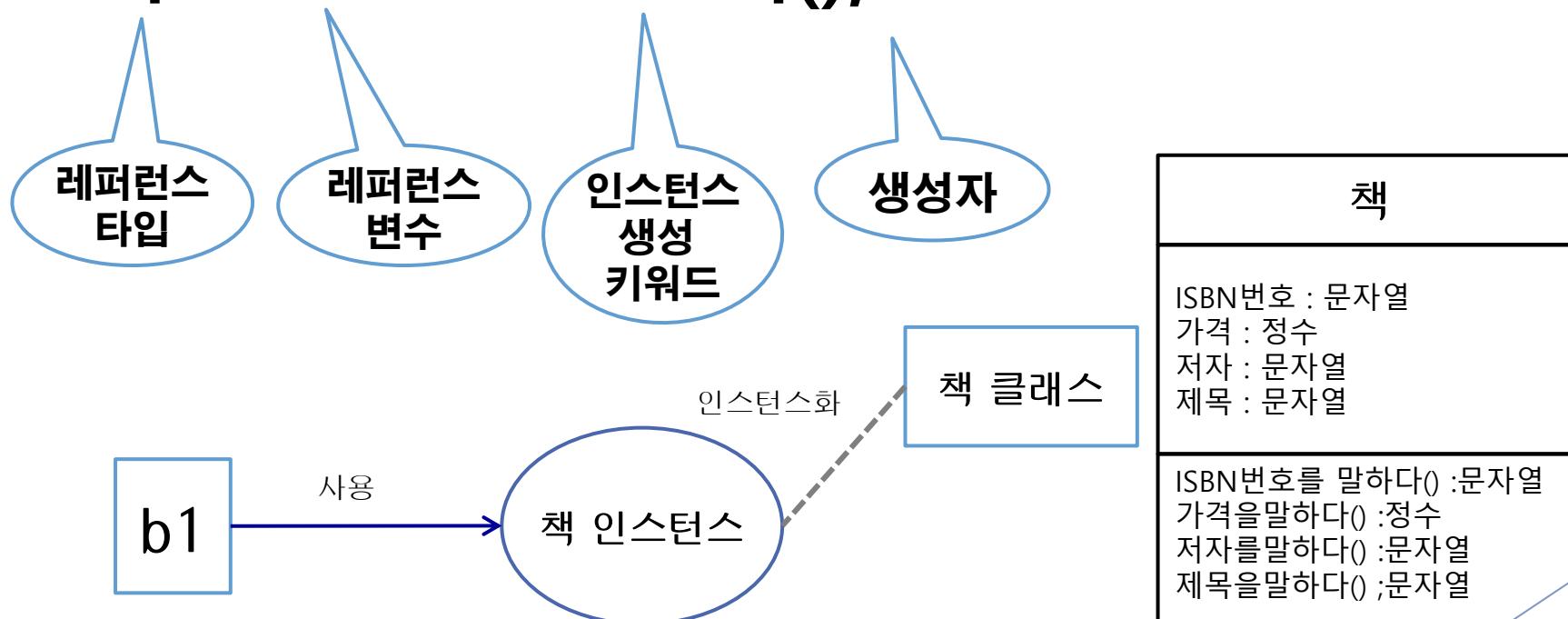
: 클래스, 인스턴스, 레퍼런스



객체 지향 기본 개념

: Java 문법으로 이해하기

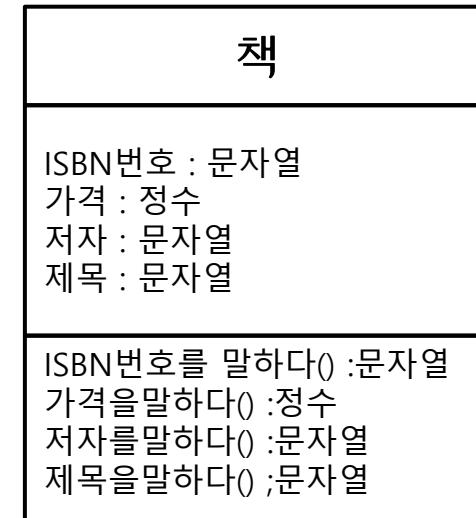
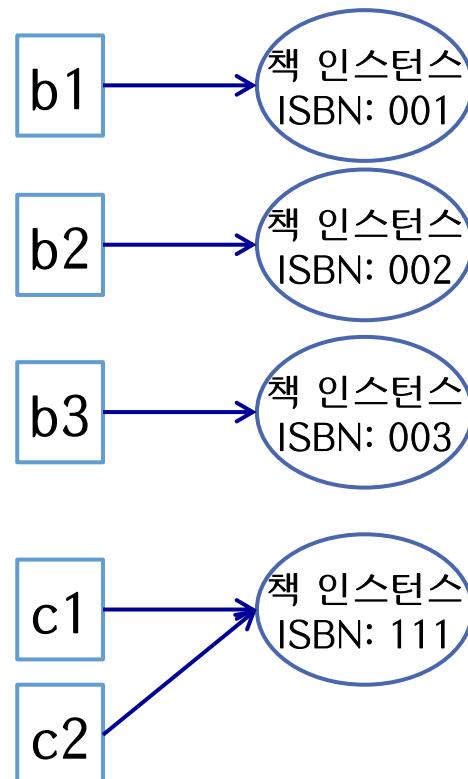
책 b1 = new 책();



객체 지향 기본 개념

: Java 문법으로 이해하기

```
public static void main(){  
    책 b1 = new 책(001);  
    책 b2 = new 책(002);  
    책 b3 = new 책(003);  
  
    책 c1 = new 책(111);  
    책 c2 = c1;  
  
    c1= null;  
}
```



객체 지향 기본 개념

: 자동차 중 버스, 스포츠카, 포크레인을 정의해 봅시다

- ▶ 버스, 스포츠카, 포크레인의 공통점을 추출하여 자동차를 구상해 봅시다.

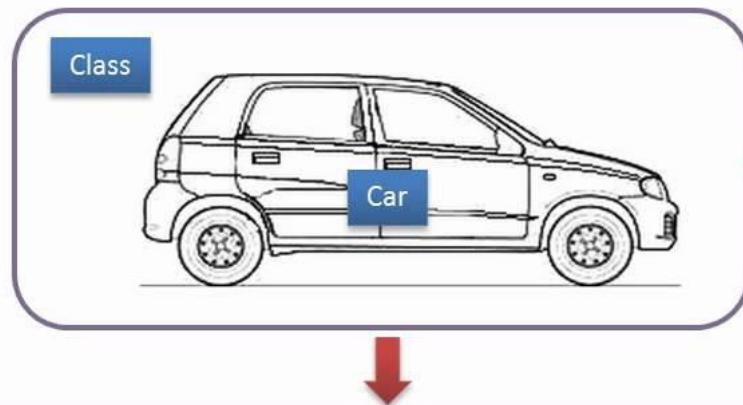
버스	스포츠카	포크레인
달리다():void 뒷문열다():void 안내방송하다():void	달리다():void 지붕을열다():void	달리다(): void 기계삽사용하다():void

객체 지향 기본 개념

: 자동차의 예

What is a Class?

A class is the blueprint from which individual objects are created.



```
1 public class Car
2 {
3     private String brand = null;
4     private String model = null;
5     private String color = null;
6
7     public String getBrand()
8     {
9         return brand;
10    }
11
12    public void setBrand(String brand)
13    {
14        this.brand = brand;
15    }
16
17    public String getModel()
18    {
19        return model;
20    }
21
22    public void setModel(String model)
23    {
24        this.model = model;
25    }
26
27    public String getColor()
28    {
29        return color;
30    }
31
32    public void setColor(String color)
33    {
34        this.color = color;
35    }
36
37 }
```

Objects

```
Car maruthiAlto10 = new Car();
maruthiAlto10.setBrand("Maruthi Alto");
maruthiAlto10.setModel("K10");
maruthiAlto10.setColor("Orange");
```

brand = Maruthi Alto
model = K10
color = Orange



```
Car swift = new Car();
swift.setBrand("Swift");
swift.setModel("ZDI");
swift.setColor("Red");
```

brand = Swift
model = ZDI
color = Red

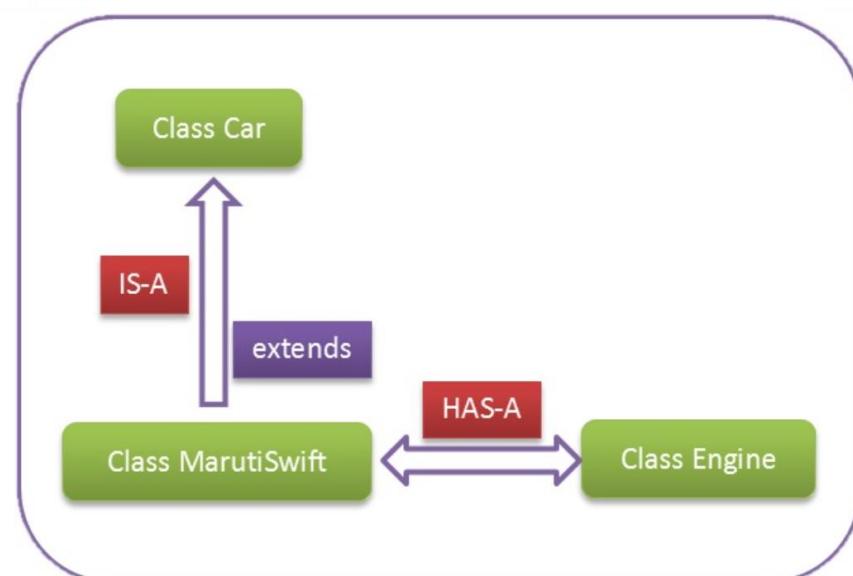
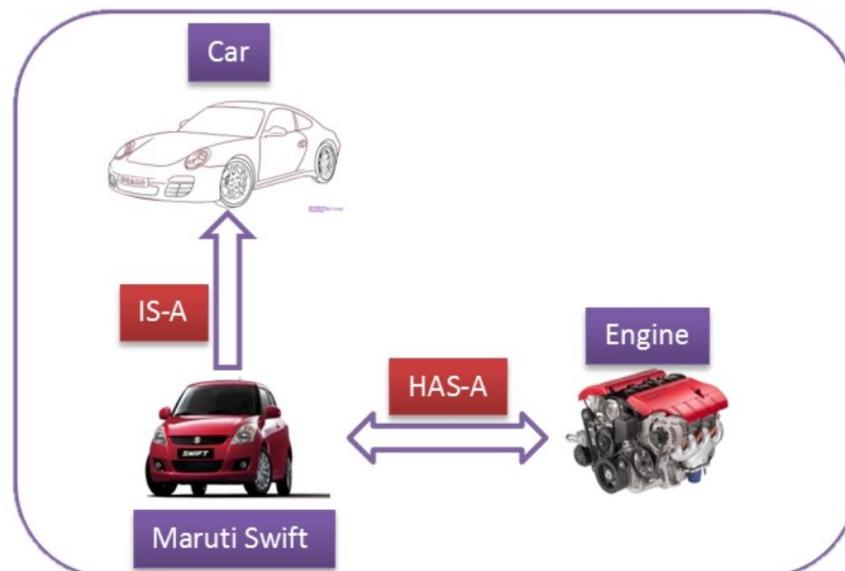
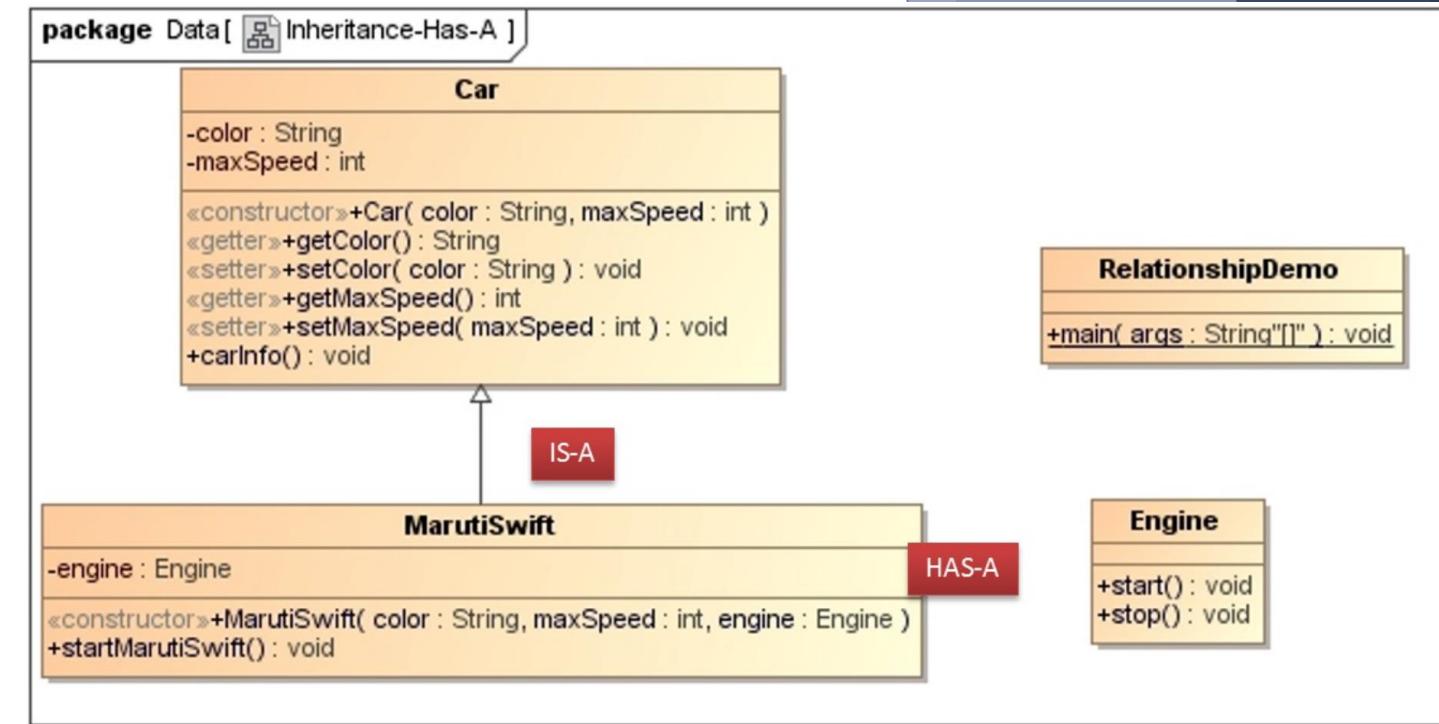


```
Car maruthiAlto800 = new Car();
maruthiAlto800.setBrand("Maruthi Alto");
maruthiAlto800.setModel("800");
maruthiAlto800.setColor("Blue");
```

brand = Maruthi Alto
model = 800
color = Blue



객체지향 기본 개념 : 자동차의 예

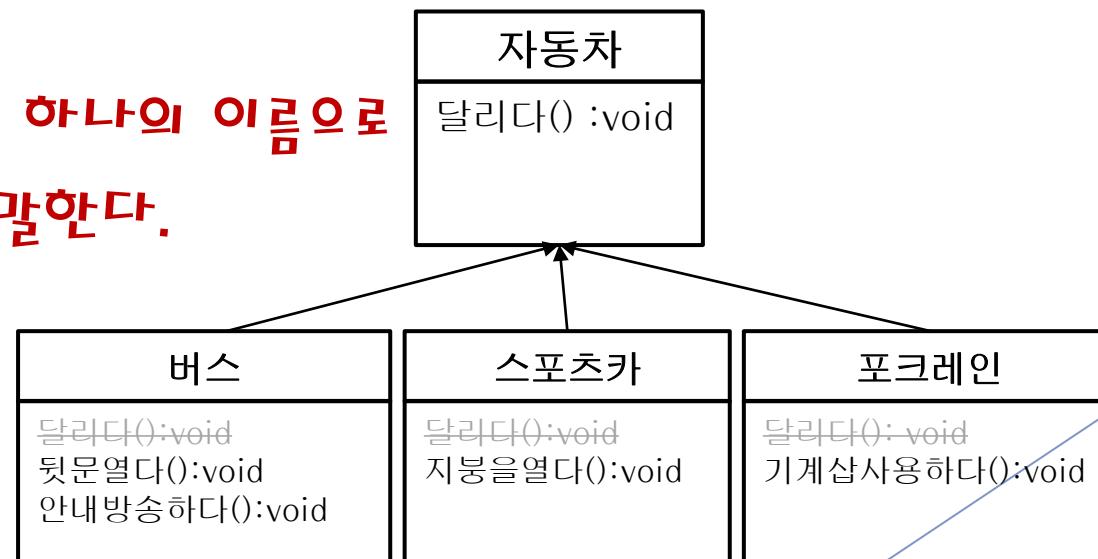


객체 지향 기본 개념

: 주차장 프로그램의 설계 예시

- ▶ 버스, 스포츠카, 포크레인은 자식 클래스
- ▶ 자동차는 부모 클래스
- ▶ 버스, 스포츠카, 포크레인을 일반화 한 것은 자동차
- ▶ 자동차를 상속(확장)한 것은 버스, 스포츠카, 포크레인

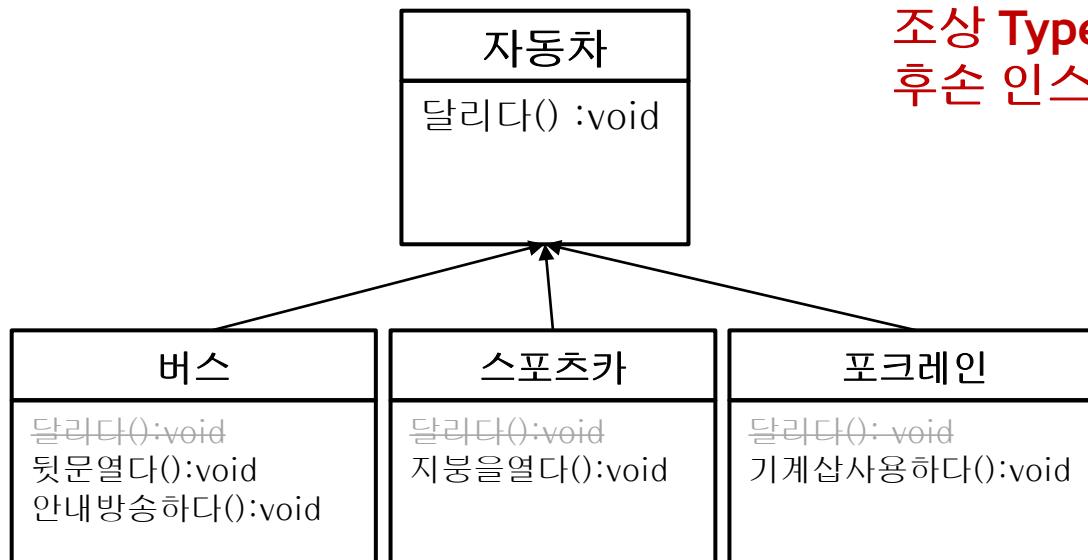
공통점이 있는 여러가지 물체들을 하나의 이름으로
부르는것을 “일반화” 라고 말한다.



객체 지향 기본 개념

: 주차장 프로그램의 설계 예시

- ▶ 버스는 자동차다
- ▶ 스포츠카는 자동차다
- ▶ 포크레인은 자동차다



자동차 c1 = new 버스();
자동차 c2 = new 스포츠카();
자동차 c3 = new 포크레인();

버스 bus = new 버스();
스포츠카 sportCar = new 스포츠카();
자동차 poclarein = new 포크레인();

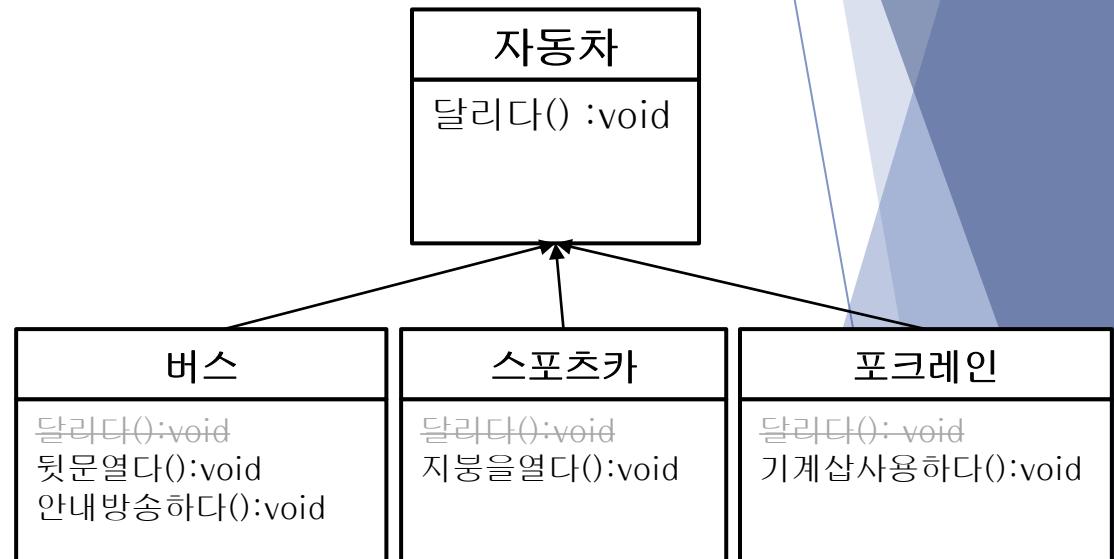
조상 Type의 레퍼런스 변수는
후손 인스턴스를 레퍼런스 할 수 있다.

객체 지향 기본 개념

: 주차장 프로그램의 설계 예시

```
버스 bus1 = new 버스();
bus1.달리다();
bus1.뒷문열다();
bus1.안내방송하다();
```

- ▶ 버스 주차를 요청하니 안내방송을 실행
- ▶ 스포츠카 주차를 요청하니 지붕을 열고 닫음
- ▶ 포크레인 주차를 요청하니 땅을 파고 있음

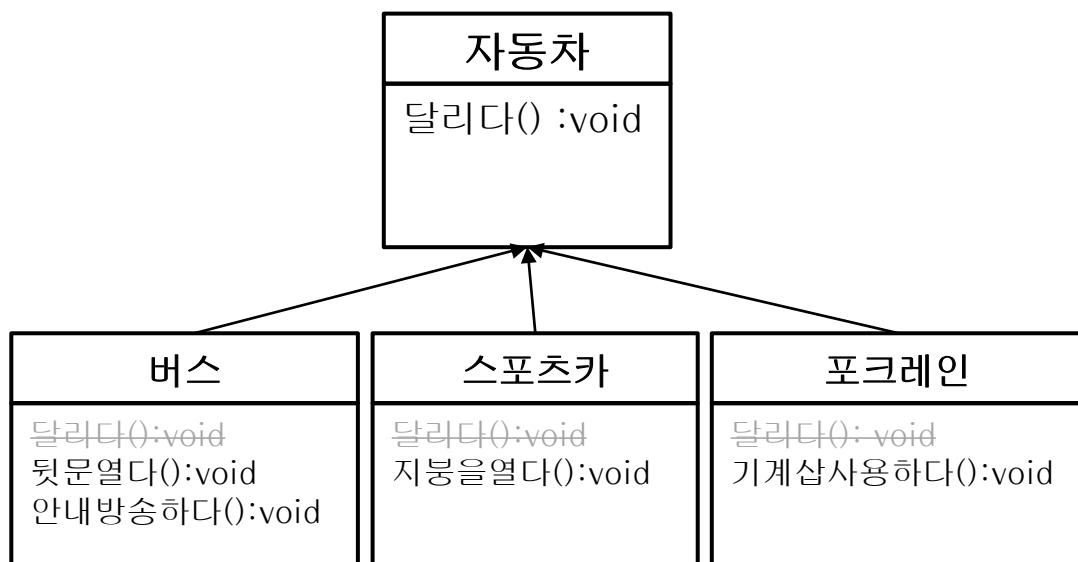


자동차의 주인은 자동차의 기본 기능만 이용하여 주차하기를 바란다.

객체 지향 기본 개념

: 주차장 프로그램의 설계 예시

- ▶ 레퍼런스 타입 클래스에 설계되어 있는 기능만 사용할 수 있다.



버스와 그 조상 설계도에
있는 기능만 사용 가능

버스 bus1 = new 버스();
bus1.달리다();
bus1.뒷문열다();
bus1.안내방송하다();

자동차와 그 조상 설계도에
있는 기능만 사용 가능

자동차 bus2 = new 차();
bus2.달리다();
bus2.뒷문열다(); (X)
bus2.안내방송하다(); (X)

설계도에 구현된 부분이외의 기능은 사용할 수 없다.

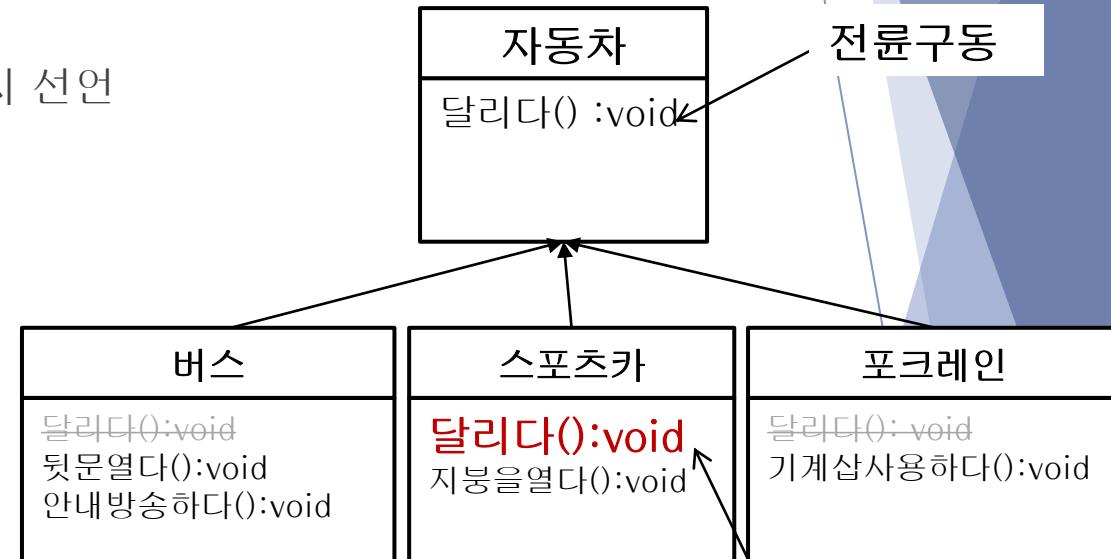
객체 지향 기본 개념

: 주차장 프로그램의 설계 예시

- ▶ 스포츠카는 자동차가 가지고 있는 본래의 "달리다" 기능을 자신만의 구동방식으로 바꾸고자 합니다.
 - ▶ 부모의 기능과 완전히 같은 모양으로 내용을 다시 선언
-> 메서드 오버라이드

```
자동차 bus = new 버스();  
bus.달리다();  
bus.뒷문열다(); (X)  
bus.안내방송하다(); (X)
```

```
자동차 spoCar = new 스포츠카();  
spoCar.달리다(); <후륜구동  
spoCar.뒷문열다(); (X)  
spoCar.안내방송하다(); (X)
```



메소드 오버라이드(Override: 덮어쓰다)
<- 오버로드(Overload) 와 자주 혼동되니 주의

객체 지향 기본 개념

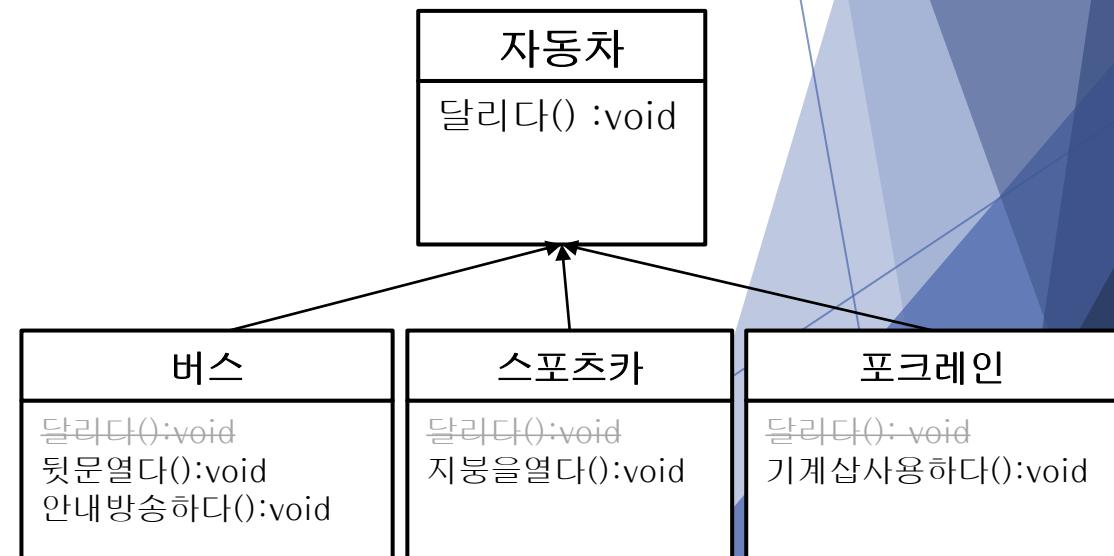
: 주차장 프로그램의 설계 예시

- ▶ 주차 공간의 확보

```
자동차 bus1= new 버스();  
자동차 spo1= new 스포츠카();  
자동차 poc1= new 포크레인();
```

```
자동차[] carArray = new 자동차[15]  
carArray[0] = bus1;  
carArray[1] = spo1;  
carArray[2] = fork1;
```

```
버스[] busArray = new 버스[5]  
스포츠카[] spoArray = new 스포츠카[5]  
포크레인[] poclArray = new 포크레인[5]
```



Java Programming

Object Oriented Programming

Object Oriented Programming

: 객체지향 프로그래밍이란?

- ▶ 현대 컴퓨터 프로그래밍 패러다임의 하나
- ▶ 컴퓨터 프로그램을 여러 개의 독립된 단위(객체)들의 모임으로 파악
- ▶ 특징
 - ▶ 객체는 데이터와 그 데이터를 처리할 수 있는 메서드를 갖는다
 - ▶ 소프트웨어의 부품화, 재사용을 주요 목표로 함
 - ▶ 코드의 재사용성이 높다
 - ▶ 코드의 관리가 쉬워짐 : 개발과 유지보수에 강점, 직관적 코드 분석을 가능하게 함
 - ▶ 신뢰도 높은 프로그램 개발을 가능하게 함

Object Oriented Programming

: 객체(Object)와 클래스(Class)

- ▶ 객체의 정의
 - ▶ 실세계에 존재하는 것, 사물 또는 개념
 - ▶ 객체는 정보를 효율적으로 관리하기 위해 의미를 부여하고 분류하는 논리적 단위
 - ▶ 객체는 다른 객체와 서로 상호작용하면서 동작한다
- ▶ 객체의 구성 요소
 - ▶ 객체는 속성(정보)과 동작(기능)의 집합 -> 객체의 멤버(member: 구성요소)라 함
 - ▶ 속성은 필드(field), 동작은 메서드(method)로 정의
- ▶ 클래스
 - ▶ 객체를 정의해 놓은 것
 - ▶ 객체를 생성하는데 사용되는 설계도

클래스	객체
제품 설계도	제품
TV설계도	TV
붕어빵기계	붕어빵

Object Oriented Programming

: 인스턴스 (Instance)

- ▶ 객체 (Object)는 인스턴스 (Instance)의 일반적 의미
- ▶ 객체가 메모리에 할당되어 실제 사용될 때 인스턴스라고 부른다
- ▶ 인스턴스 (instance): 객체가 메모리에 할당되어 실제 사용될 때
- ▶ 인스턴스화 (instantiate): 클래스로부터 인스턴스를 생성하는 것



Object Oriented Programming

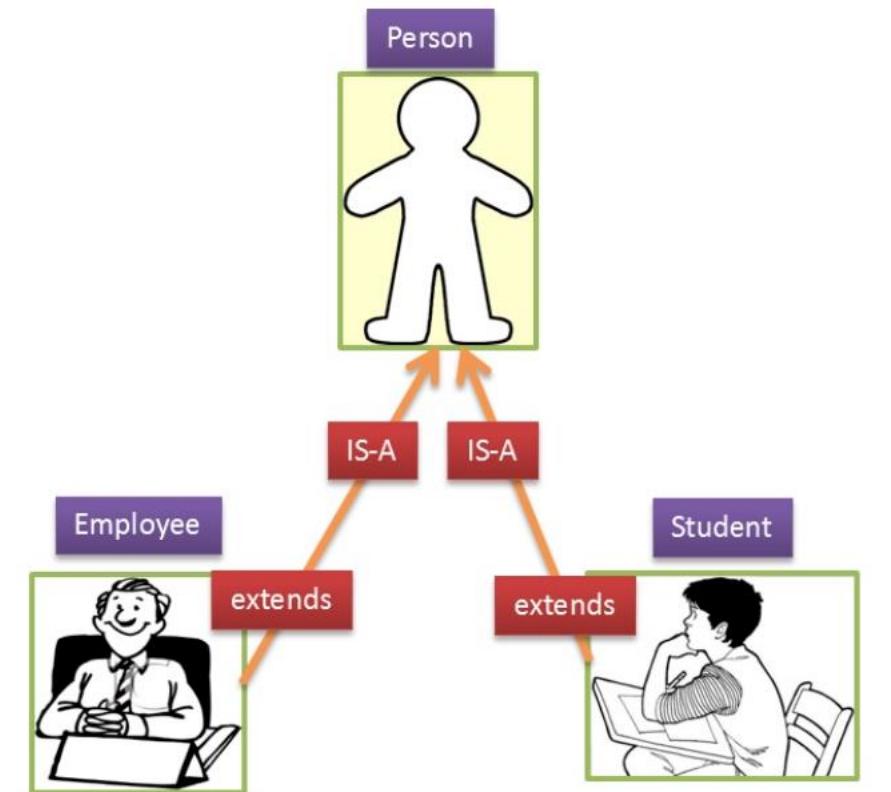
: 객체지향 네 가지 특성

- ▶ 상속성 (Inheritance)
- ▶ 캡슐화 (Encapsulation)
- ▶ 다형성 (Polymorphism)
- ▶ 추상화 (Abstraction)

Object Oriented Programming

: 상속 (Inheritance)

- ▶ 이미 만든 객체와 비슷하지만 필드와 메서드가 약간 차이가 나는 객체를 생성
 - ▶ 기존의 클래스에서 공통된 필드와 메서드를 상속(재사용)
 - ▶ 더 필요한 필드와 메서드를 추가
- ▶ 코드를 간결하게 하고 코드의 재사용성을 높임



Object Oriented Programming

: 캡슐화 (Encapsulation)

- ▶ 객체의 실제 구현된 내용을 감추고 접근 방법만 노출하는 것
- ▶ 외부 객체(객체를 사용하는 쪽)에서는 객체의 내부 구조를 알지 못하며
- ▶ 객체가 노출하여 제공하는 필드와 메서드만 이용할 수 있음
- ▶ 객체 작성시 개발자는 숨겨야하는 필드와 메서드, 공개하는 필드와 메서드를 구분하여 작성한다

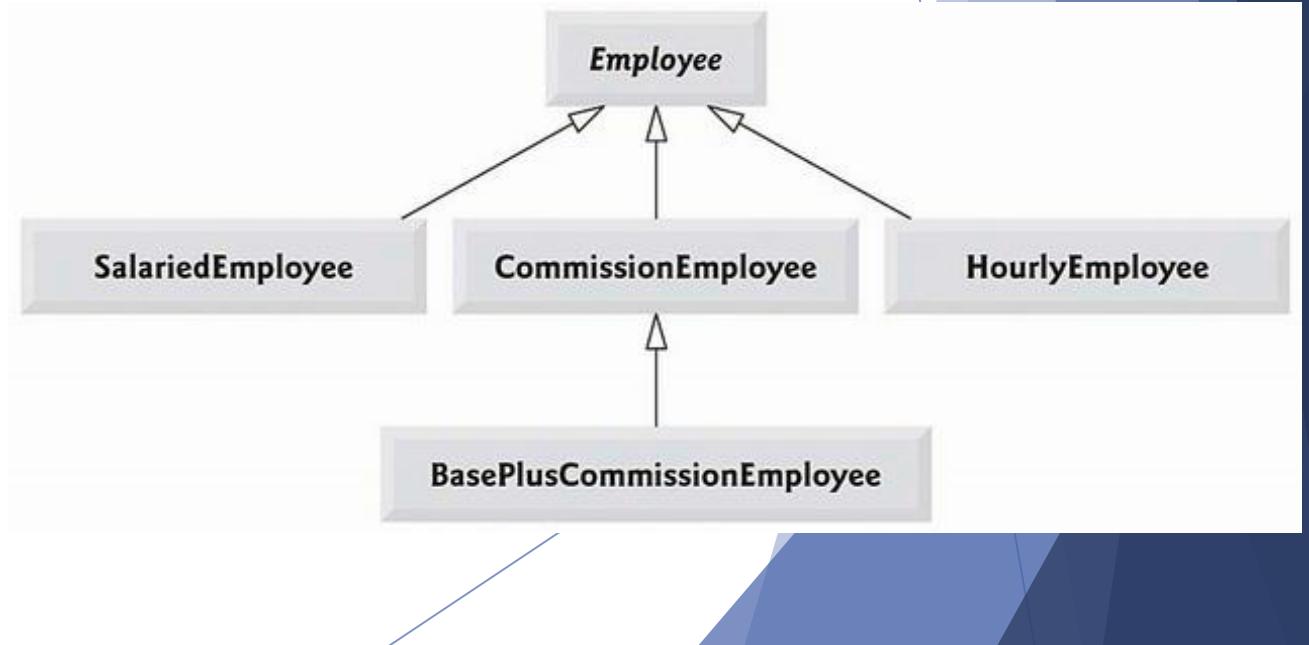
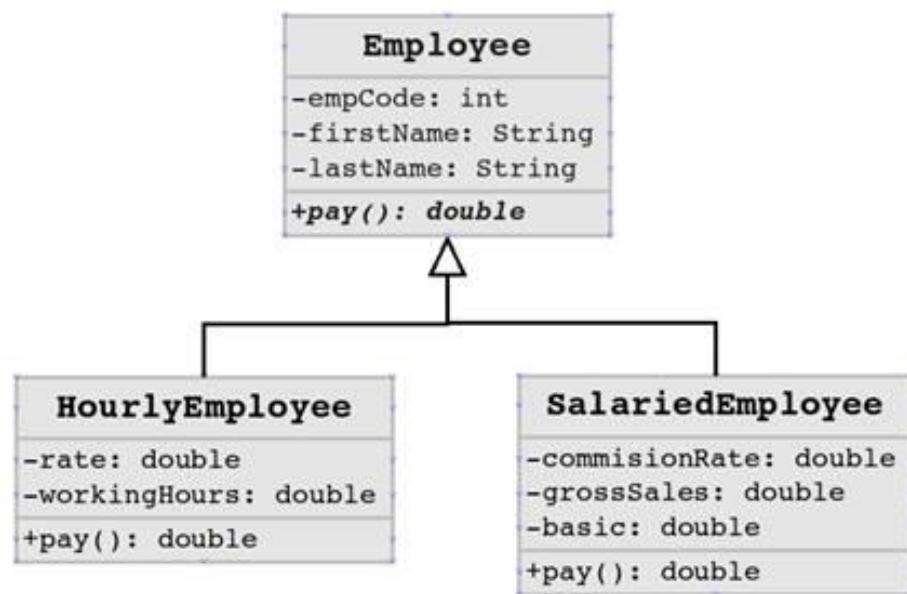
- ▶ 캡슐화하는 이유는 외부의 잘못된 사용으로 객체가 손상되는 것을 피하기 위함
- ▶ 접근 제한자(Access Modifier)를 사용하여 객체의 필드와 메서드의 사용 범위를 제한한다

정보 은닉

Object Oriented Programming

: 다형성 (Polymorphism)

- ▶ Java 객체지향에서 가장 중요한 개념
- ▶ 하나의 메서드나 클래스를 다양한 구현으로 사용 가능하게 하는 개념
- ▶ 오버로드(Overload)와 오버라이드(Override)를 통해 다형성을 구현

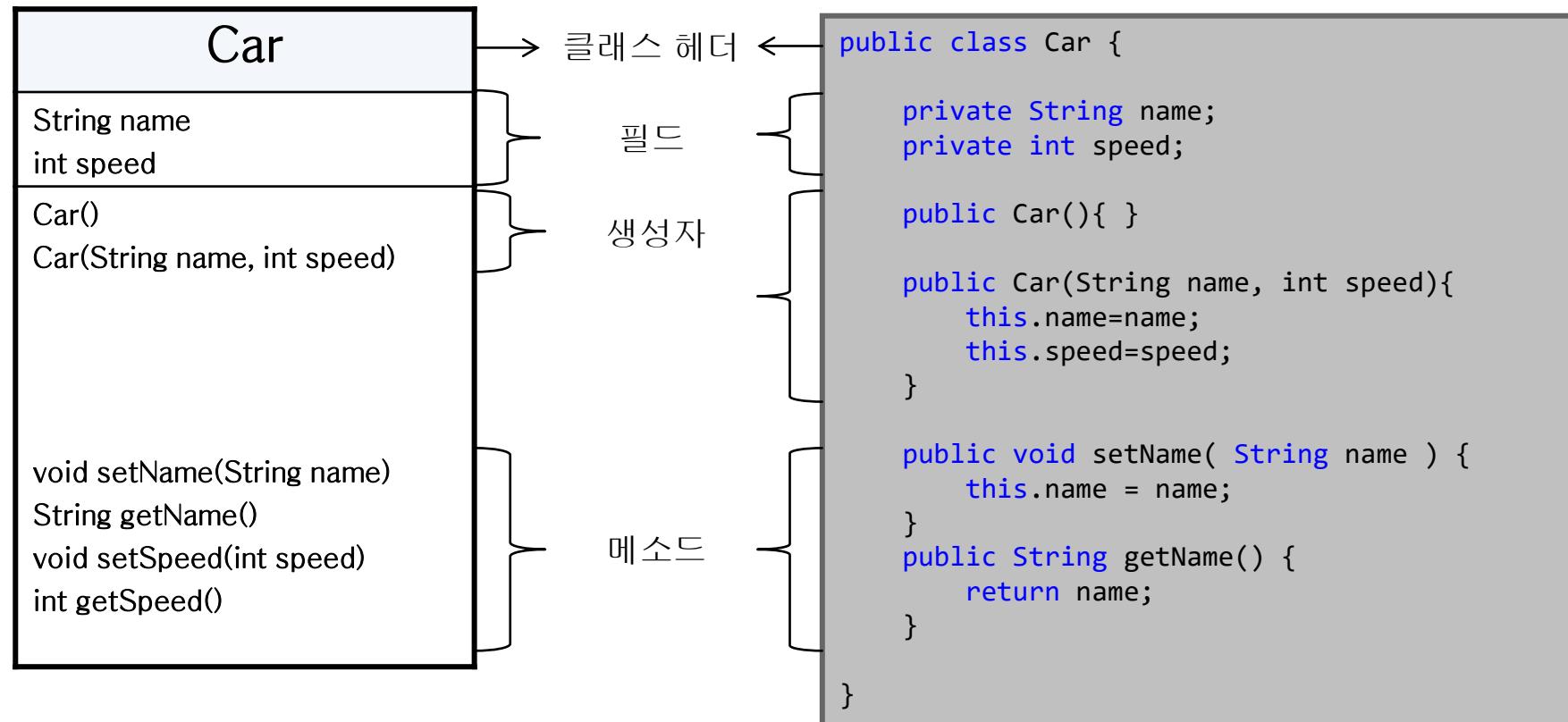


Java Programming

Java Class

Java Class

: 클래스의 구조



Java Class

: 필드 (field)

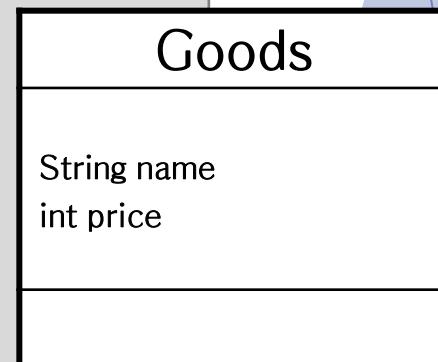
- ▶ 객체의 데이터, 상태를 저장하는 변수
- ▶ 주로 기본 타입 또는 참조 타입으로 정의
- ▶ 멤버 변수라고도 함

[문제]

쇼핑몰에서 상품을 관리하기 위해 상품관리 프로그램을 만들려고 합니다. 프로그램을 만들기 전에 업무(비즈니스) 분석을 통해 상품 객체를 분석하고 다음과 같은 Goods클래스를 정의 하였습니다.

Goods 클래스를 정의하고 GoodsApp 클래스에서 Goods 클래스를 테스트 하세요.

- 1) Goods 객체를 하나 생성하고 이 객체에 대한 레퍼런스 변수를 camera로 합니다.
- 2) 이 객체의 데이터인 각 각의 인스턴스 변수는 다음과 같은 값을 가지도록 합니다.
상품이름 : "nikon", 가격: 400000
- 3) 값을 세팅 한 후, 객체의 데이터를 출력해 보세요.



연습문제

: 클래스의 정의와 사용

[문제]

다음의 데이터를 추가한 후, 출력해 보세요

- 상품이름 : “LG그램”, 가격: 900000
- 상품이름 : “머그컵”, 가격: 2000

Goods
String name
int price

Java Class

: 접근자 (Access Modifier)

- ▶ 객체의 필드와 메서드에 접근을 제한하기 위해 사용
- ▶ 정보 은닉을 위한 방법 (캡슐화)
- ▶ 정보 접근 수준에 따라 public, protected, default, private 네 가지가 있다

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

public > protected > default > private

연습문제

: 접근자 연습

[문제]

- Goods 클래스의 필드 접근자를 public으로 변경해 봅니다.
- Goods 클래스의 필드 접근자를 default보다 강한 접근 제어자인 private로 지정하여 어떤 변화가 있는지 확인해 봅니다.

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

public > protected > default > private

Java Class

: 메서드 (method)

- ▶ 객체의 기능 또는 행동을 정의
- ▶ 정의 방법

```
1  2  3  4  
public int getSum( int i, int j ) {  
    int result = i + j; 5  
    return result; 6  
}
```

1. 접근 지정자
2. 리턴 타입
3. 메서드 이름
4. 메서드 인자 (파라미터)
5. 구현코드
6. 리턴문

- ▶ 호출 방법

```
1  2  3  4  5  
int sum = util.getSum(3, 2);
```

1. 자료형
2. 변수명
3. 레퍼런스변수
4. 메서드명
5. 메서드 인자 (파라미터)

Java Class

: 메서드 (method)

- ▶ 매개변수(parameter)
 - ▶ 메서드를 선언할 때 괄호 안에 표현된 Input 값을 나타내는 변수 (type1 name1, type2 name2, ...)
 - ▶ 메서드 호출에서 들어가는 구체적인 값은 인자(Argument)라고 함
- ▶ 반환타입(return type)
 - ▶ 메서드는 0개 혹은 1개의 값을 Output으로 반환할 수 있음
 - ▶ 반환 값이 없을 때: void
 - ▶ 반환 값이 있을 때: int, boolean, Goods, ...
 - ▶ 반환 되는 값은 메서드 선언에서 정의된 반환 타입과 일치해야 함
- ▶ 메서드 이름
 - ▶ 자바의 식별자 규칙 대/소문자, 숫자, _, \$ 조합하여 지을 수 있고 숫자로 시작할 수 없다.
 - ▶ 관례에 따라 소문자로 작성하고 두 단어가 조합될 경우 두 번째 시작문자는 대문자로 짓는다.
 - ▶ 메서드 명은 기능을 쉽게 알 수 있도록 작성하는 것이 좋다

Java Class

: 가변 인수

- ▶ 메소드의 매개 변수의 개수를 알 수 없을 때 사용
- ▶ 가장 간단한 해결방법은 매개변수를 배열로 선언하는 것

```
double sum(double[] values) { }
```

- ▶ 이러한 경우, 메소드 호출시 배열을 넘겨주어 여러 개의 값을 전달할 수 있다

```
double[] numbers= { 1, 2, 3, 4, 5 };  
double result = sum(numbers);  
double result = sum(new double[] { 1, 2, 3, 4, 5 });
```

- ▶ 방법 2: 매개 변수를 ... 를 이용하여 선언
 - ▶ 자동으로 배열이 생성되고 매개값으로 사용된다

```
double sum(double ... values) { }
```

```
double result = sum(1, 2, 3, 4, 5 );
```

Java Class

: Getter, Setter

- ▶ 일반적으로 객체의 데이터는 객체 외부에서 직접적으로 접근하는 것을 막는다.
- ▶ 객체의 외부에서 객체 내부의 데이터를 마음대로 읽고 쓸 경우 데이터의 무결성을 보장하기 힘들기 때문이다.
- ▶ 메소드를 통한 접근을 하게 되면 객체의 데이터를 변경할 경우 무결성 체크를 할 수 있다.

✓ 클래스를 정의할 때 필드는 private로 하여 객체 내부의 정보를 보호하고(정보은닉)
필드에 대한 Setter와 Getter를 두어 객체의 값을 변경하고 참조하는 것이 좋다.

- 외부에서 읽기만 가능하게 하기 위해선 Getter만 해당 필드에 대해서만 작성하면 된다.
- 외부에서 쓰기만 가능하게 하기 위해선 Setter만 해당 필드에 대해서만 작성하면 된다.
- Getter와 Setter가 없으면 객체 내부 전용 변수가 된다.

연습문제

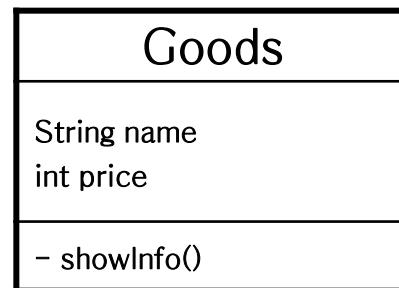
: 클래스 정의/확장 연습

[문제]

아래와 같이 클래스를 정의하여 프로그램을 작성하세요

Goods 클래스를 만드세요

- 1) 필드 접근자를 private로 작성해서 외부에서 접근할 수 없게 합니다.
- 2) 필드에 값을 저장할 수 있도록 set메소드를 만드세요.
- 3) 필드에 값을 읽을 수 있도록 get메소드를 만드세요.
- 4) 아래와 같이 상품의 모든 정보를 출력해 주는 showInfo()를 만드세요.



GoodsApp 클래스 만드세요

- 1) showInfo()메소드를 이용하여 다음과 같이 출력하세요.

```
<terminated> GoodsApp [Java Application] C:\Program Files\Java\jdk1.8
상품이름 : 니콘
가격 : 400000

상품이름 : LG그램
가격 : 900000

상품이름 : 마그컵
가격 : 2000
```

The screenshot shows the Java IDE's Console tab. It displays the output of a program named 'GoodsApp'. The output lists three items: Nikon (price 400,000), LG Gram (price 900,000), and Mag Cup (price 2,000). The console interface includes tabs for Problems, Javadoc, Declaration, and Console.

연습문제

: 클래스 정의/확장 연습

Goods 예제를 떠올리며 만들어 봅시다

[문제]

아래와 같이 클래스를 정의하여 프로그램을 작성하세요

Point 클래스를 만드세요

- x, y 좌표를 나타낼 수 있는 필드 작성
- x, y 좌표에 접근할 수 있는 getter/setter 메소드 작성
- 다음 실행 결과를 참조하여 draw()메소드 작성

PointApp 클래스 만드세요

- 1) draw() 메소드를 호출하여 다음과 같이 출력하세요



연습문제

: 클래스 정의/확장 연습

Goods 예제를 떠올리며 만들어 봅시다

[문제]

아래와 같이 클래스를 정의하여 프로그램을 작성하세요

Song 클래스를 만드세요

Song 클래스는 다음과 같은 필드를 가지고 있습니다.

- 노래의 제목을 나타내는 title
- 가수를 나타내는 artist
- 노래가 속한 앨범 제목을 나타내는 album
- 노래의 작곡가를 나타내는 composer
- 노래가 발표된 연도를 나타내는 year
- 노래가 속한 앨범에서 트랙 번호를 나타내는 track

1) 필드의 접근을 제한하고 getter/setter 메소드를 통해 접근하세요.

2) 노래정보를 출력하는 showInfo() 메소드를 작성하세요.

SongApp 클래스 만드세요

1) showInfo() 메소드를 호출하여 다음과 같이 출력하세요



The screenshot shows an IDE interface with a 'Console' tab selected. The console window displays the following output:

```
Problems @ Javadoc Declaration Console
No consoles to display at this time.

아이유, 좋은날 ( Real, 2010, 3번 track, 이민수 작곡 )
BIGBANG, 거짓말 ( Always, 2007, 2번 track, G-DRAGON 작곡 )
버스커버스커, 벚꽃엔딩 (버스커버스커1집, 2012, 4번 track, 장범준 작곡 )
```

Java Class

: 생성자 (Constructor)

- ▶ `new` 연산자와 같이 사용되어 클래스로부터 객체를 생성할 때 호출되고 객체의 초기화를 담당 한다.
- ▶ 생성자를 실행 시키지 않고 클래스로부터 객체를 만들 수 없다.
- ▶ 생성자가 성공적으로 실행되면 JVM의 Heap영역에 객체가 생성되고 객체의 참조 값이 참조변수에 저장 된다.
- ▶ 몇 가지 조건을 제외하면 메소드와 같다.
- ▶ 모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다.

Java Class

: 생성자 (Constructor)

▶ 생성자의 정의

- ▶ 생성자의 이름은 클래스와 같아야 한다
- ▶ 생성자의 리턴 값은 없다. (하지만 **void**를 쓰지 않는다)

```
접근자 클래스이름 ( 파라미터 ) {  
    // 인스턴스 생성시 수행할 코드  
    // 주로 인스턴스 변수의 초기화 코드  
}
```

```
public class Goods {  
    public Goods() {  
        // 초기화 코드  
    }  
  
    public Goods( String name, int price ) {  
        // 초기화 코드  
    }  
}
```

Java Class

: 생성자 (Constructor)

▶ 기본 생성자

- ▶ 매개 변수가 없는 생성자
- ▶ 클래스에 생성자가 하나도 없으면 컴파일러가 기본 생성자를 추가한다
- ▶ 생성자는 필요에 따라 여러 개 작성할 수 있다.

```
public class Goods {  
  
    public Goods() {  
        // 초기화 코드  
    }  
}
```

연습문제

: 클래스 정의/확장 연습

[문제]

- 1) Goods 클래스에서 생성되는 모든 필드 초기화 하는 생성자를 정의합니다.
- 2) 오류가 발생하면 오류가 발생하는 원인을 생각해 보세요.
- 3) 생성자 오버로딩을 확인합니다.

L

[문제]

- 1) Point 클래스의 기본 생성자와 모든 필드를 초기화 할 수 있는 생성자를 작성하고 테스트 합니다.

[문제]

- 1) Song 클래스의 기본 생성자와 모든 필드를 초기화 할 수 있는 생성자를 작성하고 테스트 합니다.

Java Class

: this

- ▶ **this** 키워드는 메서드 호출을 받는 객체를 의미한다
- ▶ 현재 사용중인 객체 그 자체를 의미한다
- ▶ **this()**는 클래스의 한 생성자에서 다른 생성자를 호출할 때 사용할 수 있다.

[문제]

- 1) Goods 클래스에서 이름만 입력 받아 초기화하는 생성자를 오버로딩합니다.
- 2) 모든 필드 입력 생성자에서 이 생성자를 호출하도록 합니다.
- 3) 이 생성자에서 다른 생성자를 호출하도록 합니다.

L

[문제]

- 1) Song 클래스에서 노래 제목과 가수만 입력 받아 필드를 초기화하는 생성자를 하나 더 오버로딩 합니다.
- 2) 모든 필드 입력 생성자에서 이 생성자를 호출하도록 합니다..

Java Class

: method overloading

반환값이 다른 것은 오버로딩이 아닙니다

- ▶ 하나의 클래스에 같은 이름의 메서드가 여러 개 존재할 수 있다
 - ▶ 각 메서드들은 매개변수 타입, 개수, 그리고 순서가 다른 형태로 구별된다
- * 메소드 시그너처(Signature) : 메소드 인자의 타입, 개수, 순서

[문제]

1) Point 클래스에 점을 안보일 수 있는 기능까지 추가된 draw() 메소드를 하나 더 추가하고 아래 실행결과가 나도록 테스트 하세요



The screenshot shows a Java application window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the following text:
<terminated> PointApp [Java Application] C:\Program Files\Java
점 [x=5, y=5]을 그렸습니다.
점 [x=10, y=23]을 그렸습니다.
점 [x=5, y=5]을 지웠습니다.
점 [x=10, y=23]을 지웠습니다.

연습문제

아래 TV 클래스의 main 메소드를 실행할 수 있도록, 요구 조건을 참조하여 TV 클래스를 정의 하세요.

- 1) 모든 필드는 private으로 접근 제어를 하고 getter만 작성합니다. (channel, volume, power 필드 read-only)
- 2) channel, volume, power의 초기값을 각각 7, 20, false 로 초기화 하는 생성자 작성
- 3) 기본 생성자 오버로딩
- 4) void power(boolean on) 메소드 구현
- 5) void channel(int channel) 메소드 구현 (1~255 유지)
- 6) void channel(boolean up) 메소드 오버로딩 (1~255 유지, 1씩 증감)
- 7) void volume(int volumem) 메소드 구현 (0 ~ 100 유지)
- 8) void volume(boolean up) 메소드 오버로딩 (0 ~ 100 유지, 1씩 증감)
- 9) void status() 메소드 구현(TV 정보 출력)

TV
int channel
int volume
boolean power
getChannel()
getVolume()
power(boolean)
channel(int)
channel(boolean)
volume(int)
volume(boolean)
status();

```
public class TVApp {  
    public static void main( String[] args ) {  
        TV tv = new TV( 7, 20, false );  
  
        tv.status();  
  
        tv.power( true );  
        tv.volume( 120 );  
        tv.status();  
  
        tv.volume( false );  
        tv.status();  
  
        tv.channel( 0 );  
        tv.status();  
  
        tv.channel( true );  
        tv.channel( true );  
        tv.channel( true );  
        tv.status();  
  
        tv.power( false );  
        tv.status();  
    }  
}
```

Java Class

: static 변수, instance 변수, local 변수

▶ 선언 위치에 따른 변수의 종류

```
public class Goods{
```

```
    static int countOfGoods; //클래스(static)변수  
    private String name; //인스턴스변수  
    private int price; //인스턴스변수
```

```
    public void setPrice(int price){
```

```
        int localVal; //지역변수
```

```
}
```

클래스영역

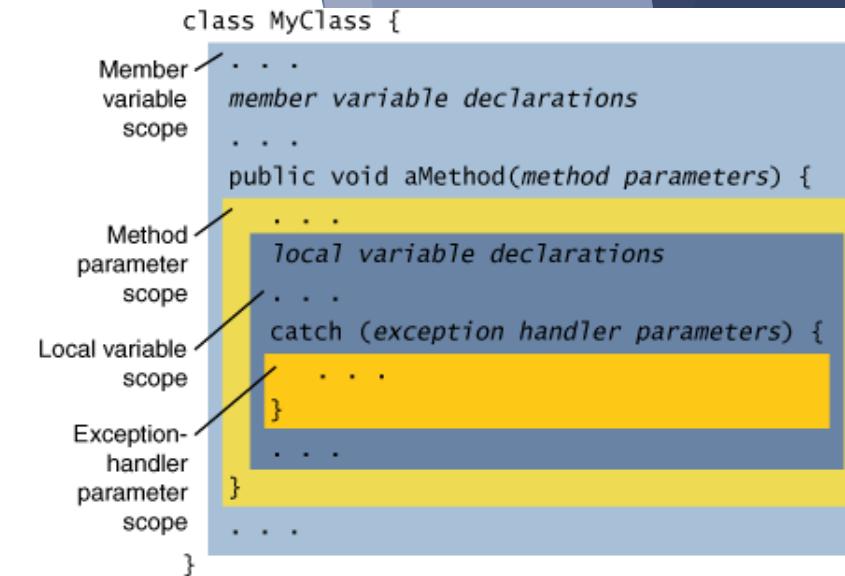
메소드영역

변수의 종류	선언위치	생성시기
클래스(static) 변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스 변수		인스턴스 생성 시
지역변수	메서드 영역	변수 선언문 수행 시

Java Class

: 변수의 범위 (Scope)

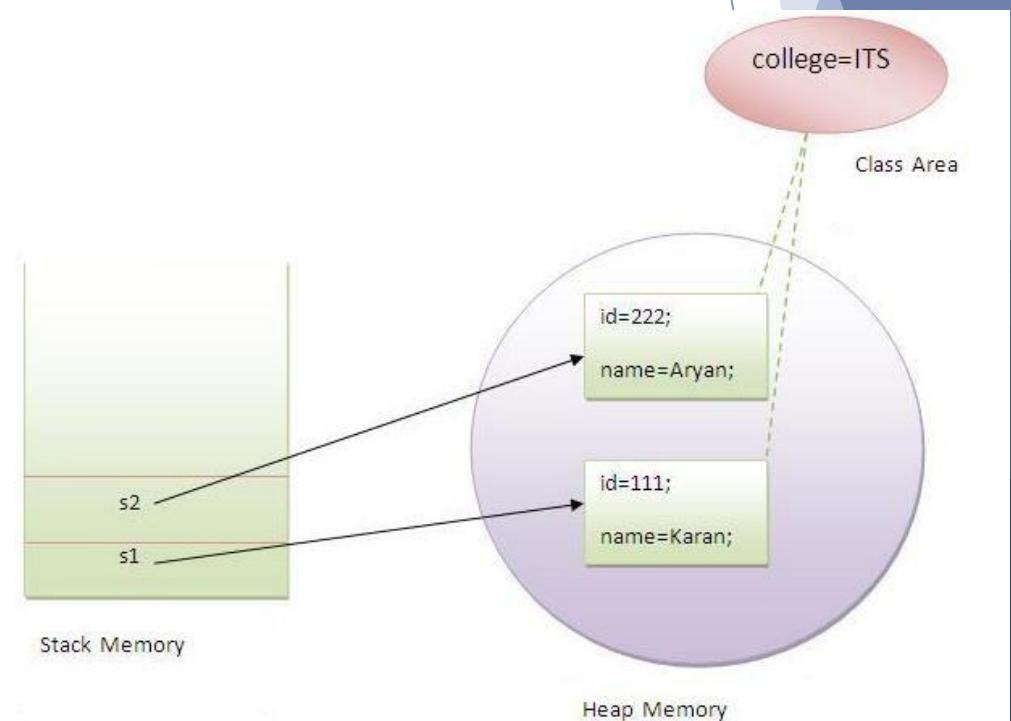
- ▶ 인스턴스 변수(instance variable)
 - ▶ 각 인스턴스의 개별적인 저장공간. 인스턴스마다 다른 값 저장 가능
 - ▶ 인스턴스 생성 후, ‘참조변수.인스턴스 변수명’으로 접근
 - ▶ 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지 컬렉터에 의해 자동 제거됨
- ▶ 클래스(static) 변수(class variable)
 - ▶ 같은 클래스의 모든 인스턴스들이 공유하는 변수
 - ▶ 인스턴스 생성 없이 ‘클래스이름.클래스(스태틱)변수명’으로 접근
 - ▶ 클래스가 로딩될 때 생성되고 프로그램이 종료될 때 소멸
- ▶ 지역 변수(local variable)
 - ▶ 메소드 내에 선언되며, 메소드의 종료와 함께 소멸
 - ▶ 조건문, 반복문의 블럭{} 내에 선언된 지역변수는 블럭을 벗어나면 소멸
- ▶ 인스턴스 변수와 클래스(static) 변수
 - ▶ 인스턴스 변수는 인스턴스가 생성될 때마다 생성되므로 인스턴스마다 각기 다른 값을 유지할 수 있지만, 클래스(static) 변수는 모든 인스턴스가 하나의 저장공간을 공유하므로 항상 공통된 값을 갖는다.



Java Class

: 클래스 (static) 멤버

- ▶ 전역 변수와 전역 함수를 만들 때 사용
- ▶ 모든 클래스에서 공유하는 전역 변수나 전역 함수를 만들어 사용할 수 있다
 - ▶ 클래스 멤버라고도 함
- ▶ 객체를 생성하지 않고 접근할 수 있다
- ▶ static 메서드 내에서는 **this** 사용 불가
- ▶ static 메서드 내에서는 static 멤버만 접근할 수 있다
- ▶ static 멤버의 초기화는 static 블록에서 할 수 있다
 - ▶ static 블록은 클래스가 메모리에 로딩될 때 실행된다
 - ▶ static 블록은 생성자보다 앞서 실행된다



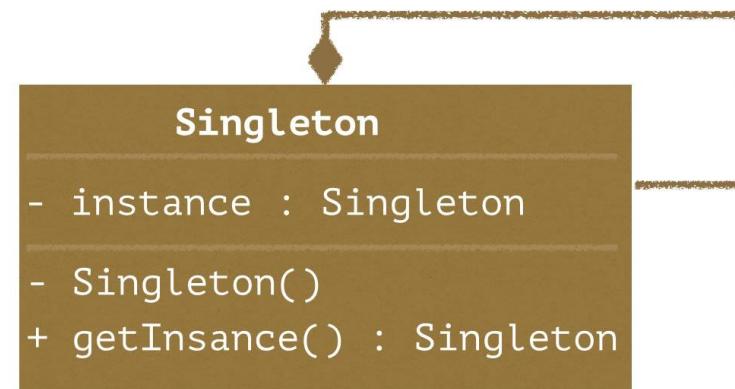
Java Class

: static의 활용 - Singleton

- ▶ 전체 프로그램 상에서 동일한 인스턴스를 사용해야 할 경우 활용
 - ▶ 어떠한 상황에서도 **단 하나**의 인스턴스만 유지한다
- ▶ 4대 디자인 패턴에 포함될 정도로 유명하고, 유용한 패턴
- ▶ 예) 스마트폰의 주소록

Singleton Design Pattern

“Ensure that a class has only one instance and provide a global point of access to it.”



Java Class

: 패키지(package)

- ▶ 패키지란?
 - ▶ 서로 관련 있는 클래스 또는 인터페이스들의 묶음
- ▶ 패키지 이용의 장점
 - ▶ 클래스들을 묶음 단위로 제공하여 필요할 때만 사용 가능 (`import` 문 이용)
 - ▶ 클래스 이름의 혼란을 막고 충돌을 방지
 - ▶ 패키지 단위의 접근 권한 지정 가능
- ▶ 사용법
 - ▶ 1) `import 패키지명.클래스명(인터페이스명); // 지정된 클래스(인터페이스)만 불러옴`
 - ▶ 예) `import java.applet.Applet; // java.applet 패키지 내의 Applet을 불러옴`
 - ▶ 2) `import 패키지명.*; // 패키지 내의 모든 클래스(인터페이스)를 불러옴`
 - ▶ 예) `import java.io.*; // java.io 패키지 내 모든 클래스(인터페이스)를 불러옴`

Java Class

: 새로운 패키지의 작성

- ▶ 패키지의 이름과 계층 구조를 결정
- ▶ 패키지를 위치시킬 디렉토리에 패키지의 계층 구조와 동일한 디렉토리 구조 생성
- ▶ 패키지에 추가할 클래스들을 생성하고 해당 디렉토리로 이동
- ▶ 새로 생성된 패키지가 위치한 디렉토리를 환경변수에 추가

```
package com.example.myproject;

public class MyClass {
    public void greeting( String name ) {
        System.out.println( "Hello " + name );
    }
}
```

```
package com.example.myproject.test;
import com.example.myproject.MyClass;

public class MyProjectTest {
    public static void main( String[] args ) {
        MyClass myClass = new MyClass();
        myClass.greeting( "Java" );
    }
}
```

접근 제한자

- ▶ 외부에서 접근할 수 있는 것과 없는 것을 구분하는 것이 바람직
- ▶ 클래스, 멤버 등에 적용 가능
- ▶ 접근자를 생략하면 **default** 제한자가 설정된다

제한자	적용대상	접근할 수 없는 클래스
public	클래스, 필드, 생성자, 메소드	없음
protected	필드, 생성자, 메소드	자식 클래스가 아닌 다른 패키지에 소속된 클래스
default	클래스, 필드, 생성자, 메소드	다른 패키지에 소속된 클래스
private	필드, 생성자, 메소드	모든 외부 클래스

Java Programming

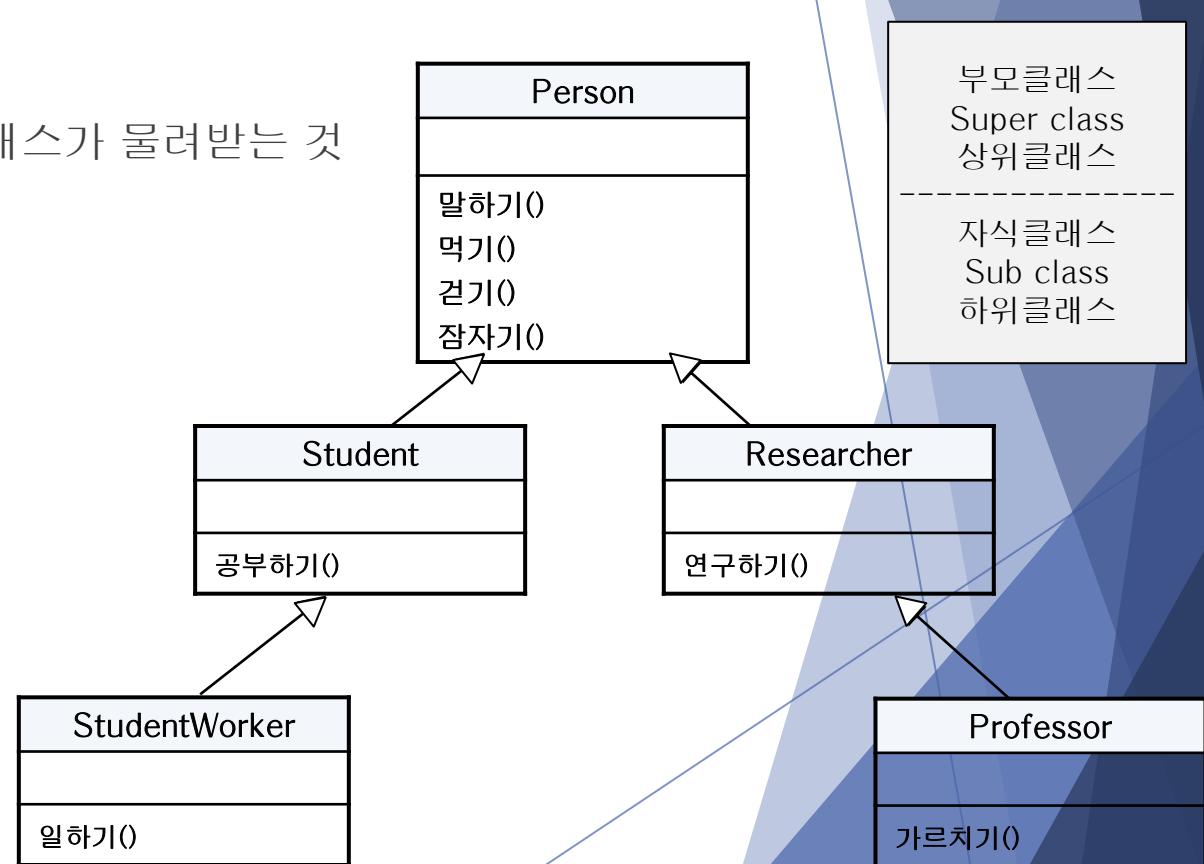
Inheritance and Polymorphism

상속과 다형성

Inheritance and Polymorphism

: 상속 (Inheritance)

- ▶ 상속
 - ▶ 부모 클래스에 정의된 필드와 메서드를 자식 클래스가 물려받는 것
- ▶ 상속의 필요성
 - ▶ 클래스 사이의 멤버 중복 선언 불필요
 - ▶ 필드, 메서드 재사용으로 클래스가 간결
 - ▶ 클래스간 계층적 분리 및 관리



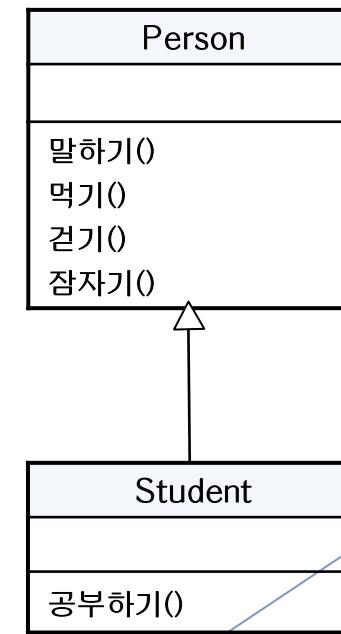
Inheritance and Polymorphism

: 상속 (Inheritance)

- ▶ 자바 언어 상속의 특징
 - ▶ 다중 상속을 지원하지 않음
 - ▶ 상속의 회수에 제한을 두지 않음
 - ▶ 최상위 클래스는 `java.lang.Object`
- ▶ 상속 선언의 예

```
public class Person {  
    public void tell();  
    public void eat();  
    public void walk();  
    public void sleep();  
}
```

```
public class Student extends Person  
{  
    public void study()  
}
```



Inheritance and Polymorphism

: 상속 (Inheritance)

▶ 상속과 생성자

- ▶ 자식 생성자에서 특별한 지시가 없으면 부모 클래스의 기본 생성자가 선택된다
- ▶ 부모 클래스의 특정 생성자를 호출해야 할 경우 `super()`로 명시적으로 호출
- ▶ **부모의 필드나 메서드에 접근시에는 `super` 키워드를 사용**

▶ 상속과 접근제한자

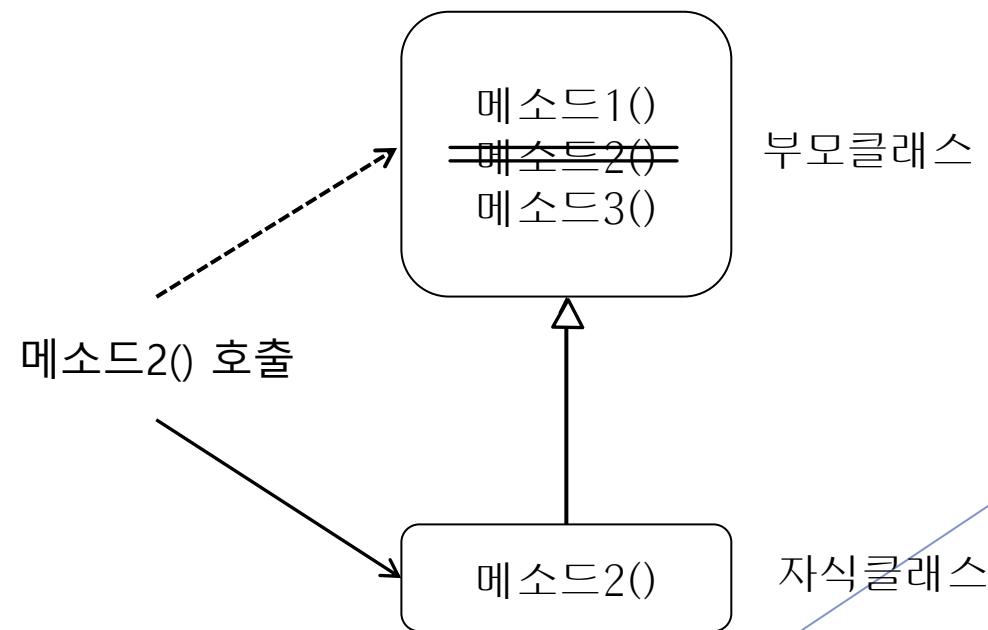
지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

public > protected > default > private

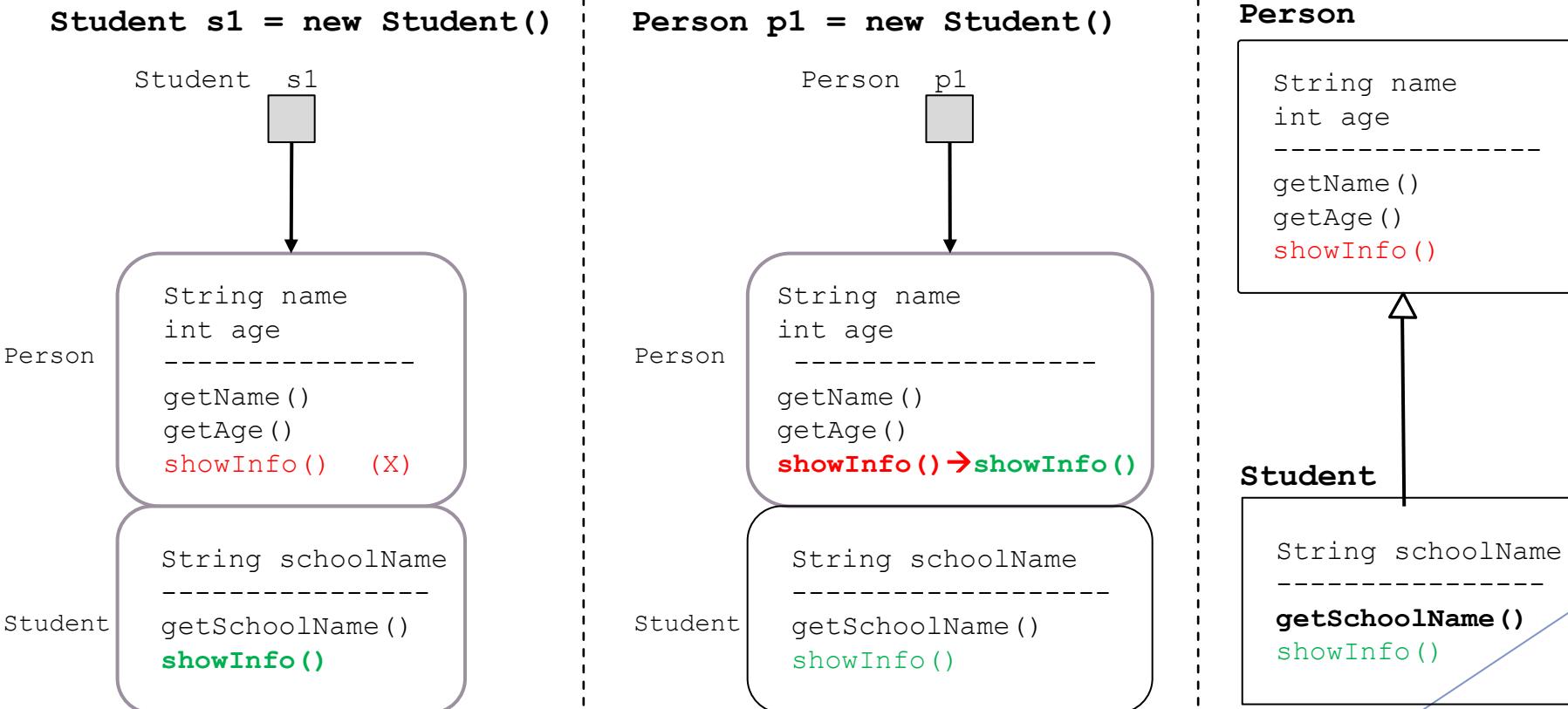
Inheritance and Polymorphism

: 메서드 오버라이딩 (Method Overriding)

- ▶ 부모 클래스와 자식 클래스의 메서드 사이에서 발생하는 관계
- ▶ 부모 클래스의 메서드를 동일한 이름으로 재 작성
 - ▶ 같은 이름, 같은 리턴타입, 같은 시그너처
- ▶ 부모 클래스 메서드 무시하기
- ▶ @Overriding



Inheritance and Polymorphism

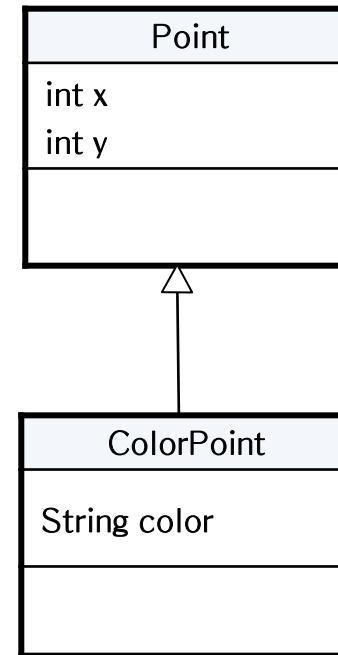


연습문제

: 상속 연습

[문제]

- Point 클래스를 만드세요.
✓생성자, getter/setter, draw()
- Point 클래스를 상속받아 ColorPoint 클래스를 만드세요.
✓생성자, getter/setter, draw()
- PointApp 클래스를 통해서 인스턴스를 생성하고 showInfo()를 통해 확인하세요.
✓`Point p = new Point(4,4);`
✓`ColorPoint cp1 = new ColorPoint("red");`
✓`ColorPoint cp2 = new ColorPoint(10,10,"blue");`
- 자식 클래스와 부모클래스의 생성자 순서를 확인하세요

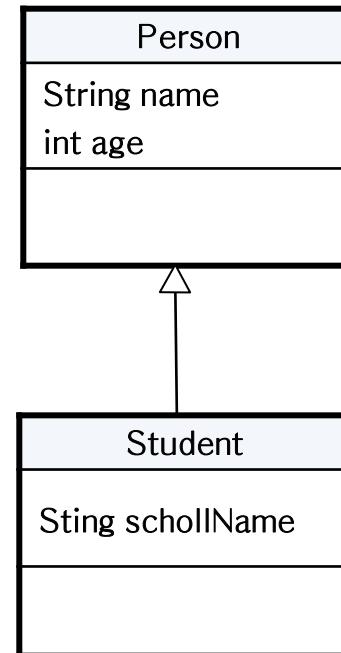


연습문제

: 상속 연습

[문제]

- Person 클래스를 만드세요.
✓생성자, getter/setter, showInfo()
- Person 클래스를 상속받아 Student 클래스를 만드세요.
✓생성자, getter/setter, showInfo()
- PersonApp 클래스를 통해서 인스턴스를 생성하고 showInfo()를 통해 확인하세요.
✓Person p = **new Person("정우성", 45);**
✓Student s1 = **new Student("서울고등학교");**
✓Student s2 = **new Student("이정재", 45, "한국고등학교");**
- 자식 클래스와 부모클래스의 생성자 순서를 확인하세요



Inheritance and Polymorphism

: Upcasting and Downcasting

- ▶ 업캐스팅 (Up Casting or Promotion)
 - ▶ 자식 클래스가 부모 클래스 타입으로 변환되는 것
 - ▶ 명시적으로 타입 변환을 하지 않아도 된다
- ▶ 다운캐스팅 (Down Casting)
 - ▶ 업캐스팅된 것을 원래대로 되돌리는 것
 - ▶ 명시적으로 타입 변환을 해야 한다
 - ▶ 다운캐스팅시 어떤 클래스를 객체화한 것인지 알고자 한다면 **instanceof** 를 사용한다
- ▶ 다형성(Polymorphism)
 - ▶ 같은 타입이지만, 실행 결과가 다른 객체를 이용할 수 있는 성질
 - ▶ 자바는 부모 클래스로의 타입 변환을 허용한다

Java Programming

Abstract Class and Interface

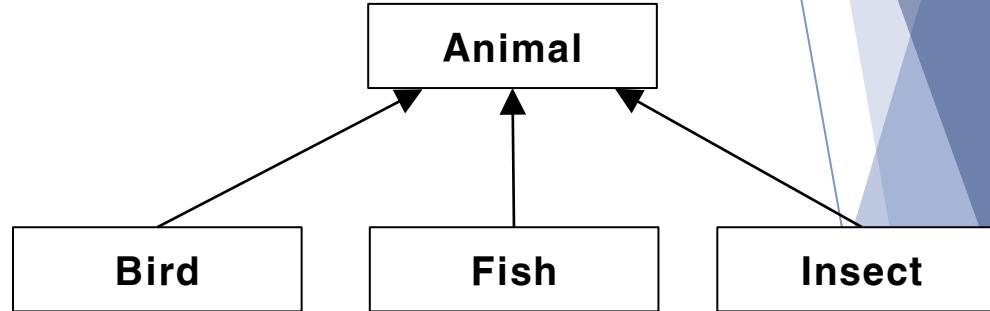
추상 클래스와 인터페이스

Abstract Class and Interface

: 추상 클래스

- ▶ 추상화
 - ▶ 객체의 속성과 기능 중 중요한 것들은 남기고 필요 없는 것은 없애는 것
 - ▶ 또는 객체들간의 공통되는 특성을 추출하는 것
- ▶ 추상 클래스
 - ▶ 실제 클래스의 공통적인 특성들을 추출해서 선언한 클래스
 - ▶ 실제 클래스를 만들기 위한 **부모 클래스**로만 사용
 - ▶ 스스로 객체가 될 수는 없다
 - ▶ 확장을 위한 용도로만 사용
 - ▶ 하나 이상의 추상 메서드를 가짐
 - ▶ 속성(필드)와 기능(메서드)을 정의할 수 있다
- ▶ 추상 메서드
 - ▶ 구현이 불가능한 메서드로서 선언만 한다
 - ▶ 추상 클래스를 상속하는 실제 자식 클래스는 추상 메서드를 반드시 구현해야 한다
 - ▶ 추상 메서드는 추상 클래스에만 존재한다

실체 클래스를 위한 설계 규격



Abstract Class and Interface

: 추상 클래스

- ▶ 추상 클래스의 선언

- ▶ 클래스 선언에 **abstract** 키워드

```
public abstract class 클래스명 {  
    //필드  
    //생성자  
    //메소드  
}
```

- ▶ 추상 클래스의 상속

- ▶ **extends** 키워드 사용
 - ▶ 추상 클래스를 상속하는 클래스는 반드시 추상 클래스 내의 추상 메서드를 구현해야 함
 - ▶ 특정 기능의 구현을 강요하는 측면도 있음 (예: 자동차의 브레이크 기능은 꼭 구현되어야 함)

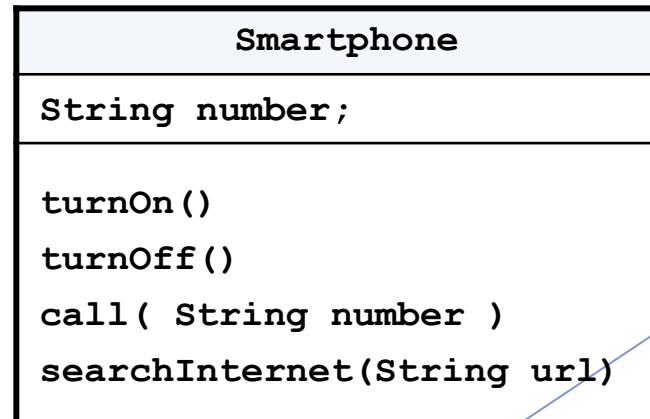
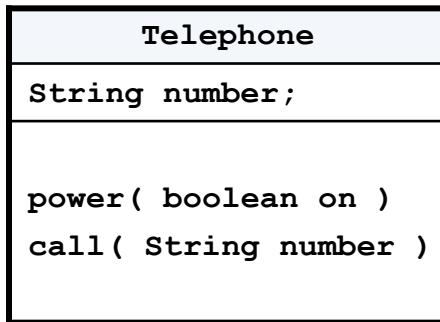
- ▶ 활용

- ▶ 여러 클래스들이 **상당수 공통점**을 가지고 있으나 **부분적으로 그 처리 방식이 다른 경우** 부모 클래스를 추상 클래스로 정의하여 자식 클래스들이 각각 해당 메서드를 구현

연습문제

: 추상 클래스

- 연습문제
 - Shape 클래스를 상속받은 Circle, Rectangle 클래스를 정의 하세요.
 - Shape 클래스는 추상 클래스로 추상 메소드 double area(), void draw()를 가짐
 - Shape 클래스를 상속 받은 각 클래스들은 자신만의 area(), draw() 메소드를 구현
- 연습문제
 - 다음 두 클래스의 공통된 속성과 기능을 추출하여 부모 클래스 Phone을 정의한 후, Telephone(유선) 과 SmartPhone(무선) 클래스도 정의하고 테스트해 보세요.
 - 전원메소드를 turnOn(boolean on) 메소드로 사용하세요



Abstract Class and Interface

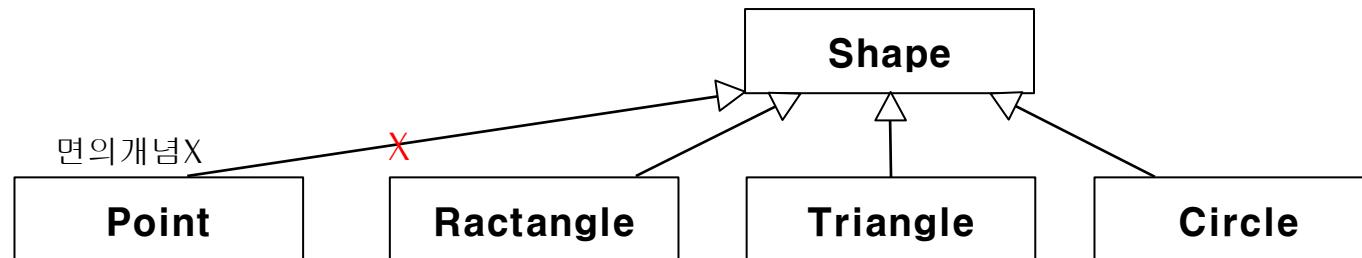
: 인터페이스 (Interface)

▶ 개념

- ▶ 서로 관계가 없는 물체들이 상호작용을 하기 위해 사용하는 장치나 시스템
- ▶ 클래스 구조상 관계와 상관 없이 클래스들에 의해 구현될 수 있는 규약

▶ 사용 목적

- ▶ 클래스들 사이의 유사한 특성을 부자연스러운 상속 관계를 설정하지 않고 얻어냄
- ▶ 개발 코드를 수정하지 않고, 가용하는 객체를 변경할 수 있도록 하기 위함



▶ 활용

- ▶ 하나 혹은 그 이상의 클래스들에서 똑같이 구현되어질 법한 메서드를 선언하는 경우
- ▶ 클래스 자체를 드러내지 않고 객체의 프로그래밍 인터페이스를 제공하는 경우

Abstract Class and Interface

: 인터페이스 (Interface)

- ▶ 인터페이스 선언

```
public interface 인터페이스명 {  
    //추상메소드 (abstract 키워드를 지정하지 않아도 됨)  
}
```

- ▶ 인터페이스 구현

```
public class Point implements 인터페이스명 {  
    //추상메소드 구현 (abstract 키워드를 지정하지 않아도 됨)  
}
```

- ▶ 다중 상속을 지원하지 않는 자바에서 다중 상속의 장점을 활용할 수 있음

- ▶ 인터페이스는 다중 implements를 할 수 있다

```
public class Point implements Drawable, Resizeable {  
}
```

연습문제

: interface 연습

- ▶ Shape 예제에서 Point 클래스를 추가하고 Drawable 인터페이스를 추가해 봅니다

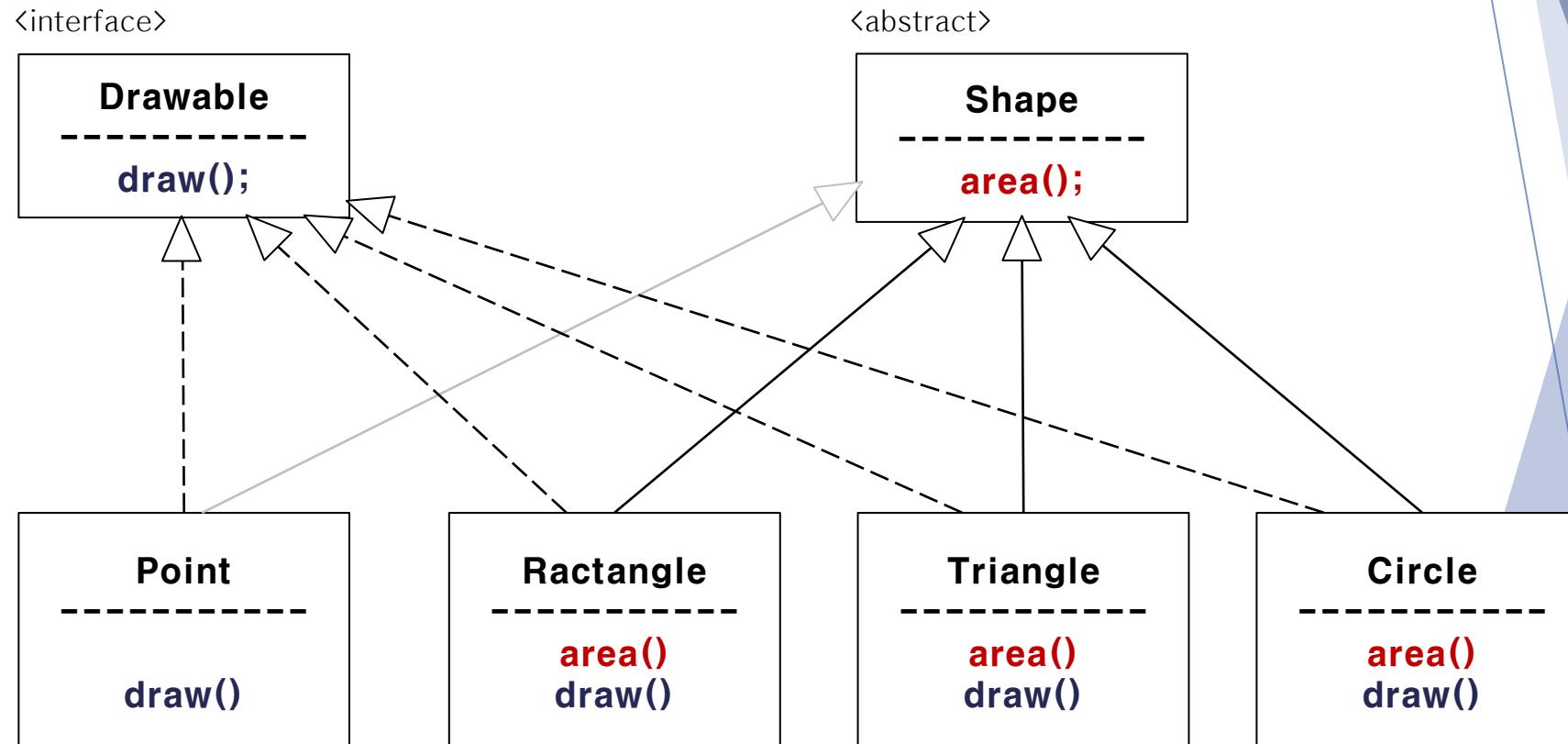
```
public interface Drawable {  
    public void draw();  
}
```

- ▶ instanceof 연산자

```
Shape c = new Circle();  
  
// 객체가 Circle 클래스의 인스턴스 인가?  
System.out.println( c instanceof Circle );  
  
// 객체가 Drawable 인터페이스를 구현하였는가?  
System.out.println( c instanceof Drawable );  
  
// 객체가 Rectangle 클래스의 인스턴스 인가?  
System.out.println( c instanceof Rectangle );  
  
// 객체가 Shape 클래스의 인스턴스 인가?  
System.out.println( c instanceof Shape );
```

Abstract Class and Interface

: 추상클래스 vs 인터페이스 (Interface)



면의개념이 아님(shape이 될 수 없음)
그림판에서는 같이 관리 되어야 함

Abstract Class and Interface

: 인터페이스 (Interface)

▶ 일반 클래스 vs 추상 클래스 vs 인터페이스

	일반클래스	추상클래스	인터페이스
메소드 (Method)	모두 실제 메소드 *	실체 메소드 추상 메소드	모두 추상 메소드
필드 (Instance Variable)	가질 수 있음	가질 수 있음	가질 수 없음 상수 필드는 가능
객체화 (Instantiation)	가능	불가	불가

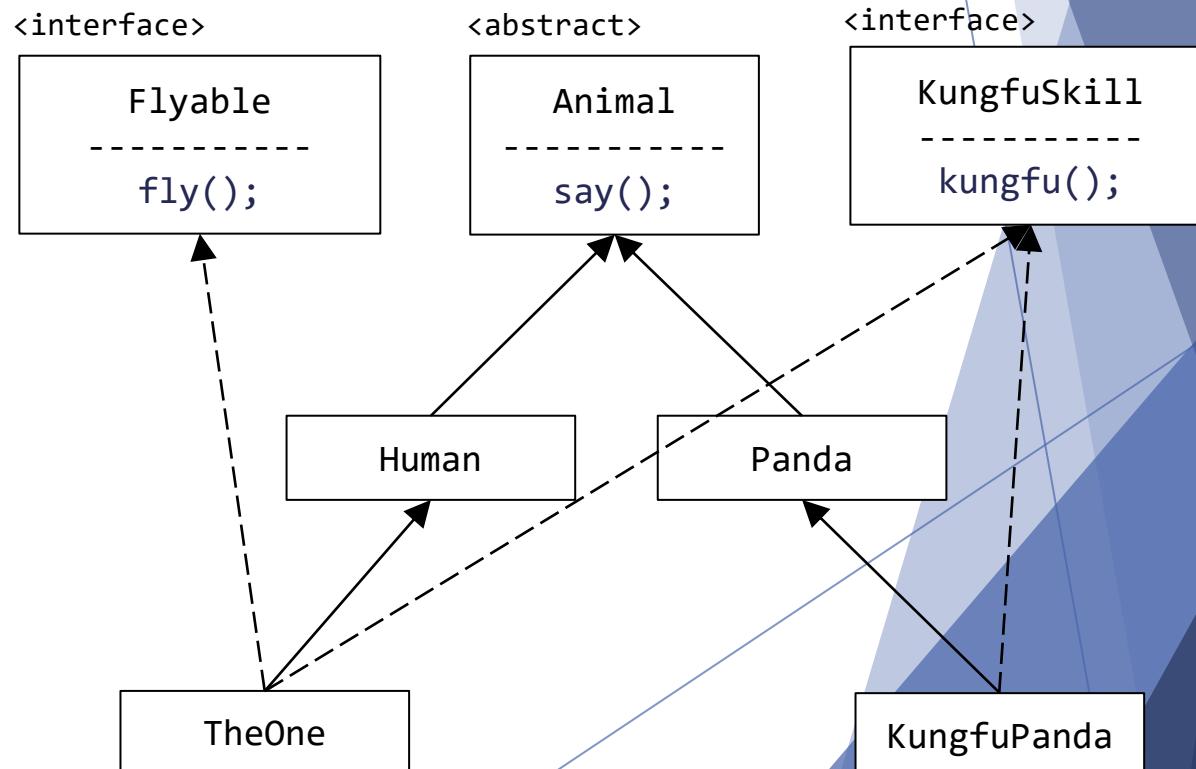
▶ 실제 메서드 (Concrete Method)

- ▶ 메서드의 구현을 포함한 일반적인 메서드를 **Concrete Method**라 함
- ▶ 추상 메서드 (Abstract Method)의 반대 개념

Abstract Class and Interface

: 정리

- ▶ 추상 클래스와 인터페이스 모두 규약으로서의 의미가 강하지만
 - ▶ 추상 클래스는 객체를 일반화한 공통 필드와 메서드의 정의에 집중 (종적 확장)
 - ▶ 인터페이스는 자연스러운 상속 관계를 해치지 않으면서도 추가 내용을 규정할 수 있음(횡적 확장)



Java Programming

Exception Handling

예외 처리

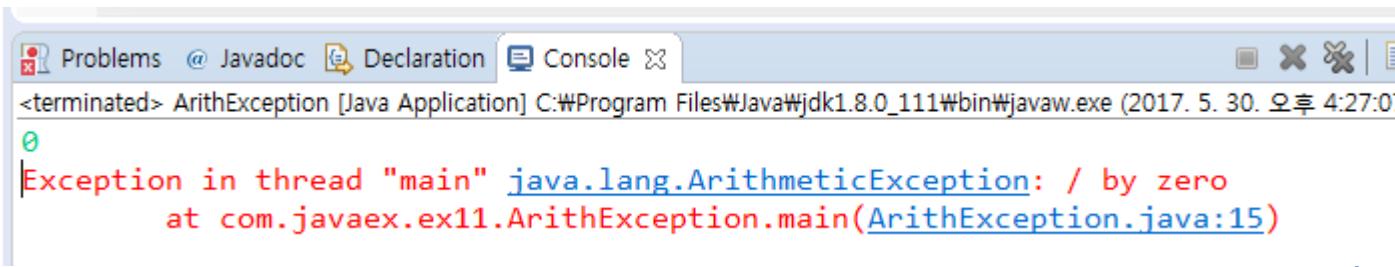
Exception Handling

- ▶ 예외 (Exception)
 - ▶ 프로그램이 실행되는 동안 발생할 수 있는 비정상적인 상태
 - ▶ 컴파일시의 에러가 아닌 실행시의 에러를 예외라 함
- ▶ 자바의 예외 처리
 - ▶ Exception 클래스 정의
 - ▶ 기본적인 예외는 자바에 미리 정의된 예외를 통해 처리 가능
 - ▶ 사용자가 필요한 예외를 직접 정의할 수 있음
 - ▶ 예상되는 예외는 미리 처리해주면 비정상적인 프로그램의 종료를 피할 수 있음
 - ▶ 예외처리는 프로그램의 신뢰도를 높여줌

Exception Handling

: Example

```
public class ArithException {  
  
    public static void main(String[] args) {  
  
        double result;  
        int num;  
  
        Scanner sc = new Scanner(System.in);  
        num = sc.nextInt();  
  
        result = 100/num; // java.lang.ArithemeticException 발생  
  
        System.out.println(result); // 예외 발생으로 수행되지 않음  
        sc.close();  
  
    }  
}
```



The screenshot shows the Java IDE's console tab. The output window displays the following text:
<terminated> ArithException [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (2017. 5. 30. 오후 4:27:07)
0
Exception in thread "main" java.lang.ArithemeticException: / by zero
at com.javaex.ex11.ArithException.main(ArithException.java:15)

Exception Handling

: try ~ catch ~ finally

0

```
try {  
    // 예외가 발생할 가능성이 있는 실행문  
} catch ( 처리할 예외 타입 선언 ) {  
    // 예외 처리문  
} finally {  
    // 예외 발생 여부와 상관없이 무조건 실행되는 문장 ( 생략가능 )  
}  
4
```

1

2

3

- ▶ try 블록에서 예외가 발생했을 경우 : 0 -> 1 -> 2 -> 3 -> 4
- ▶ try 블록에서 예외가 발생하지 않았을 경우 : 0 -> 1 -> 3 -> 4

Exception Handling

: Exception Class

▶ 주요 예외 클래스

예외 클래스	예외 발생 경우
ArithmeticException	어떤 수를 0으로 나눌 때
NullPointerException	null 객체를 참조할 때
ClassCastException	변환할 수 없는 타입으로 객체를 변환 할 때
ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못된 경우
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않은 타입의 숫자로 변환한 경우
IOException	입출력 동작 실패, *인터럽트 발생할 경우

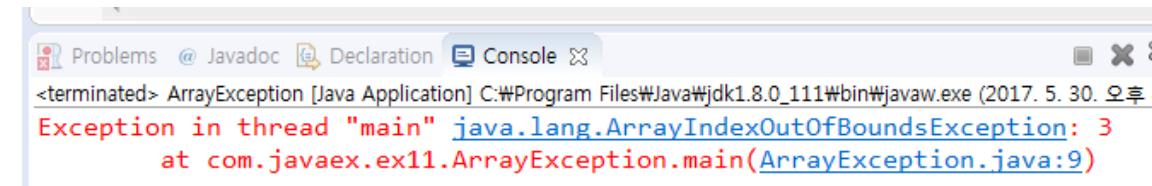
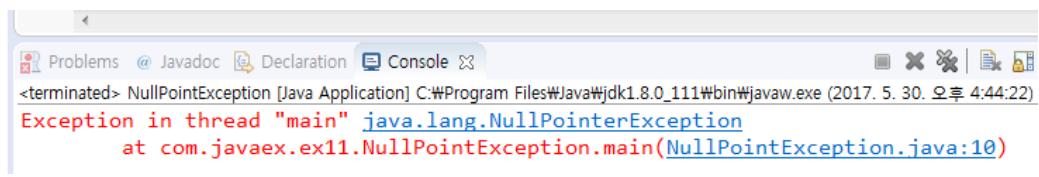
연습문제

: 예외 처리

ArithmetiException을 처리했던 방식과 마찬가지로
ArrayException과 NullPointerException을 처리해보자

```
public class ArrayExceptionEx {  
  
    public static void main(String[] args) {  
  
        int[] intArray = new int[]{3,6,9};  
  
        System.out.println(intArray[3]);  
  
    }  
}
```

```
public class NullPointExceptionEx {  
  
    public static void main(String[] args) {  
  
        String str = new String("hello");  
        str = null;  
  
        System.out.println(str.toString());  
  
    }  
}
```



Exception Handling

: Exception Class

- ▶ 예외 (Exception)의 구분
 - ▶ Checked Exception : 컴파일할 때 확인되는 예외 -> 반드시 예외 처리가 필요함
 - ▶ Unchecked Exception : 실행 시점에서 확인되는 예외 -> 예외 처리 없어도 컴파일 됨
- ▶ 예외 처리가 필요한 시점
 - ▶ 파일을 다루는 경우
파일이 없거나 다른 프로세스에 의해 사용중인 경우 예외 발생
 - ▶ 입출력을 다루는 경우
이미 닫힌 입출력 스트림에 대해 작업하려 할 경우 예외 발생
 - ▶ 네트워크를 이용한 데이터 통신
서버나 클라이언트 한 쪽에서 응답이 없는 경우
네트워크 상태가 좋지 않아 정해진 시간 동안 데이터를 받지 못하는 경우

Exception Handling

: 강제 예외 발생

- ▶ 메서드 정의시 예외 처리
 - ▶ 해당 메서드를 호출하는 메서드에서 예외를 처리하도록 명시한다
 - ▶ throws 키워드를 사용하여 예외의 종류를 적어준다

ThrowsExepApp.java

```
public class ThrowsExceptApp{  
    public static void main( String[] args ) {  
  
        ThrowsExcept except= new ThrowsExcept();  
        // IO exception 발생  
        except.executeExcept();  
  
    }  
}
```

ThrowsExep.java

```
public class ThrowsExcept {  
    public void executeExcept() throws IOException {  
  
        System.out.println( “강제예외발생” );  
        throw new IOException(); // 강제로 예외 발생  
    }  
}
```

Java Basic APIs

Language API

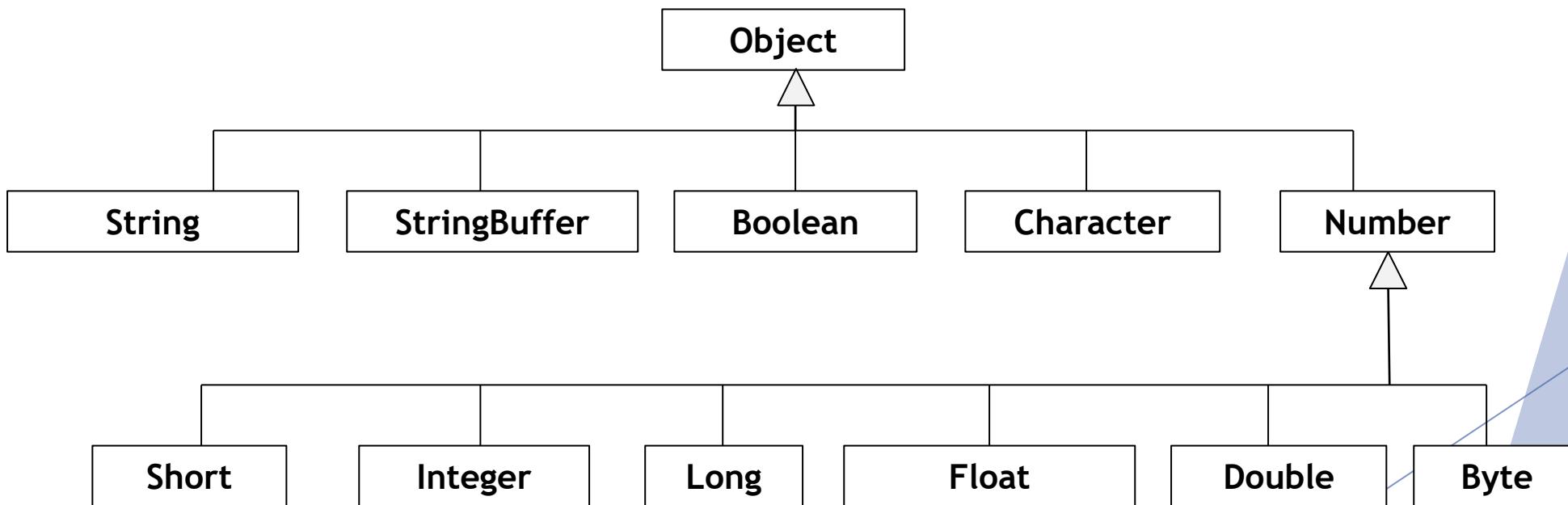
Java Basic APIs

Object Class

Language API

: java.lang

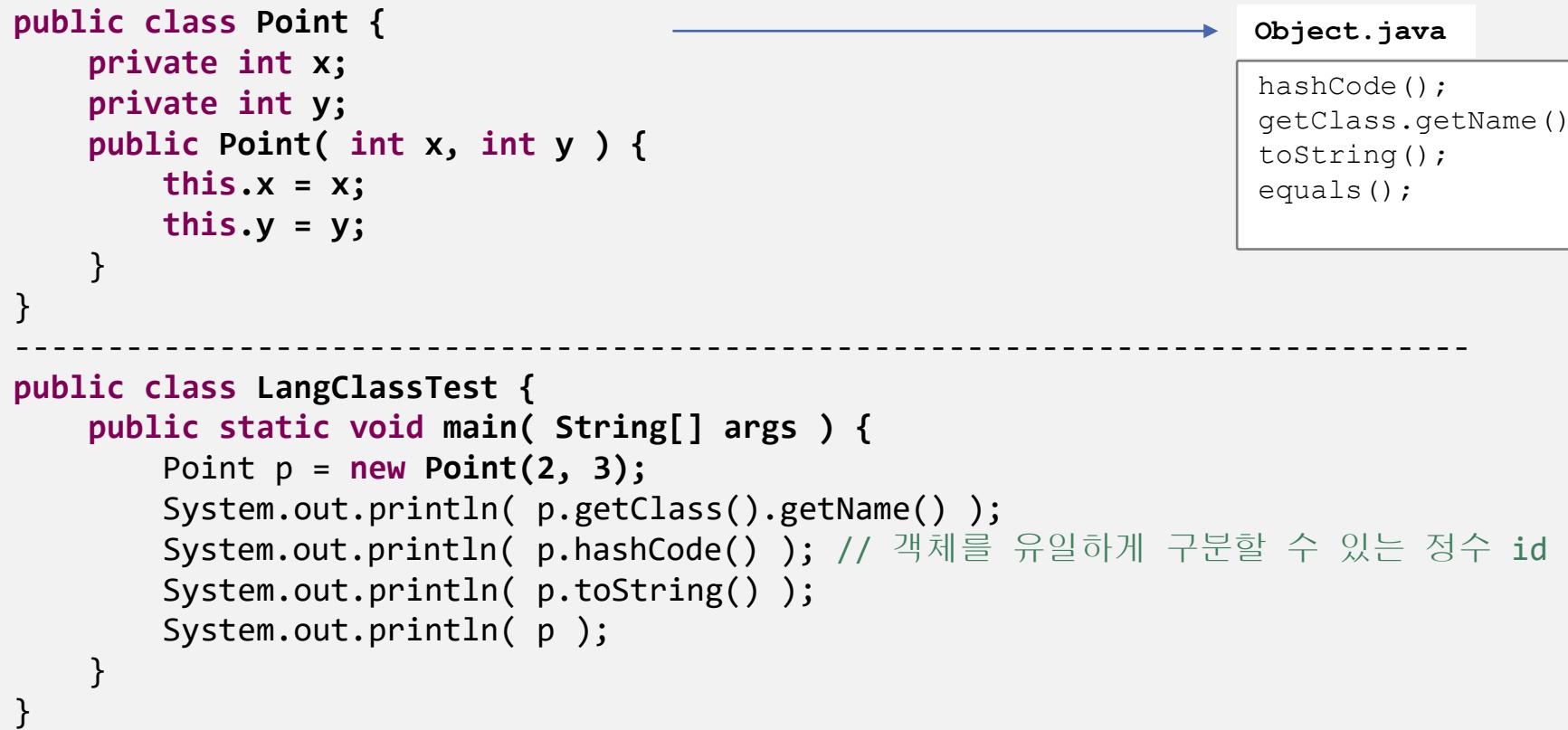
- ▶ 자바 프로그램에서 가장 많이 사용하는 패키지
- ▶ import 하지 않아도 자동으로 포함됨



Language API

: Object 클래스

- ▶ 모든 클래스의 최상위 클래스
- ▶ 명시적으로 `extends java.lang.Object` 하지 않아도 자동으로 상속받게 된다



기존에 만들어 둔 Point 클래스의
toString 메서드를 오버라이드 해 봅시다

Language API

: Object 클래스 toString() 메서드 오버라이드

- ▶ `toString()` : 객체의 정보를 문자열로 반환
- ▶ Object의 `toString` :
 - ▶ `getClass().getName() + "@" + Integer.toHexString(hashCode())`

`Object.java`

```
hashCode();
getClass.getName();
toString();
equals();
```

```
public class Point {
    private int x;
    private int y;
    public Point( int x, int y ) {
        this.x = x;
        this.y = y;
    }
    public String toString() {
        return "Point(" + x + "," + y + ")"; // Point 클래스만의 toString()
    }
}
```



Language API

: equals()

- ▶ 두 객체의 비교시 == 와 Object 클래스의 equals() 메서드를 사용한다
- ▶ == 와 equals()의 차이 : **확실히 구분**하여 사용
 - ▶ == 참조변수값 비교
 - ▶ equals() : 정의한 값 비교
- ▶ 참조 변수값을 먼저 비교한다
- ▶ 참조변수값이 같으면 두 객체는 같은 것으로 한다
- ▶ 참조변수값이 다르면 두 객체의 속성값을 비교한다

Language API

: 객체 비교 == vs equals() 메서드

```
public class LangClassTest {  
    public static void main(String[] args) {  
        Point a = new Point(2, 3);  
        Point b = new Point(2, 3);  
  
        System.out.println( a == b );  
        System.out.println( a.equals( b ) );  
  
        String s1 = new String("hello");  
        String s2 = new String("hello");  
  
        System.out.println( s1 == s2 );  
        System.out.println( s1.equals( s2 ) );  
    }  
}
```

Language API

: equals() 오버라이딩

```
public class Point {  
    private int x;  
    private int y;  
    public Point( int x, int y ) {  
        this.x = x;  
        this.y = y;  
    }  
    public boolean equals( Point p ) {  
        if( x == p.x && y == p.y ) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}  
  
-----  
public class LangClassTest {  
    public static void main( String[] args ) {  
        Point a = new Point(2, 3);  
        Point b = new Point(2, 3);  
        System.out.println( a == b );  
        System.out.println( a.equals( b ) );  
    }  
}
```



실습예제

[문제]

- int 타입의 x, y, radius 필드를 가지는 Circle 클래스를 작성하세요.
- equals() 메소드를 재정의 하여 두 개의 Circle 객체의 반지름이 같으면 두 Circle 객체가 동일한 것으로 판별하도록 하세요.

CircleApp.java

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

        Circle a = new Circle(6, 4, 10);
        Circle b = new Circle(2, 12, 10);
        Circle c = new Circle(3, 3, 12);
        Circle d = c;

        System.out.println(a.equals(b));
        System.out.println(a.equals(c));
        System.out.println(a.equals(d));
        System.out.println(d.equals(c));
    }
}
```

실습예제

[문제]

- Rectangle 클래스를 만들고 equals() 사용해 봅시다.
- int 타입의 width, height의 필드를 가지는 Rectangle 클래스를 작성하고 인스턴스를 생성합니다.
- 구해지는 면적이 같으면 두 객체가 같은 것으로 판별하도록 equals()를 오버라이딩 하세요.

RectangleApp.java

```
public class RectangleApp{  
    public static void main(String[] args){  
  
        Rectangle a = new Rectangle(6, 4);  
        Rectangle b = new Rectangle(2, 12);  
        Rectangle c = new Rectangle(3, 3);  
        Rectangle d = c;  
  
        System.out.println(a.equals(b));  
        System.out.println(a.equals(c));  
        System.out.println(a.equals(d));  
        System.out.println(d.equals(c));  
    }  
}
```

Language API

: 객체의 복제 - 얇은 복제

- ▶ 객체 복제 : 원본 객체의 값과 동일한 값을 가진 새로운 객체를 생성하는 것
- ▶ 얇은 복제 : 단순히 필드 값을 복사하는 방식으로 객체를 복제하는 것
- ▶ **Cloneable** 인터페이스를 구현하여 **clone** 메서드를 사용 가능하게 해야 한다

```
public class Point implements Cloneable {  
    private int x;  
    private int y;  
  
    // ...  
    public Point getClone() {  
        Point clone = null;  
        try {  
            clone = (Point)clone();  
        } catch (CloneNotSupportedException e) {}  
        return clone;  
    }  
}
```



Language API

: 객체의 복제 - 깊은 복제

- ▶ 얕은 복제는 참조 타입 필드의 경우, 주소만 복사되기 때문에 참조 타입의 필드는 같은 주소를 갖게 된다
- ▶ 깊은 복제 : 참조하고 있는 객체도 복사하는 것
- ▶ **Cloneable** 인터페이스를 구현한 후, **clone** 메서드를 오버라이드 하여 내부의 참조 객체도 복제해야 한다

```
public class Scoreboard implements Cloneable {  
    private int scores[];  
  
    // ...  
    @Override  
    protected Object clone() throws CloneNotSupportedException{  
        //먼저 얕은 복제를 시도  
        Scoreboard clone = (Scoreboard)super.clone();  
        //내부 참조 객체 복제 시도  
        clone.scores = Arrays.copyOf(scores, scores.length);  
        return clone;  
    }  
}
```

Java Basic APIs

String Class

Language API

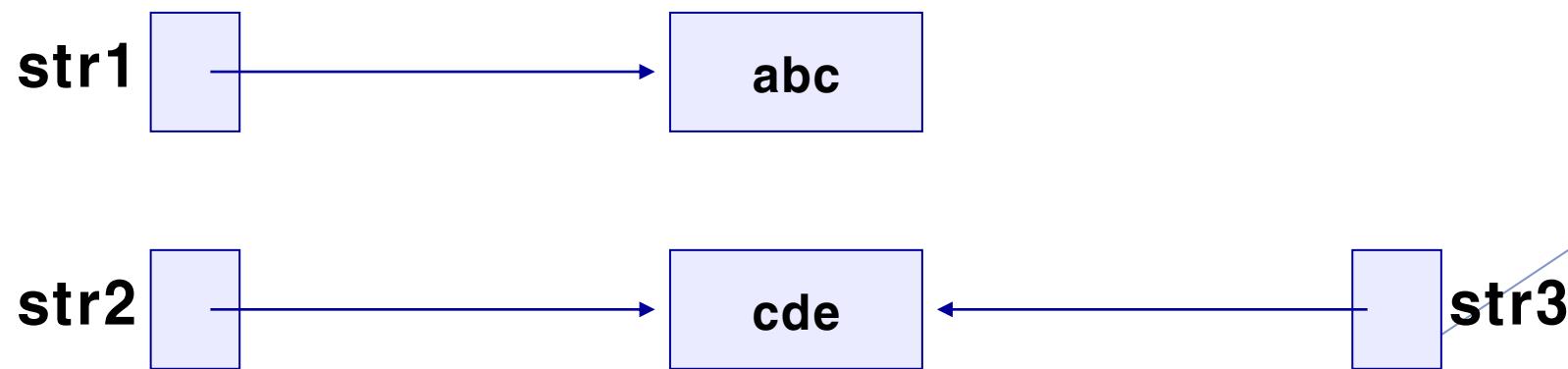
: String 클래스 - Character vs String

- ▶ 문자 (Character)
 - ▶ 기본 자료형 (char)
 - ▶ 문자 선언 `char letter;`
 - ▶ 문자 할당 `letter = 'A';`
 - ▶ 문자 사용 `System.out.println(letter); System.out.println((int)letter);`
- ▶ 문자열 (String) 클래스
 - ▶ 연속된 문자 (character)들을 저장하고 다루기 위한 클래스
 - ▶ 문자열 상수: “”로 둘러싸인 문자열 -> 한번 만들어진 **String** 객체는 변경 불가 (immutable)
 - ▶ 문자열 선언 `String greeting;`
 - ▶ 문자열 할당 `greeting = "Hello Java!";`
 - ▶ 문자열 사용 `System.out.println(greetin);`
 - ▶ 특수 문자의 표현 (Escape characters)
 - ▶ \', \", \\, \n, \r, \t 등

Language API

: String 클래스 - 메모리 그림 (I)

```
String str1;  
String str2, str3; // String 클래스변수 str1, str2, str3 선언  
str1 = "abc"; // str1은 생성된 String 클래스의 객체(Object)를 가리킴  
str2 = "cde"; // str2은 생성된 String 클래스의 객체(Object)를 가리킴  
str3 = str2; // str3에 str2의 주소 할당
```



Language API

: String 클래스 - String 연산

마지막 문자는 `length - 1`

Index	0	1	2	3	4	5	6	7	8	9	10
char	H	e	I	I	o		J	A	V	A	!

- ▶ “+” 연산자 : 문자열의 연결. 문자열과 다른 타입의 연결은 문자열로 변환되어 연결

```
String greet = "Hello";
String name = "JAVA";
System.out.println( greet + name + "!" );
System.out.println( greet + " " + name + "!" );
```

- ▶ Index

- ▶ String 객체 내의 문자 인덱싱은 배열과 같이 0부터 시작됨
- ▶ `charAt(position)` : 해당 위치의 문자를 반환
- ▶ `Substring(start, end)` : start부터 end까지의 문자들을 새로운 문자열로 반환

```
String greeting = "Hello JAVA!";
greeting.charAt(0);
greeting.charAt(10);
greeting.substring(1, 3);
```

Example

: String Class

Q) 메모리 그림과 API 문서를 활용해서 화면에 출력되는 값을 예상해 보십시오.

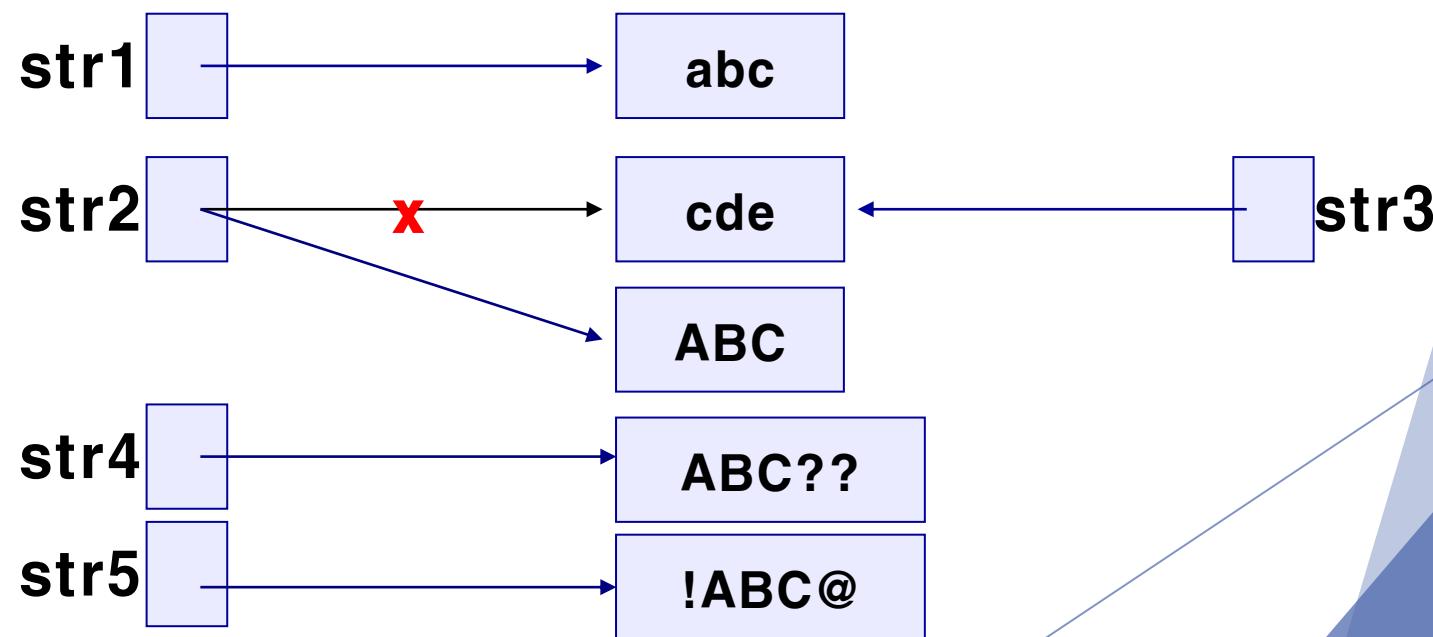
```
String str1 = "abc";
String str2;
str2 = str1.toUpperCase();
String str4 = str2.concat("??");
String str5 = "!.concat(str2).concat("@");

System.out.println("str1: " + str1);
System.out.println("str2: " + str2);
System.out.println("str4: " + str4);
System.out.println("str5: " + str5);
```

Language API

: String 클래스 - 메모리 그림 (II)

```
str2 = str1.toUpperCase();
String str4 = str2.concat( "??");
String str5 = "!".concat(str2).concat( "@" );
```

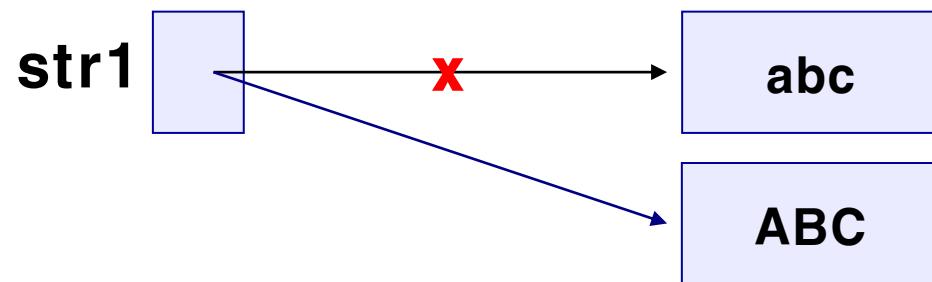


Language API

: String 클래스 - 메모리 그림 (III)

- ▶ Java의 String 객체는 한번 생성하면 변경할 수 없고 재할당시 새로운 String 클래스 객체가 생성

```
str1 = str1.toUpperCase();
```



Language API

: String 클래스 - 스트링 리터럴 vs new String()

```
public class StringTest {  
    public static void main( String[] args ) {  
        String s1 = "hello";  
        String s2 = "hello";  
  
        String s3 = new String("hello");  
        String s4 = new String("hello");  
  
        System.out.println( s1 == s2 );  
        System.out.println( s3 == s4 );  
        System.out.println( s2 == s4 );  
    }  
}
```

Language API

: String class methods

스트링 연관 메

public class String		
String(String s)		<i>create a string with the same value as s</i>
int length()		<i>number of characters</i>
char charAt(int i)		<i>the character at index i</i>
String substring(int i, int j)		<i>characters at indices i through (j-1)</i>
boolean contains(String substring)		<i>does this string contain substring?</i>
boolean startsWith(String pre)		<i>does this string start with pre?</i>
boolean endsWith(String post)		<i>does this string end with post?</i>
int indexOf(String pattern)		<i>index of first occurrence of pattern</i>
int indexOf(String pattern, int i)		<i>index of first occurrence of pattern after i</i>
String concat(String t)		<i>this string with t appended</i>
int compareTo(String t)		<i>string comparison</i>
String toLowerCase()		<i>this string, with lowercase letters</i>
String toUpperCase()		<i>this string, with uppercase letters</i>
String replaceAll(String a, String b)		<i>this string, with as replaced by bs</i>
String[] split(String delimiter)		<i>strings between occurrences of delimiter</i>
boolean equals(Object t)		<i>is this string's value the same as t's?</i>
int hashCode()		<i>an integer hash code</i>

실습예제

다음의 실행결과와 같이 출력하는 프로그램을 작성하세요.

- 1) “aBcAbCabcABC” 문자열 String 객체를 생성한다.
- 2) 3번째 문자열 출력한다
- 3) “abc”문자열이 처음으로 나타나는 인덱스를 추적한다.
- 4) 위 문자열의 문자 개수를 출력한다.
- 5) ‘a’를 ‘R’로 대체한 결과를 출력한다.
- 6) “abc”문자열을 “123”문자열로 대체한 결과를 출력한다.
- 7) 처음 3개의 문자열만을 출력한다.
- 8) 문자열을 모두 대문자로 변경하여 출력한다.

Language API

: StringBuffer 클래스

- ▶ 가변 크기의 버퍼를 가짐
- ▶ String이 immutable인 것에 비해, StringBuffer 객체는 수정 가능하다
 - ▶ 버퍼 크기는 문자열의 길이에 따라 가변적으로 변한다

```
public class SBTest {  
    public static void main( String[] args ) {  
        StringBuffer sb = new StringBuffer("This"); //생성  
        System.out.println( sb );  
        sb.append(" is pencil"); // 문자열 덧붙이기  
        System.out.println( sb );  
        sb.insert(7, " my"); // 문자열 삽입  
        System.out.println( sb );  
        sb.replace(7, 10, " your"); // 문자열 대치  
        System.out.println( sb );  
        sb.setLength(5); // 버퍼크기 조정  
        System.out.println( sb );  
    }  
}
```

Java Basic APIs

Arrays Class

Language API

: Arrays 클래스

- ▶ 배열을 조작하는 기능을 가진 API
 - ▶ 복사
 - ▶ 항목 정렬
 - ▶ 항목 검색

리턴 타입	메소드 이름	설명	http://palpit.tistory.com
int	binarySearch(배열, 찾는값)	전체 배열 항목에서 찾는 값이 있는 인덱스 리턴	
타겟 배열	copyOf(원본배열, 복사할 길이)	원본 배열의 인덱스 0에서 복사할 길이만큼 복사한 배열 리턴	
타겟 배열	copyOfRange(원본배열, 시작, 끝 인덱스)	원본 배열의 시작과 끝 인덱스까지 복사한 배열 리턴	
boolean	deepEquals(배열, 배열)	두 배열의 깊은 비교(중첩 배열 비교)	
boolean	equals(배열, 배열)	두 배열의 얕은 비교	
void	fill(배열, 값)	전체 배열 항목에 동일한 값을 저장	
void	fill(배열, 시작, 끝 인덱스, 값)	시작부터 끝 인덱스까지의 항목에만 값을 저장	
void	sort(배열)	배열의 전체 항목을 오름차순 정렬	
String	toString(배열)	“[값1, 값2, …]” 와 같은 문자열 리턴	

Java Basic APIs

Wrapper Class

Language API

: Wrapper 클래스

- ▶ 기본 데이터형을 객체로 다루기 위한 포장 클래스



byte	short	int	long	char	float	double	boolean
Byte	Short	Int	Long	Character	Float	Double	Boolean

- ▶ 사용 이유

- ▶ 자바에서는 객체를 대상으로 처리하는 경우가 많음
- ▶ 특정 클래스는 객체만을 다루기 때문에 기본 데이터 형을 사용할 수 없음
- ▶ 문자열을 기본 타입값으로 변환할 때도 유용
- ▶ 그외, 다른 타입으로부터의 변환 등 유용한 유틸리티 메서드 제공

Language API

: Wrapper 클래스 - 객체 생성

- ▶ new를 이용한 Wrapper 클래스 객체 생성은 Deprecated 되었음
- ▶ Java 9 이후부터는 내부의 static 메서드 valueOf를 이용하여 생성

```
public class WrapperClassTest {  
    public static void main( String[] args ) {  
        Integer i = Integer.valueOf( 10 );  
        Integer i2 = new Integer(20); //Deprecated  
        Character c = Character.valueOf( 'c' );  
        Float f = Float.valueOf( 3.14f );  
        Boolean b = Boolean.valueOf( true );  
        Integer i2 = Integer.valueOf( "10" );  
        Double d2 = Double.valueOf( "3.14" );  
        Boolean b2 = Boolean.valueOf( "false" );  
    }  
}
```

Language API

: Wrapper 클래스 - 활용

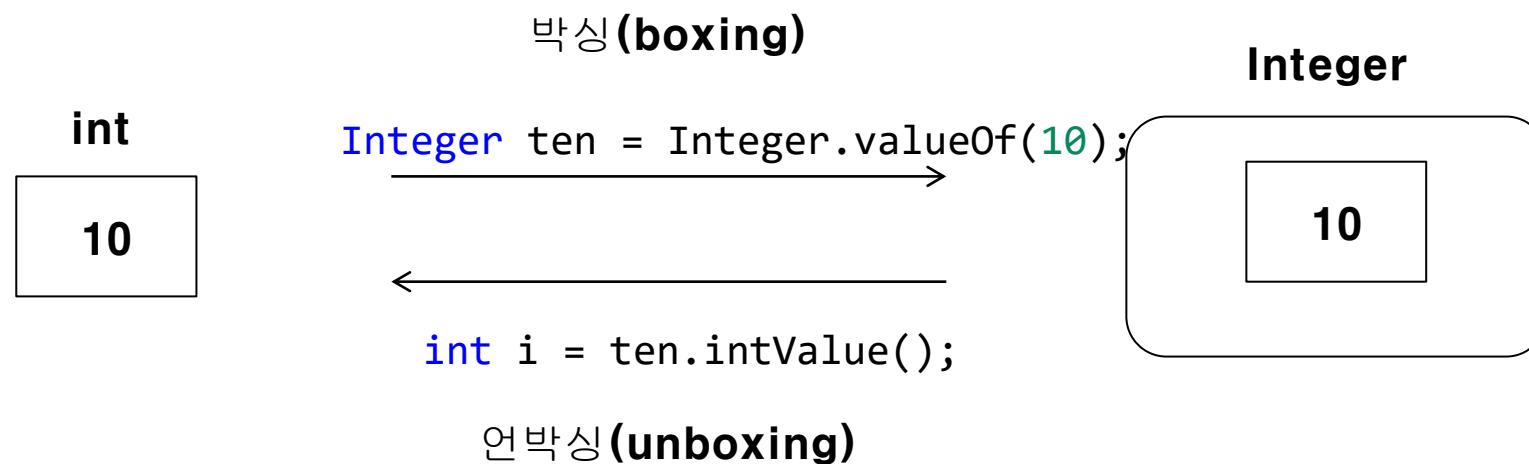
- ▶ 타입간 변환, 대부분 static 메서드
- ▶ 문자열을 기본 데이터 값으로 변환 또는 그 반대

```
public class WrapperTest {  
    public static void main( String[] args ) {  
        Integer i = Integer.valueOf( 10 );  
        Character c = Character.valueOf( '4' );  
        Double d = Double.valueOf( 3.1234566 );  
        System.out.println( Character.toLowerCase('A') ); // 대문자 A를 소문자로 변환  
        if( Character.isDigit( c ) ){ // 문자 c 가 숫자를 나타내면  
            System.out.println( Character.getNumericValue( c ) ); // 문자를 숫자로 변환  
        }  
        System.out.println( Integer.parseInt( "-123" ) ); // 문자열을 정수로 변환  
        System.out.println( Integer.parseInt( "10", 16 ) ); // 16진수 문자열을 정수로 변환  
        System.out.println( Integer.toBinaryString( 28 ) ); // 2진수로 표현된 문자열로 변환  
        System.out.println( Integer.bitCount( 28 ) ); // 2진수에서 1의 개수  
        System.out.println( Integer.toHexString( 28 ) ); // 16진수 문자열로 변환  
        System.out.println( i.doubleValue() ); // 정수를 double로 변환  
        System.out.println( d.toString() ); // Double을 문자열로 변환  
        System.out.println( Double.parseDouble("44.13e-16" ) ); // 문자열을 double로 변환  
    }  
}
```

Language API

: Wrapper 클래스 - 박싱(boxing)과 언박싱(unboxing)

- ▶ 박싱 (boxing) : 기본 데이터를 Wrapper 클래스에 담는 것
- ▶ 언박싱 (unboxing) : 박싱의 반대



Language API

: Wrapper 클래스 - 자동 박싱(auto boxing)과 자동 언박싱(auto unboxing)

- ▶ JDK 1.5부터는 박싱과 언박싱이 자동으로 수행됨

```
public class WapperClassTest {  
    public static void main( String[] args ) {  
        Integer i = 10;  
        System.out.println( i );  
        int n = i + 10;  
        System.out.println( n );  
    }  
}
```

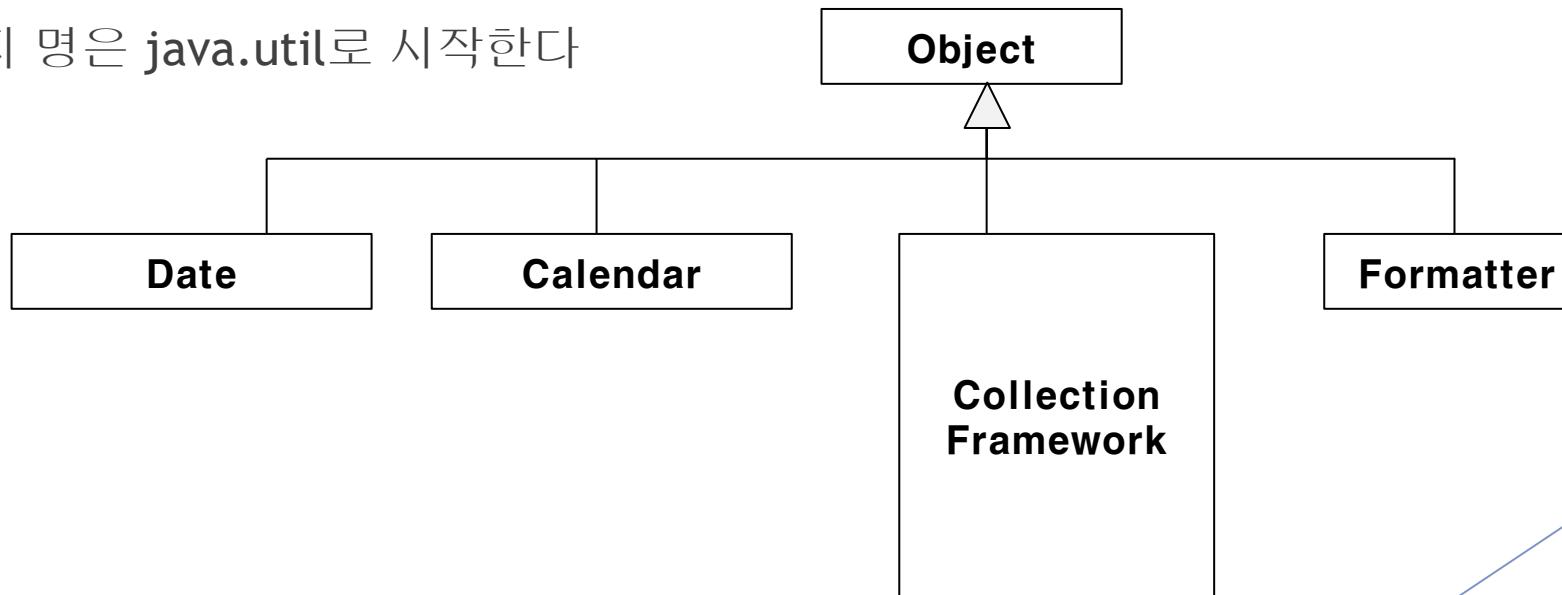
Java Basic APIs

Utility APIs

Java Utility API

: 개요

- ▶ 자바 프로그램에서 필요로 하는 편리한 기능을 모아둔 클래스들의 패키지
- ▶ 클래스 이름 앞에 패키지 이름을 사용하지 않았다면, `import`를 명시적으로 해야 하는 패키지
- ▶ 패키지 명은 `java.util`로 시작한다



Java Utility API

: Date 클래스

- ▶ 날짜와 시간에 관한 정보를 표현
- ▶ 많은 메서드들이 **deprecated** (폐지 예정)되고 **Calendar** 클래스에서 지원
 - ▶ 현재는 **Date()** 생성자만 주로 사용하고, 날짜에 관련된 기능들은 **Calendar** 클래스를 이용한다
- ▶ 만약 특정 포맷을 얻고 싶다면, **DateFormat** 클래스나 **SimpleDateFormat** 클래스를 임포트하여 사용한다

```
import java.text.DateFormat;
import java.util.Date;

public class DateTest {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Date now = new Date();
        System.out.println( now.toString() );
        System.out.println( now.toLocaleString() );      // Deprecated !!
    }
}

-----
DateFormat dateFormat1 = DateFormat.getDateInstance( DateFormat.FULL );
System.out.println( dateFormat1.format( now ) );

DateFormat dateFormat2 = DateFormat.getDateInstance( DateFormat.LONG );
System.out.println( dateFormat2.format( now ) );

DateFormat dateFormat3 =
DateFormat.getDateInstance( DateFormat.MEDIUM );
System.out.println( dateFormat3.format( now ) );

DateFormat dateFormat4 = DateFormat.getDateInstance( DateFormat.SHORT );
System.out.println( dateFormat4.format( now ) );
```

Java Utility API

: Date Format String

- ▶ `SimpleDateFormat` 클래스에 `Time Format`을 지정하여 원하는 형식의 날자 문자열을 얻을 수 있다

Format	
d	날짜의 한 자리 표현. 예) 5
dd	날짜의 두 자리 표현. 예) 05
M	월의 한 자리 표현. 예) 1
MM	월의 두 자리 표현. 예) 01
MMM	월의 세 자리 영어 표현. 예) Jan
MMMM	월의 전체 영어 표현. 예) January
yy	연도의 두 자리 표현. 예) 17
yyyy	연도의 네 자리 표현. 예) 2017

Format	
h	12시간 포맷의 한 자리 시간 표현. 예) 9
hh	12시간 포맷의 두 자리 시간 표현. 예) 09
H	24시간 포맷의 두 자리 시간 표현. 예) 09 (AM 9)
HH	24시간 포맷의 두 자리 시간 표현. 예) 21 (PM 9)
m	한 자리 분 표현. 예) 9
mm	두 자리 분 표현. 예) 09
s	한 자리 초 표현. 예) 3
ss	두 자리 초 표현. 예) 03
a	am/pm 표현

Java Utility API

: Calendar 클래스

- ▶ 날짜와 시간에 관한 정보를 표현
- ▶ `new`로 객체를 생성할 수 없고, `getInstance()` 메서드를 이용, `Calendar` 객체를 얻어 사용
- ▶ 날짜와 시간 표현을 위한 다양한 상수 제공

Java Utility API

: Calendar

▶ Calendar 클래스의 주요 상수

상수	사용방법	설명
static int YEAR	Calendar.YEAR	현재 년도를 가져온다.
static int MONTH	Calendar.MONTH	현재 월을 가져온다. (1월은 0)
static int DATE	Calendar.DATE	현재 월의 날짜를 가져온다.
static int WEEK_OF_YEAR	Calendar.WEEK_OF_YEAR	현재 년도의 몇째 주
static int WEEK_OF_MONTH	Calendar.WEEK_OF_MONTH	현재 월의 몇째 주
static int DAY_OF_YEAR	Calendar.DAY_OF_YEAR	현재 년도의 날짜
static int DAY_OF_MONTH	Calendar.DAY_OF_MONTH	현재 월의 날짜 (DATE와 동일)
static int DAY_OF_WEEK	Calendar.DAY_OF_WEEK	현재 요일 (일요일은 1, 토요일은 7)
static int HOUR	Calendar.HOUR	현재 시간 (12시간제)
static int HOUR_OF_DAY	Calendar.HOUR_OF_DAY	현재 시간 (24시간제)
static int MINUTE	Calendar.MINUTE	현재 분
static int SECOND	Calendar.SECOND	현재 초

Java Utility API

: Calendar

▶ Calendar 클래스 주요 메서드

Calendar 클래스의 메소드	설명
boolean after(Object when)	when과 비교하여 현재 날짜 이후이면 true, 아니면 false를 반환한다.
boolean before(Object when)	when과 비교하여 현재 날짜 이전이면 true, 아니면 false를 반환한다.
boolean equals(Object obj)	같은 날짜값인지 비교하여 true, false를 반환한다.
int get(int field)	현재 객체의 주어진 값의 필드에 해당하는 상수 값을 반환한다. 이 상수값은 Calendar 클래스의 상수중에 가진다.
static Calendar getInstance()	현재 날짜와 시간 정보를 가진 Calendar 객체를 생성한다.
Date getTime()	현재의 객체를 Date 객체로 변환한다.
long getTimelnMills()	객체의 시간을 1/1000초 단위로 변경하여 반환한다.
void set(int field, int value)	현재 객체의 특정 필드를 다른 값으로 설정한다.
void set(int year, int month, int date)	현재 객체의 년, 월, 일 값을 다른 값으로 설정한다.
void set(int year, int month, int date, int hour, int minute, int second)	현재 객체의 년, 월, 일, 시, 분, 초 값을 다른 값으로 설정한다.
void setTime(Date date)	date 객체의 날짜와 시간 정보를 현재 객체로 생성한다.
void setTimeInMills(long mills)	현재 객체를 1/1000초 단위의 주어진 매개변수 시간으로 설정한다.
int getActualMinimum(int field)	현재 객체의 특정 필드의 최소 값을 반환한다.
int getActualMaximum(int field)	현재 객체의 특정 필드의 최대 값을 반환한다.

Java Utility API

: Calendar 클래스 - 활용 I

```
import java.util.Calendar;
public class CalendarTest {
    public static void main(String[] args) {
        Calendar caledar = Calendar.getInstance(); // 지금!
        printDate(caledar);
        caledar.set(Calendar.YEAR, 2016); // 년도를 2016년으로 설정
        printDate(caledar); // 년,월,일을 설정함(2016-2-12), 월은 0부터 시작하므로 -1을 해줘야한다.
        caledar.set(2016, 1, 12);
        printDate(caledar);
        caledar.set(2016, 1, 12, 13, 59); // 년,월,일,시,분을 설정(2016-2-12 오후 1:59)
        printDate(caledar);
        caledar.set(2016, 5, 12, 13, 59, 30); // 년,월,일,시,분,초를 설정(2016-6-12 오후 1:59:30)
        printDate(caledar);
    }

    public static void printDate(Calendar caledar) {
        System.out.println(caledar.get(Calendar.YEAR) + "년 "
            + (caledar.get(Calendar.MONTH) + 1) + "월 " // 0부터 시작함
            + caledar.get(Calendar.DATE) + "일 "
            + (caledar.get(Calendar.AM_PM) == 0 ? "오전 " : "오후 ") // 오전:0, 오후:1
            + caledar.get(Calendar.HOUR) + "시 "
            + caledar.get(Calendar.MINUTE) + "분 "
            + caledar.get(Calendar.SECOND) + "초");
    }
}
```

Java Utility API

: Calendar 클래스 - 활용 II

```
import java.util.Calendar;

public class CalendarTest {

    public static void main(String[] args) {
        // 오늘
        Calendar caledar = Calendar.getInstance();
        printDate(caledar);
        // 100일 후
        caledar.add(Calendar.DATE, 100);
        printDate(caledar);
        // 다시 오늘
        caledar = Calendar.getInstance();
        printDate(caledar);
        // 10달전
        caledar.add(Calendar.MONTH, -10);
        printDate(caledar);
        // 다시 오늘
        caledar = Calendar.getInstance();
        printDate(caledar);
        // 오늘은 올해 몇 일째 되는날?
        System.out.println("오늘은 올해 " + caledar.get(Calendar.DAY_OF_YEAR) + "일째 되는 날입니다.");
    }
}
```

Java Utility API

: Formatter 클래스

- ▶ 데이터의 형식화된 문자열을 생성하는 클래스
- ▶ 형식 문자열 (format 지정)
- ▶ 문자열 생성을 위하여 Appendable 인터페이스를 구현한 클래스 객체 (StringBuffer 객체)를 Formatter 생성자에서 등록한다

```
StringBuffer sb = new StringBuffer();
Formatter f = new Formatter( sb );
String name = "Simon Readmore";
int score = 100;
f.format( "%s님의 점수는 %d 입니다.", name, score );
```

```
System.out.println(f);
```

```
String s = String.format("%s님의 점수는 %d 입니다.", name, score );
```

Java Basic APIs

Generic

제네릭(Generic)

- ▶ 클래스 내부에서 사용할 데이터 타입을 외부에서 지정하는 기법
(클래스 내부에서 사용할 데이터 타입을 나중에 인스턴스를 생성할 때 확정하여 사용)
- ▶ 제네릭을 사용해야 하는 이유
 - ▶ 객체의 타입을 컴파일시에 강하게 체크할 수 있음
 - ▶ 형변환의 번거로움이 줄어든다

개별 타입만 다룰 수 있는 클래스

PointList.java, CircleList.java

```
public class PointList {  
    private Point[] pArray;  
    private int crtPos;  
    public PointList() {  
        this.pArray = new Point[3];  
        this.crtPos = 0;  
    }  
    public void add(Point o) {  
        pArray[crtPos] = o;  
        crtPos++;  
    }  
}
```

모든 타입을 다룰 수 있는 클래스

ObjList.java

```
public class ObjList {  
    private Object[] pArray;  
    private int crtPos;  
    public ObjList() {  
        this.pArray = new Object[3];  
        this.crtPos = 0;  
    }  
    public void add(Object o) {  
        pArray[crtPos] = o;  
        crtPos++;  
    }  
}
```

사용 타입을 나중에 확정
제네릭 클래스

GenericList.java

```
public class GenericList<T> {  
    private T[] pArray;  
    private int crtPos;  
    public GenericList() {  
        this.pArray = new T[3];  
        this.crtPos = 0;  
    }  
    public void add(T o) {  
        pArray[crtPos] = o;  
        crtPos++;  
    }  
}
```

제네릭(Generic)

- ▶ 제네릭 타입 : 타입을 파라미터로 가지는 클래스와 인터페이스

```
public class 클래스명<T> { ... }  
public interface 인터페이스명<T> { //... }
```

GenericBox.java

```
public class GenericBox<T> {  
    T content;    설계시에는 내부에서 사용할 타입을 명시하지 않음  
  
    public T getContent() {  
        return content;  
    }  
  
    public void setContent(T content) {  
        this.content = content;  
    }  
}  
타입 파라미터의 개수는 제한하지 않음
```

타입을 객체 생성시 결정

```
Box<Integer> intBox = new Box<Integer>();  
Box<String> strBox = new Box<String>();
```

Java Basic APIs

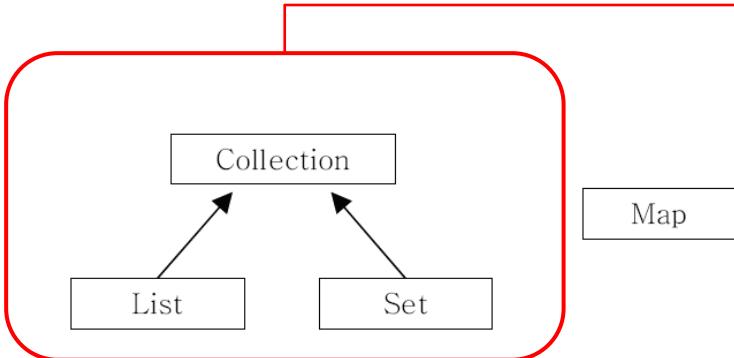
Collection Framework

Collection Framework

- ▶ 컬렉션 (Collection)
 - ▶ 다수의 데이터, 즉 데이터 그룹을 의미
- ▶ 프레임워크 (Framework)
 - ▶ 표준화, 정형화된 체계적인 프로그래밍 방식
- ▶ 컬렉션 프레임워크
 - ▶ 컬렉션을 저장하는 클래스들을 표준화한 설계
 - ▶ 다수의 데이터를 쉽게 처리할 수 있는 방법을 제공하는 클래스들로 구성
 - ▶ JDK 1.2부터
- ▶ 컬렉션 클래스 (Collection Class)
 - ▶ 다수의 데이터를 저장할 수 있는 클래스
 - ▶ Vector, ArrayList, HashSet 등

Collection Framework

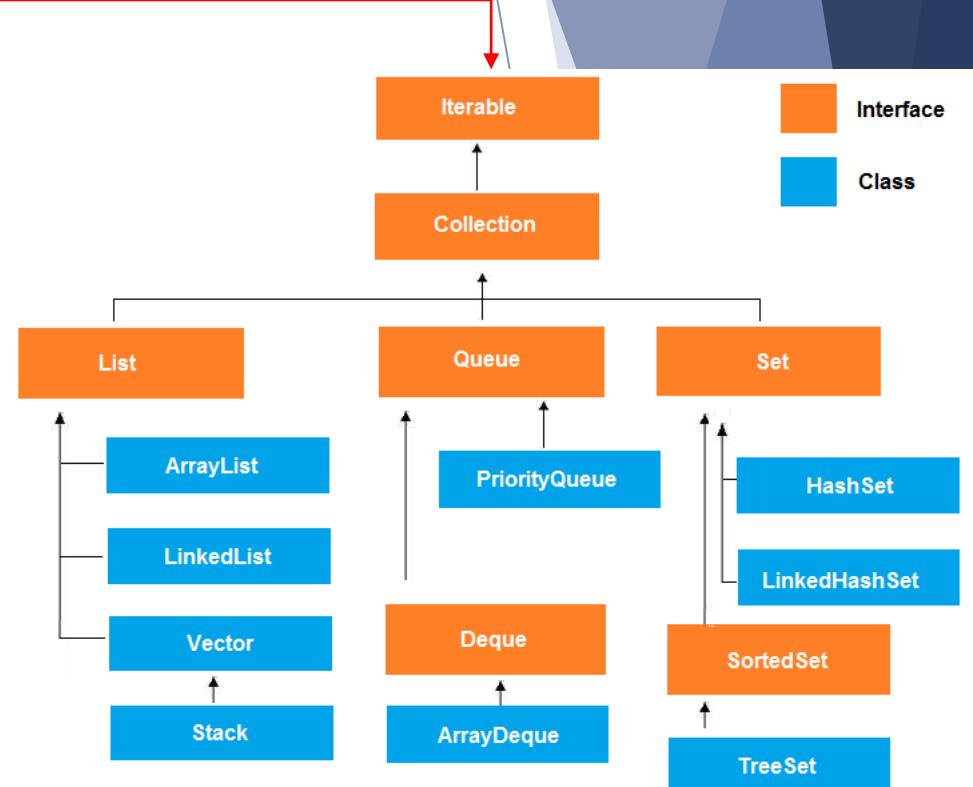
: 핵심 인터페이스



[그림11-1] 컬렉션 프레임워크의 핵심 인터페이스간의 상속계층도

인터페이스	특 징
List	순서가 있는 데이터의 집합. 데이터의 중복을 허용한다. 예) 대기자 명단 구현클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	순서를 유지하지 않는 데이터의 집합. 데이터의 중복을 허용하지 않는다. 예) 양의 정수집합, 소수의 집합 구현클래스 : HashSet, TreeSet 등
Map	키(key)와 값(value)의 쌍(pair)으로 이루어진 데이터의 집합 순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용한다. 예) 우편번호, 지역번호(전화번호) 구현클래스 : HashMap, TreeMap, Hashtable, Properties 등

[표11-1] 컬렉션 프레임워크의 핵심 인터페이스와 특징



Collection Framework

: 배열 (Array) vs 리스트 (List)

	배열(Array)	리스트(List)
장점	<ul style="list-style-type: none">▪빠른 접근 (faster access)▪기본 자료유형 사용 가능	<ul style="list-style-type: none">▪가변적인 크기▪필요시 메모리를 할당하므로 효율적
단점	<ul style="list-style-type: none">▪고정된 크기▪비효율적 메모리 점유▪최대 크기를 넘어서는 사용을 위해서는 배열을 새로 정의해야 함	<ul style="list-style-type: none">▪느린 접근 (slower access)▪참조자료 유형만 사용 가능
사용	<ul style="list-style-type: none">▪Java에서 자체적으로 지원	<ul style="list-style-type: none">▪<code>java.util.List</code> 인터페이스 정의

Collection Framework

: Vector

- ▶ **Vector** 클래스 메서드 (Method)
 - ▶ 생성자 : `Vector()` / `Vector(int capacity)`
 - ▶ 개체 추가 : `addElement(Object o)` / `insertElementAt(Object o, int index)`
 - ▶ 개체 참조 : `elementAt(int index)`
 - ▶ 역개체참조 : `indexOf(Object o)`
 - ▶ 개체 변경 : `setElementAt(Object o, int index)`
 - ▶ 개체 삭제 : `remove(Object o)` / `remove(int index);`
 - ▶ 객체 검색 : `contains(Object o)`
 - ▶ 크기와 용량 : `size()` / `capacity()`
 - ▶ 비우기 : `clear()`
 - ▶ 복사 : `clone()`

Collection Framework

: Vector Example

```
public class VectorTest {  
    public static void main(String[] args) {  
        Vector v = new Vector();  
        System.out.println("Size=" + v.size() + " Capacity=" + v.capacity());  
  
        for( int i=0; i<10; i++ ) {  
            v.addElement(Integer.valueOf(i));  
        }  
        System.out.println("Size=" + v.size() + " Capacity=" + v.capacity());  
  
        Integer i0 = (Integer)v.elementAt(0);  
        int i1 = (Integer)v.elementAt(9);  
        System.out.println(i1);  
  
        v.removeElement(i0);  
        v.removeElement(1);  
        System.out.println(v);  
    }  
}
```

Collection Framework

: Vector Class

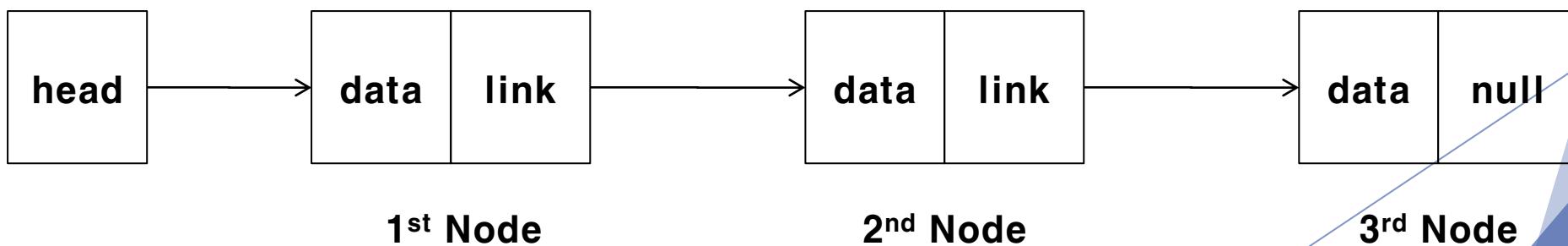
- ▶ 객체만 저장할 수 있음
- ▶ Vector의 참조 결과는 항상 Object 타입 -> 적절한 Type으로 변환 후 사용
- ▶ Generics로 지정하지 않으면 여러 타입을 동시에 저장 가능
 - ▶ 꺼낸 객체의 타입을 알고 있어야 함
 - ▶ 혹은 instanceof 연산자로 확인 후 사용

```
Vector v = new Vector();
v.addElement(1); //1은 int 기본자료형이지만 자바 버전에 따라 Packing되어 삽입된다
v.addElement(Integer.valueOf(1)); //int에 대응되는 Wrapper 클래스인 Integer 사용
v.addElement(Double.valueOf(3.14)); //어떤 타입의 객체도 저장 가능
Integer i = v.elementAt(0); //elementAt 메소드의 반환타입은 Object 이므로 컴파일 안됨
Integer i = (Integer)v.elementAt(0); //명시적 내림변환 필요(downcasting)
Double d = (Double)v.elementAt(1);
```

Collection Framework

: Linked List

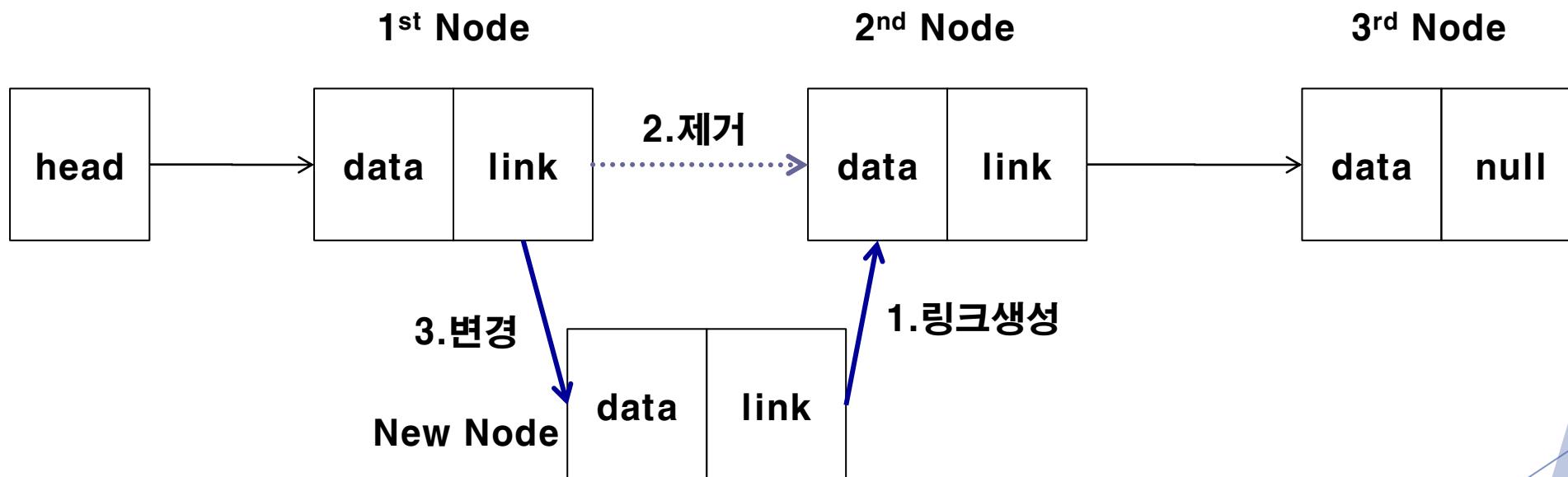
- ▶ 링크(Link)로 연결된 노드(Node)의 집합
- ▶ `java.util.LinkedList`
- ▶ 임의의 객체를 리스트로 저장하는 기능을 제공
- ▶ `Index`를 통한 참조 접근은 불가
 - ▶ `Head`로부터 링크를 따라가면서 접근
- ▶ 각 노드는 자신이 나타내는 데이터와 다음 노드(Node)로의 링크(Link)를 가지고 있음



Collection Framework

: Linked List

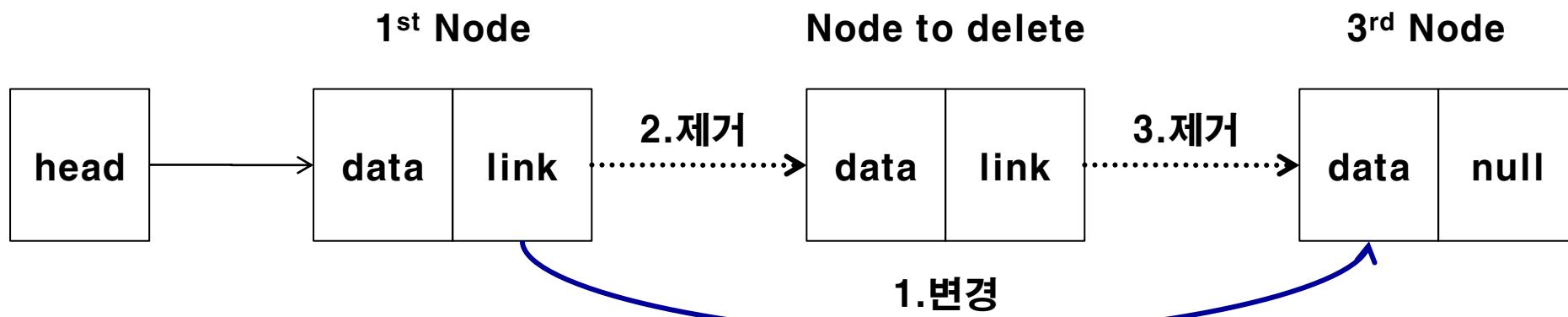
- ▶ 새로운 노드 추가



Collection Framework

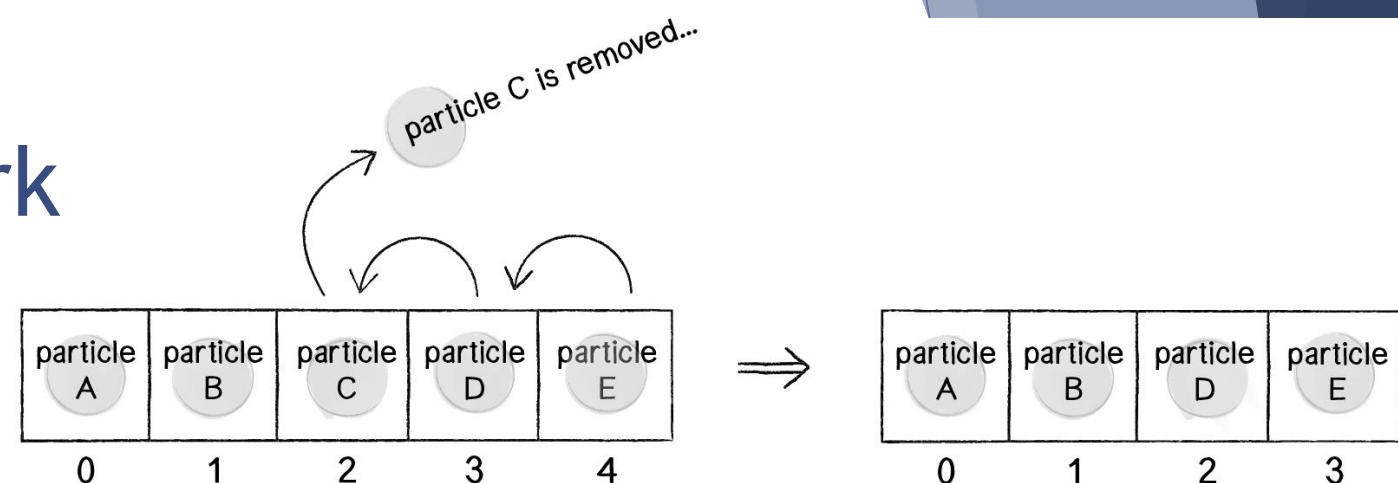
: Linked List

- ▶ 기존 노드 제거



Collection Framework

: Array List

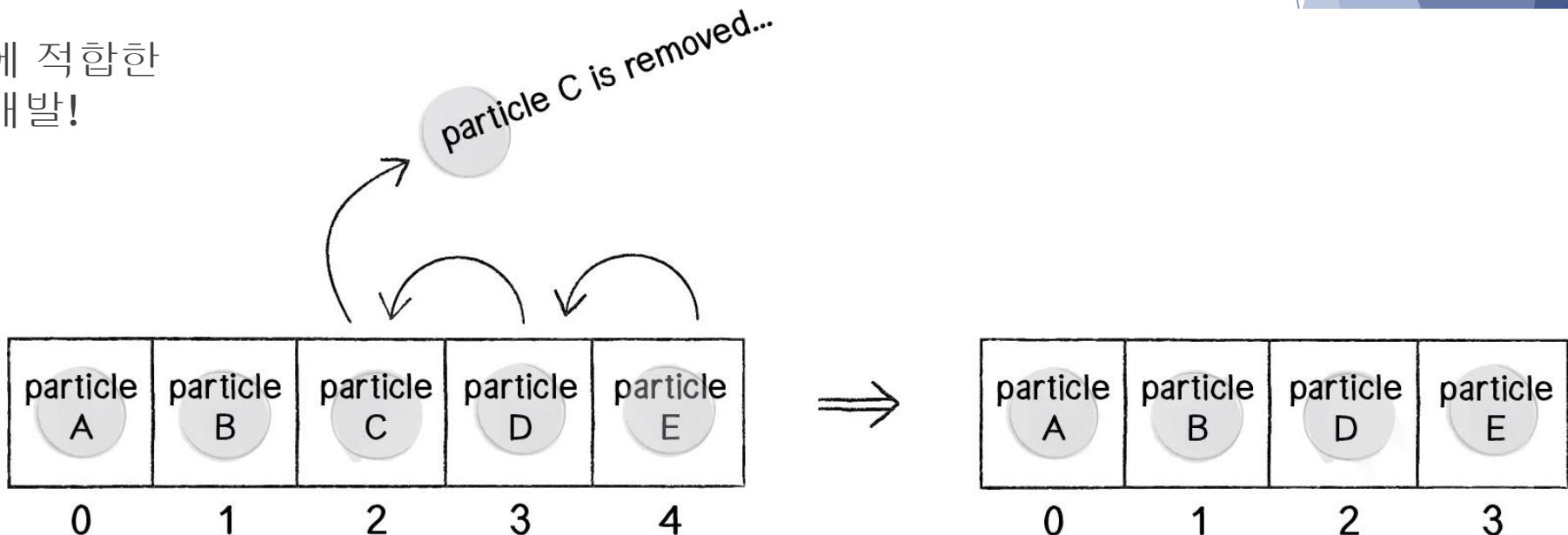


- ▶ java.util.ArrayList에 구현
- ▶ Vector, LinkedList, ArrayList 모두 추상 클래스 **AbstractList**를 상속받은 동적 자료 구조
- ▶ 배열을 다루는 것과 유사한 방식으로 동적 자료 구조를 사용할 수 있게 함
 - ▶ add(int index, E element), set(int index, E element), get(int index), remove(int index) methods
- ▶ **Vector vs List** : 항목 삽입, 삭제 동작이 동기화(synchronization) 여부의 차이
 - ▶ Vector : 동기화된 삽입 삭제 동작 제공
 - ▶ List : 삽입과 삭제가 동기화 되어 있지 않음 - 외부적 동기화 필요

Collection Framework

: ArrayList

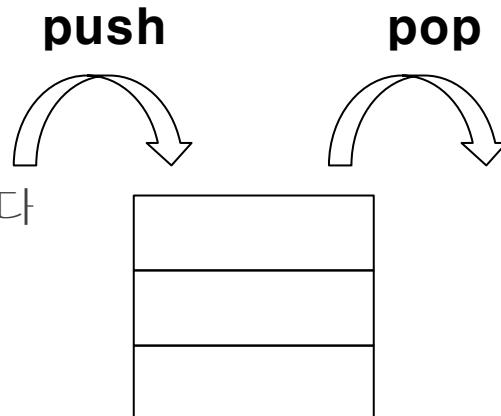
- ▶ ArrayList는 LinkedList와 동작 방식은 같으나 내부 구현 방식이 다르다
- ▶ ArrayList는 중간에 객체가 삭제되면 뒤의 객체들을 당겨, 인덱스를 재구성한다.
 - ▶ 따라서 마지막 인덱스에 객체가 추가되는 속도는 LinkedList보다 빠르지만 중간에 객체의 추가, 삭제가 빈번하게 일어나는 경우는 속도 저하가 일어나게 된다.
- ▶ 하고자 하는 작업에 적합한 자료구조를 택해 개발!



Collection Framework

: Stack

- ▶ `java.util.Stack` 클래스로 제공, `Vector` 클래스를 상속받아 구현
- ▶ 한쪽 끝점에서만 새로운 항목을 삽입, 기존 항목을 제거할 수 있음
 - ▶ Last In First Out (LIFO)
 - ▶ 쌓여있는 접시 : 새로운 접시를 위에 쌓거나 가장 위의 접시부터 사용 가능
- ▶ 스택에서의 메서드들
 - ▶ `push()` - 스택에 객체를 넣음
 - ▶ `pop()` - 스택에서 객체를 추출(`top`은 삭제)
 - ▶ `peek()` - `pop`과 같지만, `top` 값을 삭제하지 않는다
 - ▶ `empty()` - 스택이 비었는지 확인



Collection Framework

: Stack Example

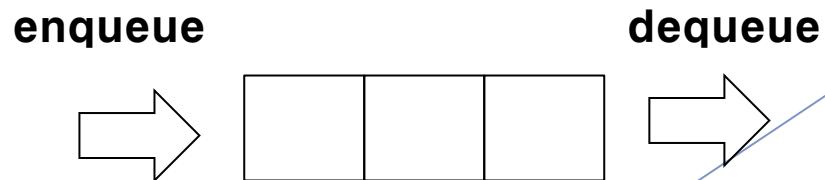
```
import java.util.Stack;
public class StackTest {
    public static void main(String[] args) {
        Stack s = new Stack();
        s.push("A");
        s.push("B");
        s.push("C");

        System.out.println(s.empty());
        System.out.println(s.pop());
        System.out.println(s.peek());
        System.out.println(s.pop());
    }
}
```

Collection Framework

: Queue

- ▶ java.util.Queue 클래스로 제공 (interface)
- ▶ 목록의 가장 마지막에 새로운 항목이 추가
- ▶ 기존 항목의 제거는 리스트의 처음에서 일어남
- ▶ First In First Out (FIFO)
- ▶ 큐에서의 메서드들
 - ▶ offer()
 - ▶ poll()
 - ▶ peek()



Collection Framework

: Queue Example

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueTest {
    public static void main(String[] args) {
        Queue q = new LinkedList();

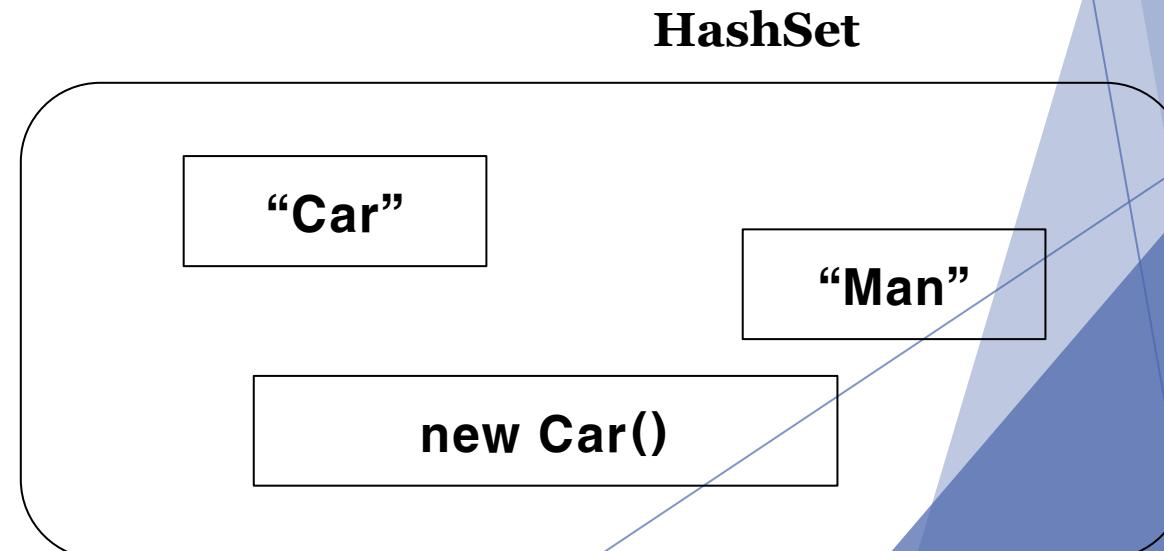
        q.offer("A");
        q.offer("B");
        q.offer("C");

        System.out.println(q.poll());
        System.out.println(q.peek());
        System.out.println(q.poll());
    }
}
```

Collection Framework

: Hash Set

- ▶ `java.util.HashSet` 클래스로 제공
- ▶ 자료 구조에 포함된 자료의 순서나 키에 상관없이 자료 전체를 하나의 셋으로 관리할 수 있도록 해주는 자료 구조
- ▶ 해시테이블에서 키 없이 값들만 존재하는 경우
- ▶ 자료의 해시로 유지, 검색이 빠르다
- ▶ `HashSet`에서 사용 가능한 메서드들
 - ▶ `add`
 - ▶ `remove`
 - ▶ `contains`



Collection Framework

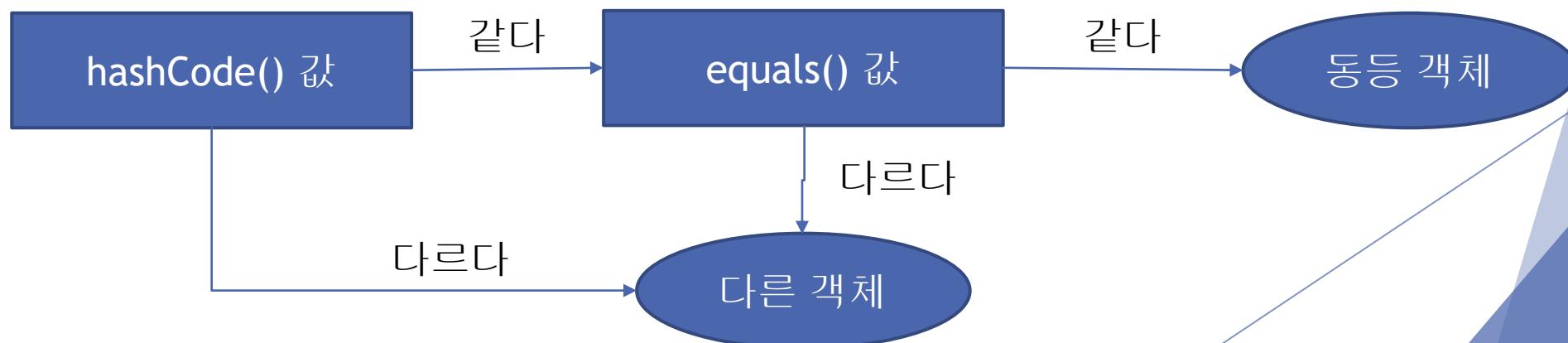
: Hash Set Example

```
public class HashSetTest {  
    public static void main(String[] args) {  
        HashSet hs = new HashSet();  
        hs.add("Car"); hs.add("Bus"); hs.add("Truck");  
        hs.add("Man"); hs.add("Woman"); hs.add("Child");  
        System.out.println(hs.size());  
        if( hs.contains("Man") ) {  
            hs.remove("Man");  
        }  
        if( hs.contains("Car") ) {  
            hs.add("Car2");  
        }  
        System.out.println(hs);  
    }  
}
```

Collection Framework

: hash 알고리즘과 hashCode()

- ▶ 객체 해시 코드란 객체를 식별할 하나의 정수값을 의미
- ▶ Object의 hashCode() 메서드는 객체의 메모리 번지를 이용하여 해시 코드를 만들어 리턴 -> 모든 객체는 다른 값을 가지고 있음
- ▶ Hash 관련 컬렉션들은 다음과 같은 방식으로 두 객체가 동등한지 아닌지를 비교 한다
 - ▶ equals와 hashCode 메서드를 이용, hash 알고리즘을 구현해 주어야 한다



Collection Framework

: Hash Table

- ▶ java.util.Hashtable 클래스
- ▶ 맵(Map) 인터페이스를 다루는 자료 구조
- ▶ 자료들의 순서가 아닌 키(key)와 값(value)의 쌍을 저장
- ▶ 키와 값은 모든 임의의 객체가 허용됨
- ▶ Hashtable에서 사용 가능한 메서드들
 - ▶ put
 - ▶ get
 - ▶ isEmpty
 - ▶ containsKey

key	value
“Car”	new Car()
“Man”	new Man()
“Cat”	new Cat()

Collection Framework

: Hash Table Example

```
public class HashTableTest {  
    public static void main(String[] args) {  
        Hashtable ht = new Hashtable();  
  
        Vector v1 = new Vector();  
        Vector v2 = new Vector();  
        v1.add("Taxi"); v1.add("Bus"); v1.add("Truck");  
        v2.add("Man"); v2.add("Woman"); v2.add("Child");  
        ht.put("Car",v1);  
        ht.put("Person",v2);  
        System.out.println(ht.get("Car"));  
  
        if( ht.containsKey("Person") ) {  
            System.out.println(ht.get("Person"));  
        }  
    }  
}
```

Collection Framework

: Enumeration and Iterator

- ▶ Enumeration : 벡터와 해시테이블에 존재하는 요소들에 대한 접근방식을 제공해주는 인터페이스
- ▶ Iterator: Collection Framework로 확장하면서 도입 (List, Set 등)
- ▶ 자바에서 제공하는 컬렉션에 대해 각 컬렉션의 항목들을 **순차적으로 접근**하는데 사용
- ▶ Enumeration
 - ▶ hasMoreElements()와 nextElement() 두 개의 메서드 제공
 - ▶ Vector::elements()
 - ▶ Hashtable::keys()
 - ▶ Hashtable::values()
- ▶ Iterator
 - ▶ hasNext(), next(), **remove()** 제공
 - ▶ Set::iterator()
 - ▶ List::iterator()

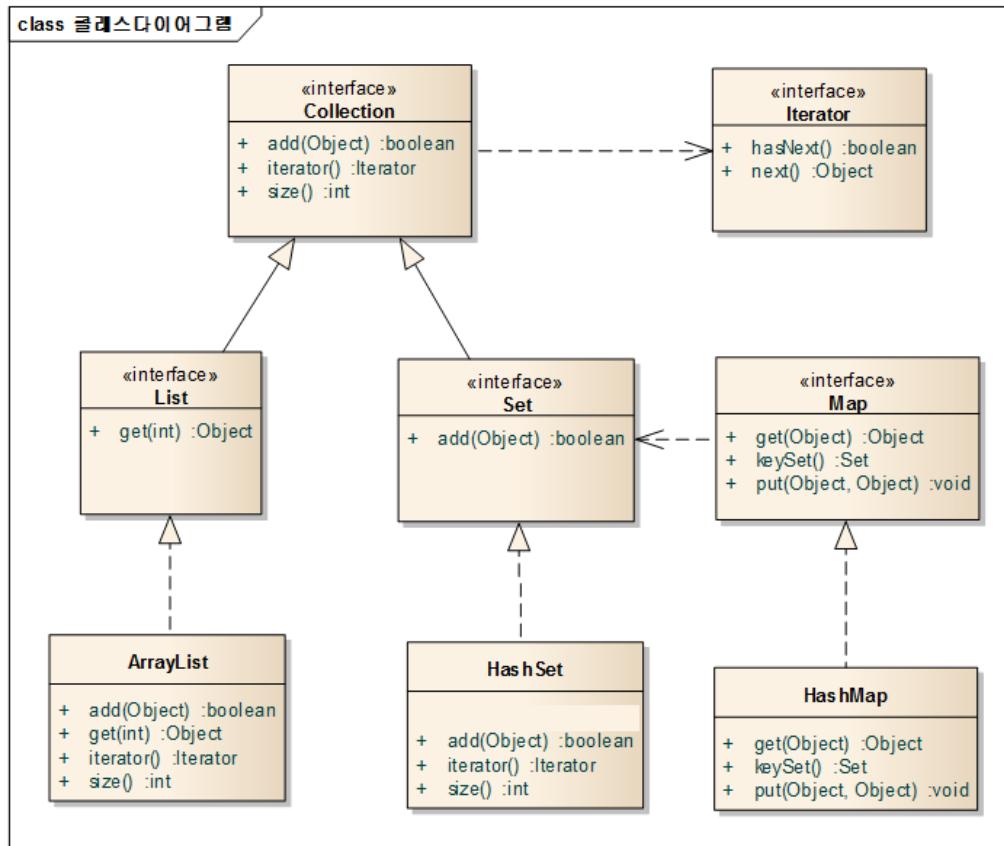
Collection Framework

: Enumeration and Iterator - Example

```
Vector v = new Vector();
// ...
Enumeration e = v.elements();
while(e.hasMoreElements()) {
    System.out.println((String)e.nextElement());
}
HashSet set = new HashSet();
// ...
Iterator iter = set.iterator();
while (iter.hasNext()) {
    i++;
    System.out.println(i + ":" + iter.next());
}
```

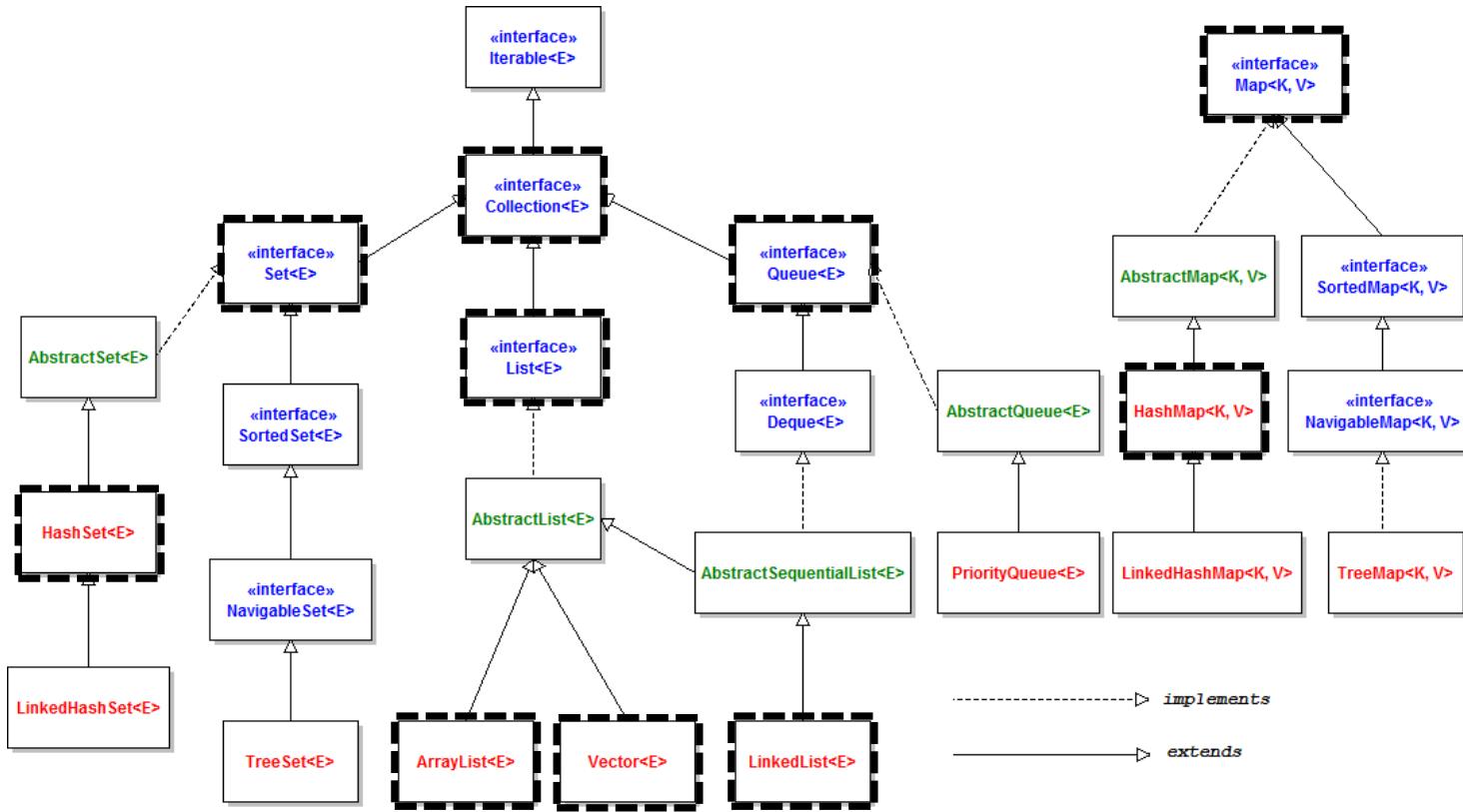
Collection Framework

: Collection Class Diagram I



Collection Framework

: Collection Class Diagram II



Java I/O Programming

Stream and I/O

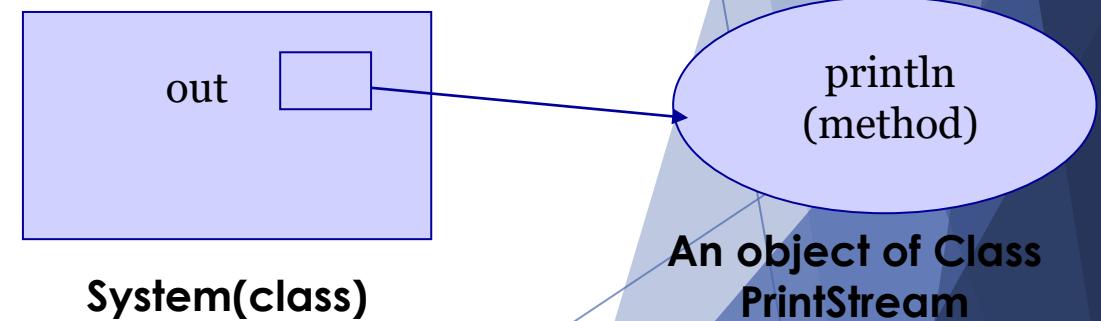
Stream and I/O

: I/O

▶ I/O (Input and Output: 입출력)

- ▶ 프로그램과 외부 소스 혹은 목적지와의 데이터 및 정보의 교환
- ▶ 입력 : 키보드, 파일 등으로부터 프로그램으로 들어오는 데이터
- ▶ 출력 : 화면, 파일 등으로 프로그램으로부터 나가는 데이터

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

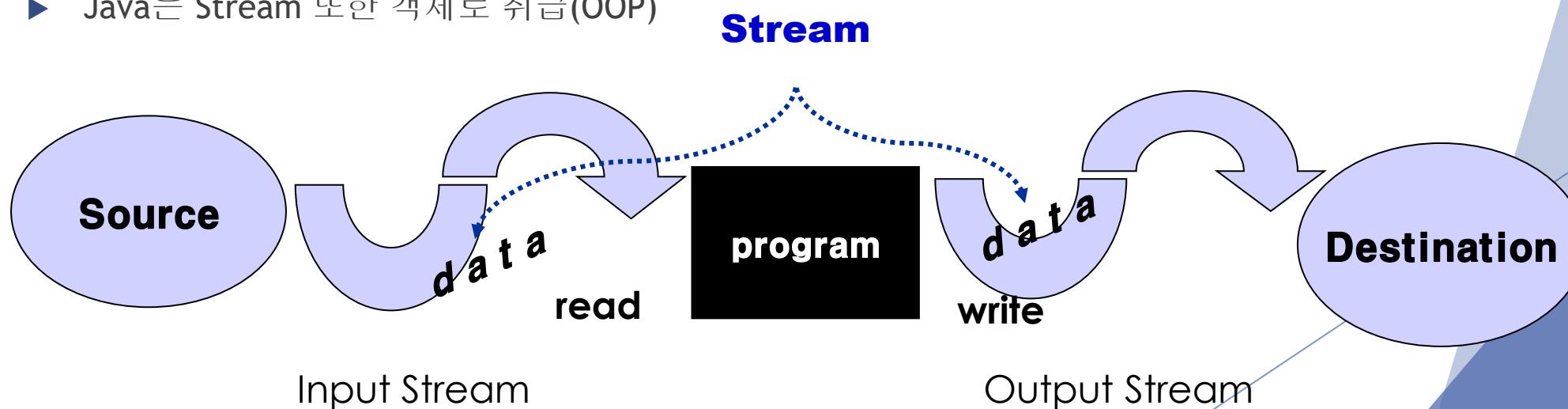


- `System.out` - PrintStream 객체를 참조하는 System Class의 static 멤버
- `System.in` - InputStream 객체를 참조하는 System Class의 static 멤버

Stream and I/O

: Stream

- ▶ 프로그램과 I/O 객체를 연결, 데이터를 소스에서 전달하거나 목적지로 수신하는 길 (Path)
 - ▶ 목적지/소스: 키보드 및 스크린을 포함한 파일, Network Connections, 다른 프로그램 등
- ▶ 일방향 (Unidirectional)으로 프로그램과 I/O 객체를 연결
 - ▶ InputStream - 데이터를 읽어들이는 객체 ex) System.in
 - ▶ OutputStream - 데이터를 써서 내보내는 객체 ex) System.out
- ▶ Java는 Stream 또한 객체로 취급(OOP)



Stream and I/O

: Byte Stream vs Character Stream

- ▶ 바이트 스트림 (Byte Stream)
 - ▶ 1과 0으로 구성된 바이너리 데이터의 입출력 처리 ex) 이미지, 사운드 등
- ▶ 문자 스트림 (Character Stream)
 - ▶ 문자, 텍스트 형태의 데이터 입출력 처리를 위한 스트림 ex) 텍스트, 웹페이지, 키보드 입력 등
- ▶ Java I/O 클래스
 - ▶ Java.io 패키지에 I/O를 위한 4개의 **추상 클래스** 정의
 - ▶ 해당 클래스에서 상속받아 파일에 데이터를 읽고 씀

InputStream	Reader
abstract int read() int read(byte[] b) int read(byte[] b, int off, int len)	int read() int read(char[] cbuf) abstract int read(char[] cbuf, int off, int len)
OutputStream	Writer
abstract void write(int b) void write(byte[] b) void write(byte[] b, int off, int len)	void write(int c) void write(char[] cbuf) abstract void write(char[] cbuf, int off, int len) void write(String str) void write(String str, int off, int len)

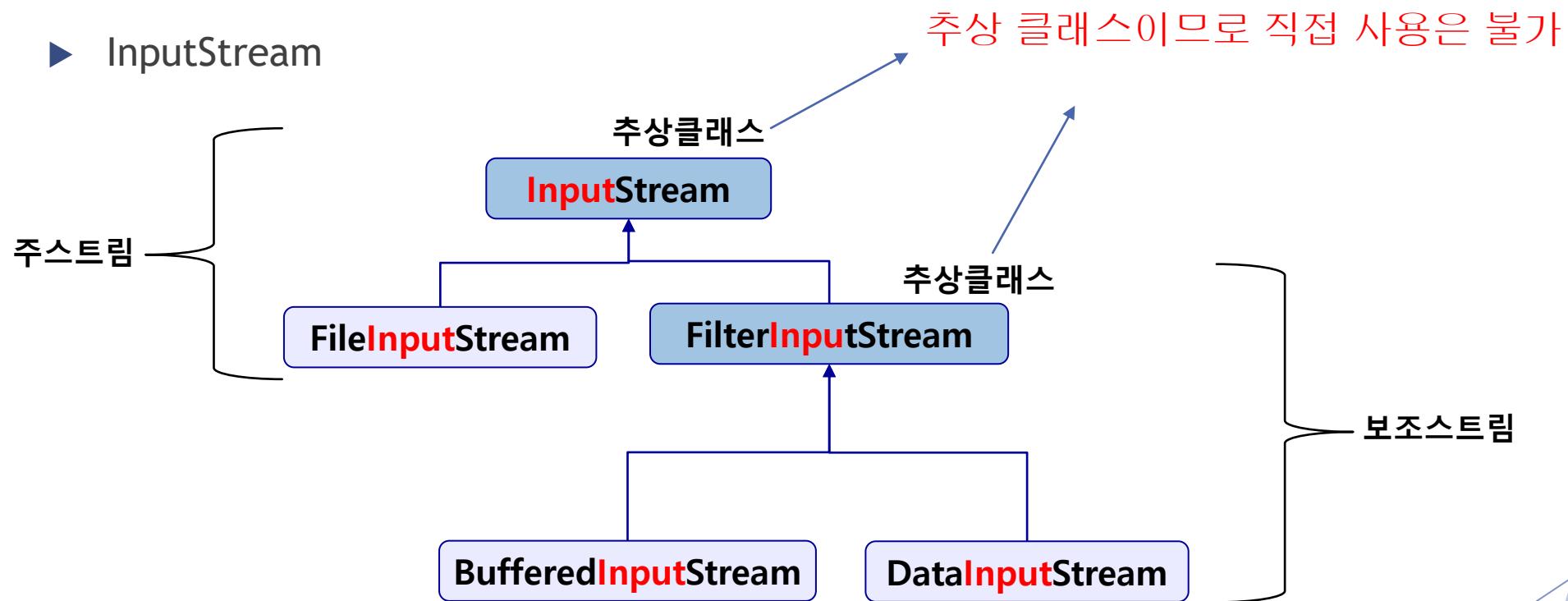
바이트 스트림용

문자 스트림용

Stream and I/O

: Byte Stream

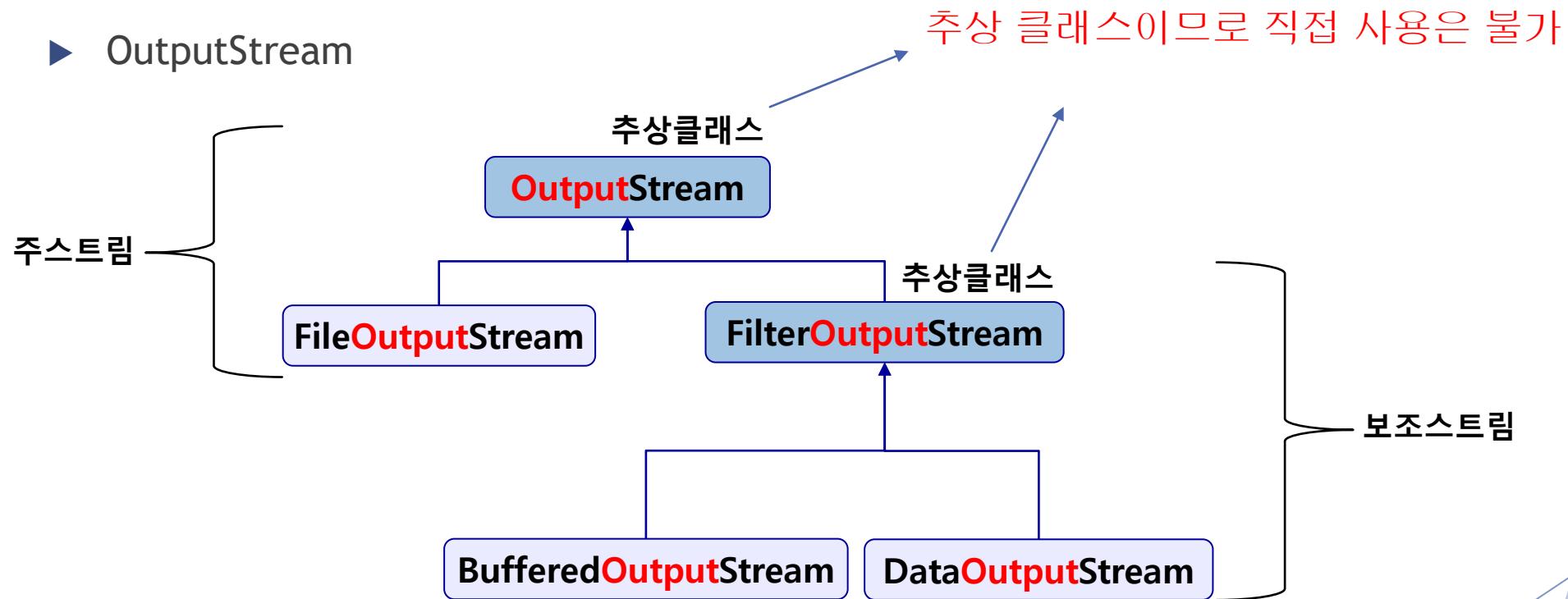
▶ InputStream



Stream and I/O

: Byte Stream

▶ OutputStream



Stream and I/O

: Byte Stream

▶ InputStream의 메서드

메서드명	설 명
int available()	스트림으로부터 읽어 올 수 있는 데이터의 크기를 반환한다.
void close()	스트림을 닫음으로써 사용하고 있던 자원을 반환한다.
void mark(int readlimit)	현재위치를 표시해 놓는다. 후에 reset()에 의해서 표시해 놓은 위치로 다시 돌아갈 수 있다. readlimit은 되돌아갈 수 있는 byte의 수이다.
boolean markSupported()	mark()와 reset()을 지원하는지를 알려 준다. mark()와 reset()기능을 지원하는 것은 선택적이므로, mark()와 reset()을 사용하기 전에 markSupported()를 호출해서 지원 여부를 확인해야한다.
abstract int read()	1 byte를 읽어 온다(0~255사이의 값). 더 이상 읽어 올 데이터가 없으면 -1을 반환한다. abstract에서 드러나 InputStream의 자손들은 자신의 상황에 알맞게 구현해야한다.
int read(byte[] b)	배열 b의 크기만큼 읽어서 배열을 채우고 읽어 온 데이터의 수를 반환한다. 반환하는 값은 항상 배열의 크기보다 작거나 같다.
int read(byte[] b, int off, int len)	최대 len개의 byte를 읽어서, 배열 b의 지정된 위치(off)부터 저장한다. 실제로 읽어 올 수 있는 데이터가 len개보다 적을 수 있다.
void reset()	스트림에서의 위치를 마지막으로 mark()이 호출되었던 위치로 되돌린다.
long skip(long n)	스트림에서 주어진 길이(n)만큼을 건너뛴다.

▶ OutputStream의 메서드

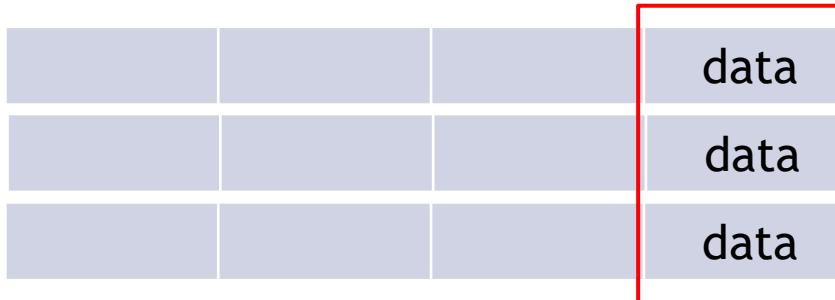
메서드명	설 명
void close()	입력소스를 닫음으로써 사용하고 있던 자원을 반환한다.
void flush()	스트림의 버퍼에 있는 모든 내용을 출력소스에 쓴다.
abstract void write(int b)	주어진 값을 출력소스에 쓴다.
void write(byte[] b)	주어진 배열 b에 저장된 모든 내용을 출력소스에 쓴다.
void write(byte[] b, int off, int len)	주어진 배열 b에 저장된 내용 중에서 off번째부터 len개 만큼만 읽어서 출력소스에 쓴다.

Stream and I/O

: InputStream 주요 메서드

- ▶ **read()**

- ▶ 입력 스트림으로부터 **1바이트를 읽고 4바이트 int 타입으로 리턴**
- ▶ 리턴된 4 바이트 중 끝의 1바이트에만 데이터가 들어가 있다
- ▶ 더 이상 입력스트림으로부터 바이트를 읽을 수 없다면 -1을 반환



이곳에만 데이터가 들어 있음

Stream and I/O

: InputStream 주요 메서드

- ▶ `read(byte[] b)` 메서드
 - ▶ 매개값으로 주어진 바이트 배열의 길이만큼 바이트를 읽고 배열에 저장
 - ▶ 실제 읽은 바이트 수가 배열의 길이보다 짧을 경우, 읽은 수만큼만 리턴
 - ▶ 더이상 읽을 바이트가 없다면 -1을 리턴

Stream and I/O

: InputStream 주요 메서드

- ▶ `read(byte[] b, int offset, int len)` 메서드

- ▶ 매개값으로 주어진 `offset`만큼을 건너뛴 후, `len` 길이만큼 바이트를 읽고 배열에 저장
- ▶ 실제 읽은 바이트 수가 배열의 길이보다 짧을 경우, 읽은 수만큼만 리턴
- ▶ 더이상 읽을 바이트가 없다면 -1을 리턴

Stream and I/O

: InputStream 주요 메서드

- ▶ **close()** 메서드

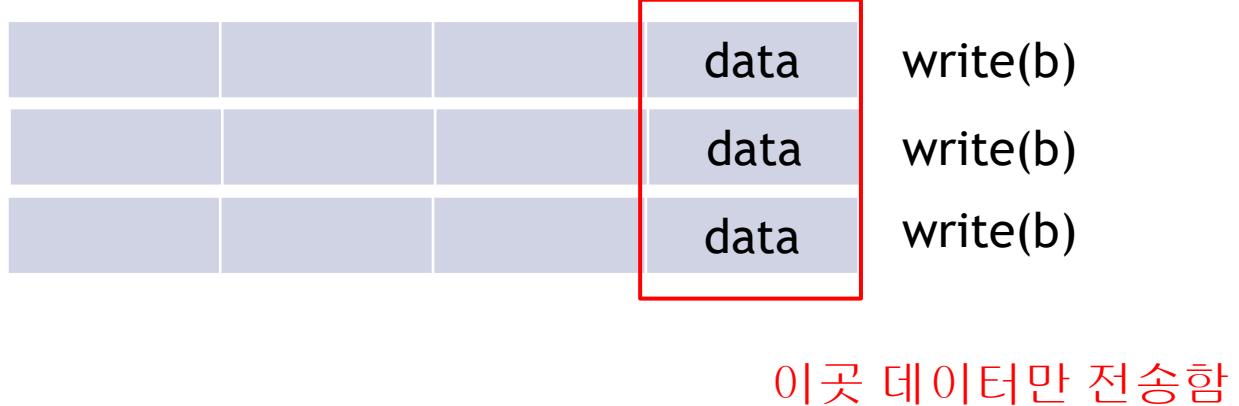
- ▶ **InputStream**을 더이상 사용하지 않을 경우, **close()** 메서드를 호출하여 시스템 자원을 풀어준다
- ▶ 시스템 자원을 풀어주는 것은 매우 중요하다

Stream and I/O

: OutputStream 주요 메서드

- ▶ write(int b) 메서드

- ▶ 매개변수로 주어진 int 값에서 끝에 있는 **1바이트만** 출력 스트림으로 전송



Stream and I/O

: OutputStream 주요 메서드

- ▶ `write(byte[] b)` 메서드
 - ▶ 매개변수로 바이트 배열의 모든 바이트를 출력 스트림으로 전송
- ▶ `write(byte[] b, int offset, int len)` 메서드
 - ▶ 매개변수로 주어진 바이트 배열의 `b[offset]` 부터 `len`개의 바이트를 출력 스트림으로 전송

Stream and I/O

: OutputStream 주요 메서드

- ▶ **flush()** 메서드
 - ▶ OutputStream은 바로 출력되는 것이 아니라 내부에 있는 작은 버퍼에 쌓여 있다가 순서대로 출력됨
 - ▶ flush() 메서드는 버퍼에 쌓여 있는 내용을 모두 출력시키고 버퍼를 비우는 메서드
- ▶ **close()** 메서드
 - ▶ OutputStream을 더이상 사용하지 않을 때 시스템 자원을 풀어준다

Stream and I/O

: ByteArrayInputStream and ByteArrayOutputStream

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.util.Arrays;

public class IOEx1 {
    public static void main(String[] args) throws IOException {
        byte[] inSrc = {0,1,2,3,4,5,6,7,8,9};
        byte[] outSrc = null;

        ByteArrayInputStream input = null;
        ByteArrayOutputStream output = null;

        input = new ByteArrayInputStream(inSrc);
        output = new ByteArrayOutputStream();

        int data = 0;

        while((data = input.read()) != -1) {
            output.write(data);
        }

        outSrc = output.toByteArray();          //Stream의 내용을 byte 배열로 반환

        System.out.println("Input Source: " + Arrays.toString(inSrc));
        System.out.println("Output Source: " + Arrays.toString(outSrc));

        output.close();
        input.close();
    }
}
```

Stream and I/O

: FileInputStream and FileOutputStream

▶ 파일에 데이터를 입출력하는 바이트 기반 스트림

생성자	설명
<code>FileInputStream(String name)</code>	지정된 파일이름(name)을 가진 실제 파일과 연결된 File Input Stream을 생성한다.
<code>FileInputStream(File file)</code>	파일의 이름이 String이 아닌 File인스턴스로 지정해주어야 하는 점을 제외하고 <code>FileInputStream(String name)</code> 와 같다.
<code>FileOutputStream(String name)</code>	지정된 파일이름(name)을 가진 실제 파일과의 연결된 File OutputStream을 생성한다.
<code>FileOutputStream(String name, boolean append)</code>	지정된 파일이름(name)을 가진 실제 파일과 연결된 File OutputStream을 생성한다. 두번째 인자인 append를 true로 하면, 출력 시 기존의 파일내용의 마지막에 덧붙인다. false면, 기존의 파일내용을 덮어쓰게 된다.
<code>FileOutputStream(File file)</code>	파일의 이름을 String이 아닌 File인스턴스로 지정해주어야 하는 점을 제외하고 <code>FileOutputStream(String name)</code> 과 같다.

Java File Basic

: File 클래스

- ▶ java.io 패키지에서 제공
- ▶ 주요 기능
 - ▶ 파일 정보: 파일 크기, 속성, 이름 등의 정보를 알아내는 기능
 - ▶ 파일 조작: 파일 생성 및 삭제
 - ▶ 디렉토리 관련: 디렉토리 생성, 파일 리스트 얻기 등
- ▶ 파일에 데이터를 읽고 쓰는 기능은 지원하지 않음
 - ▶ 파일의 입출력은 Stream을 사용해야 한다

```
File file = new File("D:\\javastudy\\file\\phonedb.txt")
```

· 파일 객체를 생성한 것이지, 파일을 생성한 것은 아니다

Java File Basic

: File 클래스

- ▶ 파일/디렉터리 생성 및 삭제 메소드
 - ▶ 먼저, `exists()` 메소드를 호출하여 이미 존재하는 파일인지 체크하자

리턴타입	메소드	설명
boolean	<code>createNewFile()</code>	새로운 파일을 생성
boolean	<code>mkdir()</code>	새로운 디렉토리를 생성
boolean	<code>mkdirs()</code>	경로상에 없는 모든 디렉토리를 생성
boolean	<code>delete()</code>	파일 또는 디렉토리 삭제

```
boolean isExist = file.exists(); // 파일이 이미 있는지 확인
```

Java File Basic

: File Class

- ▶ 파일 및 디렉터리 정보를 확인하는 메소드 목록

리턴타입	메소드	설명
boolean	canExecute()	실행할 수 있는 파일인지 여부
boolean	canRead()	읽을 수 있는 파일인지 여부
boolean	canWrite()	수정 및 저장할 수 있는 파일인지 여부
String	getName()	파일의 이름을 리턴
String	getParent()	부모 디렉토리를 리턴
File	getParentFile()	부모 디렉토리를 File 객체로 생성후 리턴
String	getPath()	전체 경로를 리턴
boolean	isDirectory()	디렉토리인지 여부
boolean	isFile()	파일인지 여부
boolean	isHidden()	숨김 파일인지 여부
long	lastModified()	마지막 수정 날짜 및 시간을 리턴
long	length()	파일의 크기 리턴
String[]	list()	디렉토리에 포함된 파일 및 서브디렉토리 목록 전부를 String 배열로 리턴
String[]	list(FilenameFilter filter)	디렉토리에 포함된 파일 및 서브디렉토리 목록 중에 FilenameFilter에 맞는 것만 String 배열로 리턴
File[]	listFiles()	디렉토리에 포함된 파일 및 서브 디렉토리 목록 전부를 File 배열로 리턴
File[]	listFiles(FilenameFilter filter)	디렉토리에 포함된 파일 및 서브디렉토리 목록 중에 FilenameFilter에 맞는 것만 File 배열로 리턴

Stream and I/O

: FileInputStream and FileOutputStream - Example

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCopy {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream(args[0]);
            FileOutputStream fos = new FileOutputStream(args[1]);

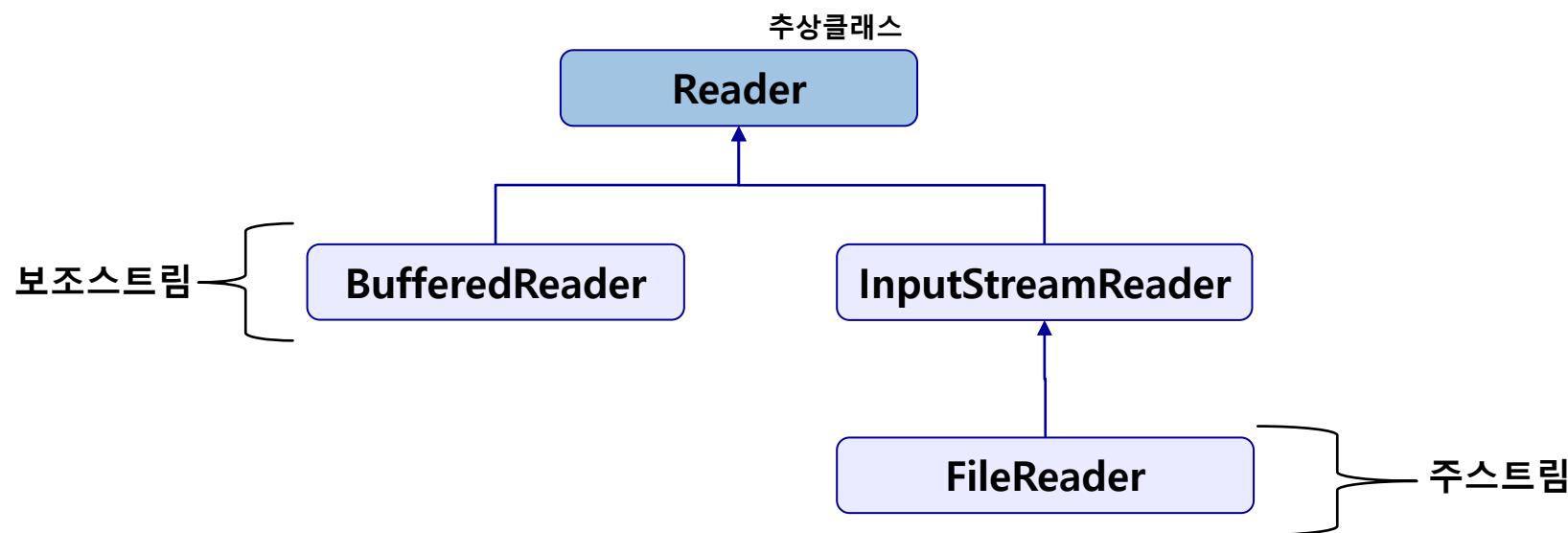
            int data = 0;
            while((data = fis.read()) != -1) {
                fos.write(data); //void write(int b)
            }

            fis.close();
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Stream and I/O

: Character Stream

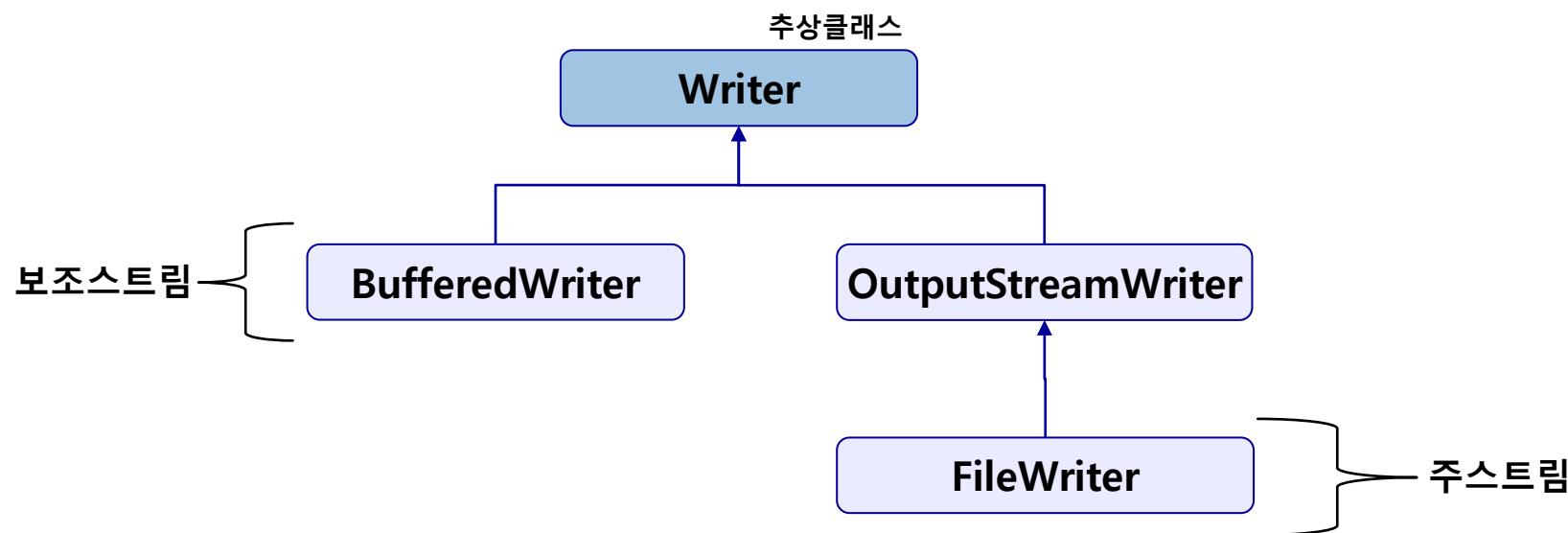
- ▶ Reader (문자기반 입력스트림 최고 조상)



Stream and I/O

: Character Stream

- ▶ Writer (문자기반 출력스트림 최고 조상)



Stream and I/O

: Character Stream

▶ Reader 메서드

메서드	설명
abstract void close()	입력스트림을 닫음으로써 사용하고 있던 자원을 반환한다.
void mark(int readlimit)	현재위치를 표시해놓는다. 후에 reset()에 의해서 표시해 놓은 위치로 다시 돌아갈 수 있다.
boolean markSupported()	mark()와 reset()을 지원하는지를 알려 준다.
int read()	입력소스로부터 하나의 문자를 읽어 온다. char의 범위인 0~65535범위의 정수를 반환하며, 입력스트림의 마지막 데이터에 도달하면, -1을 반환한다.
int read(char[] c):	입력소스로부터 매개변수로 주어진 배열 c의 크기만큼 읽어서 배열 c에 저장한다. 읽어 온 데이터의 개수 또는 -1을 반환한다.
abstract int read(char[] c, int off, int len)	입력소스로부터 최대 len개의 문자를 읽어서, 배열 c의 지정된 위치(off)부터 읽은 만큼 저장한다. 읽어 온 데이터의 개수 또는 -1을 반환한다.
boolean ready()	입력소스로부터 데이터를 읽을 준비가 되어있는지 알려 준다.
void reset()	입력소스에서의 위치를 마지막으로 mark()가 호출되었던 위치로 되돌린다.
long skip(long n)	현재 위치에서 주어진 문자 수(n)만큼을 건너뛴다.

▶ Writer 메서드

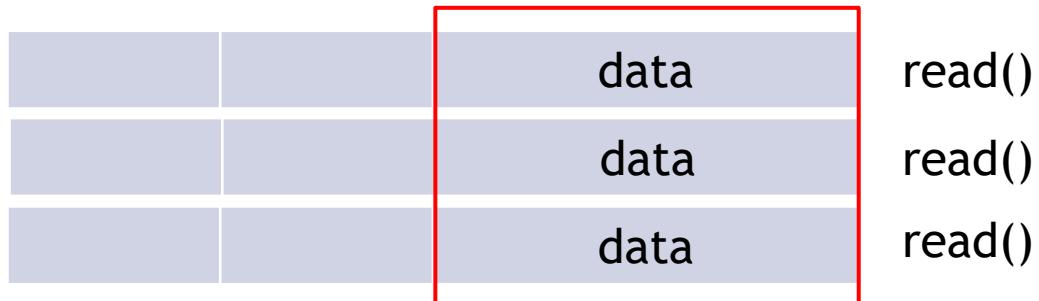
메서드	설명
abstract void close()	출력스트림을 닫음으로써 사용하고 있던 자원을 반환한다.
abstract void flush()	스트림의 버퍼에 있는 모든 내용을 출력소스에 쓴다.(버퍼가 있는 스트림에만 해당됨)
void write(int b)	주어진 값을 출력소스에 쓴다.
void write(char[] c)	주어진 배열 c에 저장된 모든 내용을 출력소스에 쓴다.
abstract void write(char[] c, int off, int len)	주어진 배열 c에 저장된 내용 중에서 off번째부터 len길이만큼만 출력소스에 쓴다.
void write(String str)	주어진 문자열(str)을 출력소스에 쓴다.
void write(String str, int off, int len)	주어진 문자열(str)의 일부를 출력소스에 쓴다.(off번째 문자부터 len개 만큼의 문자열)

Stream and I/O

: Reader 주요 메서드

- ▶ **read()** 메서드

- ▶ 입력 스트림으로부터 한 개의 문자(**2byte**)를 읽고 **4바이트 int 타입**으로 리턴
- ▶ 메서드가 리턴한 **int** 값을 **byte**로 변환하면 읽은 문자를 읽을 수 있음
- ▶ 더 이상 읽어들일 문자가 없다면 **-1**을 반환



Stream and I/O

: Reader 주요 메서드

- ▶ `read(char[] buffer)` 메서드
 - ▶ 입력 스트림으로부터 매개값으로 주어진 문자 배열만큼의 문자를 읽고 저장
 - ▶ 리턴값은 읽은 문자 수
 - ▶ 실제 읽은 문자 수가 배열의 길이보다 작을 경우, 읽은 수만큼을 리턴
- ▶ 더 이상 읽어들일 문자가 없다면 -1을 반환

Stream and I/O

: Reader 주요 메서드

- ▶ `read(char[] buffer, int offset, int len)` 메서드
 - ▶ 입력 스트림으로부터 매개값으로 주어진 `offset` 만큼 건너뛰고 `len` 만큼의 문자를 읽고 저장
 - ▶ 리턴값은 읽은 문자 수
 - ▶ 실제 읽은 문자 수가 `len` 보다 작을 경우, 읽은 수만큼을 리턴
 - ▶ 더 이상 읽어들일 문자가 없다면 -1을 반환
- ▶ `close()` 메서드
 - ▶ Reader를 더 이상 사용하지 않을 경우, 시스템 자원을 풀어준다

Stream and I/O

: Writer 주요 메서드

- ▶ `write(int c)` 메서드
 - ▶ `int` 값에 들어있는 뒤의 2바이트(한 개의 문자)만 출력 스트림으로 전송
- ▶ `write(char[] cbuffer)` 메서드
 - ▶ 매개값으로 주어진 `char[]` 배열의 모든 문자를 출력 스트림으로 전송
- ▶ `write(char[] cbuffer, int offset, int len)` 메서드
 - ▶ `cbuffer[offset]` 부터 `len`개의 문자를 출력 스트림으로 전송

Stream and I/O

: Writer 주요 메서드

- ▶ **write(String str)** 메서드 : 편의 기능
 - ▶ 매개변수로 주어진 **str** 문자열을 출력 스트림으로 전송
- ▶ **write(String str, int offset, int len)** : 편의 기능
 - ▶ 매개값으로 주어진 **str**에서 **offset** 만큼 건너뛰고 **len** 만큼의 문자열을 출력 버퍼로 전송
- ▶ **flush()**
 - ▶ 버퍼에 쌓여있는 내용을 모두 출력하고 버퍼를 비움
- ▶ **close()**
 - ▶ 시스템 자원을 해제

Stream and I/O

: Character Stream : FileReader and FileWriter

- ▶ 문자 기반 파일 입출력,
텍스트 파일의 입출력에 사용

```
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderEx1 {
    public static void main(String args[]) {
        try {
            String fileName = "test.txt";
            FileInputStream fis = new FileInputStream(fileName);
            FileReader fr = new FileReader(fileName);

            int data = 0;
            // FileInputStream으로 파일 내용을 읽어 화면에 출력
            while((data = fis.read()) != -1) {
                System.out.println((char)data);
            }
            System.out.println();
            fis.close();

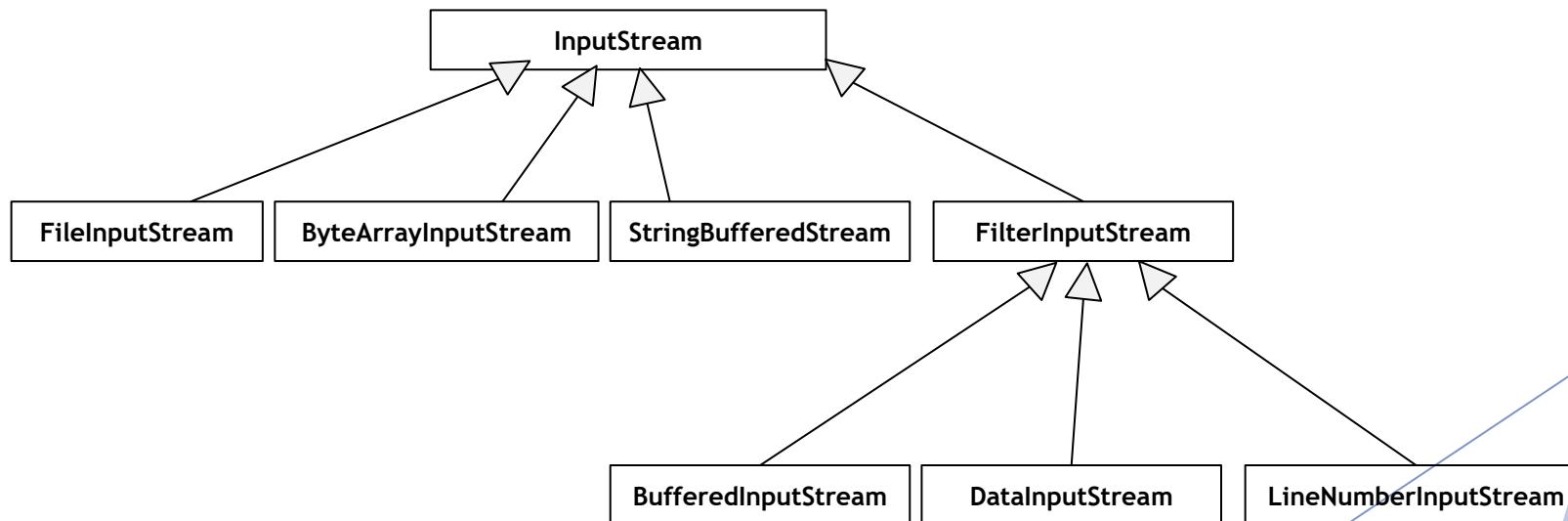
            // FileReader로 파일 내용을 읽어 화면에 출력
            while((data = fr.read()) != -1) {
                System.out.println((char)data);
            }
            System.out.println();
            fr.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Stream and I/O

: 보조 스트림

▶ 데코레이터 패턴

- ▶ 객체의 추가적인 요건을 동적으로 첨가
- ▶ 서브클래스를 만드는 방식으로 기능을 유연하게 확장할 수 있는 방법을 제공
- ▶ 장식할 대상과 장식을 동일시하는 패턴



Stream and I/O

: 바이트 기반 보조 스트림

▶ FilterInputStream과 FilterOutputStream

- ▶ 모든 바이트 기반 보조스트림의 최고조상
- ▶ 보조스트림은 자체적으로 입출력을 수행할 수 없다
- ▶ 상속을 통해 FilterInputStream/FilterOutputStream의 read()와 write()를 오버라이딩 해야 한다

```
public class FilterInputStream extends InputStream {  
    protected volatile InputStream in;  
    protected FilterInputStream(InputStream in) {  
        this.in = in;  
    }  
  
    public int read() throws IOException {  
        return in.read();  
    }  
    ...  
}
```

FilterInputStream의 자손 - BufferedInputStream, DataInputStream, PushbackInputStream 등
FilterOutputStream의 자손 - BufferedOutputStream, DataOutputStream, PrintStream 등

Stream and I/O

: 바이트 기반 보조 스트림

▶ BufferedInputStream과 BufferedOutputStream

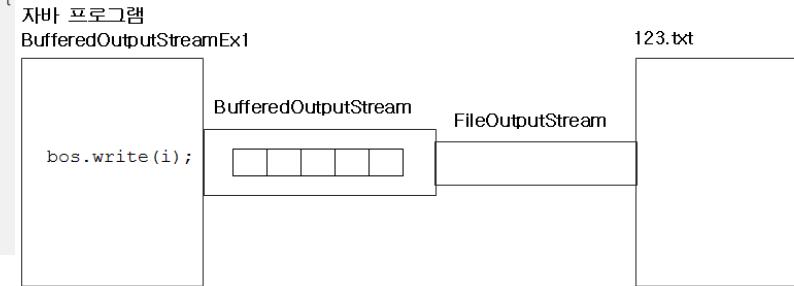
- ▶ 입출력 효율을 높이기 위해 버퍼(byte[])를 사용하는 보조 스트림
- ▶ 보조스트림을 닫으면 기반스트림도 닫힌다

```
import java.io.*;

class BufferedOutputStreamEx1 {
    public static void main(String args[]) {
        try {
            FileOutputStream fos = new FileOutputStream("123.txt");
            // BufferedOutputStream의 버퍼 크기를 5로 한다.
            BufferedOutputStream bos = new BufferedOutputStream(fos, 5);
            // 파일 123.txt에 1부터 9까지 출력한다.
            for(int i='1'; i <= '9'; i++) {
                bos.write(i);
            }

            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

【실행결과】
C:\jdk1.5\work\ch14>java BufferedOutputStreamEx1
C:\jdk1.5\work\ch14>type 123.txt
12345



Stream and I/O

: 문자 기반 보조 스트림

▶ BufferedReader와 BufferedWriter

- ▶ 입출력 효율을 높이기 위해 버퍼(char[])를 사용
- ▶ 라인(line) 단위의 입출력에 편리

String readLine() - 한 라인을 읽어온다. (BufferedReader의 메서드)
void newLine() - '라인 구분자(개행문자)' 를 출력한다. (BufferedWriter의 메서드)

```
import java.io.*;

class BufferedReaderEx1 {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("BufferedReaderEx1.java");
            BufferedReader br = new BufferedReader(fr);

            String line = "";
            for(int i=1;(line = br.readLine())!=null;i++) {
                // ";"를 포함한 라인을 출력한다.
                if(line.indexOf(";") !=-1)
                    System.out.println(i+":"+line);
            }
            br.close();
        } catch(IOException e) {}
    } // main
}
```

【실행결과】

```
1:import java.io.*;
6:    FileReader fr = new FileReader("BufferedReaderEx1.java");
7:    BufferedReader br = new BufferedReader(fr);
9:    String line = "";
10:   for(int i=1;(line = br.readLine())!=null;i++) {
11:       // ";"를 포함한 라인을 출력한다.
12:       if(line.indexOf(";") !=-1)
13:           System.out.println(i+":"+line);
```

Stream and I/O

: 문자 기반 보조 스트림

- ▶ InputStreamReader와 OutputStreamWriter

- ▶ 바이트 기반 스트림을 문자 기반 스트림처럼 사용할 수 있게 해준다
- ▶ 인코딩(encoding)을 변환하여 입출력할 수 있게 해준다
- ▶ 콘솔(console)로부터 라인 단위로 입력받기

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
String line = br.readLine();
```

- ▶ 인코딩 변환하기

```
FileInputStream fis = new FileInputStream("korean.txt");
InputStreamReader isr = new InputStreamReader(fis, "KSC5601");
```

Stream and I/O

: 문자 기반 보조 스트림

- ▶ InputStreamReader와 OutputStreamWriter 예제

```
import java.io.*;

class ReadKBLine() throws IO Exception {
    public static void main(String[] arg) throws Exception {
        try {
            BufferedReader keybd = new BufferedReader(new InputStreamReader(System.in));
            System.out.println(keybd.readLine());
        } catch (IOException e) {}
    }
}
```

- ▶ BufferedReader 클래스의 Constructor : BufferedReader(Reader in)
- ▶ InputStreamReader 클래스의 Constructor : InputStreamReader(InputStream in)
- ▶ System.in 자체가 InputStream 객체

Stream and I/O

: 문자 기반 보조 스트림 - InputStreamReader와 OutputStreamWriter 예제 2

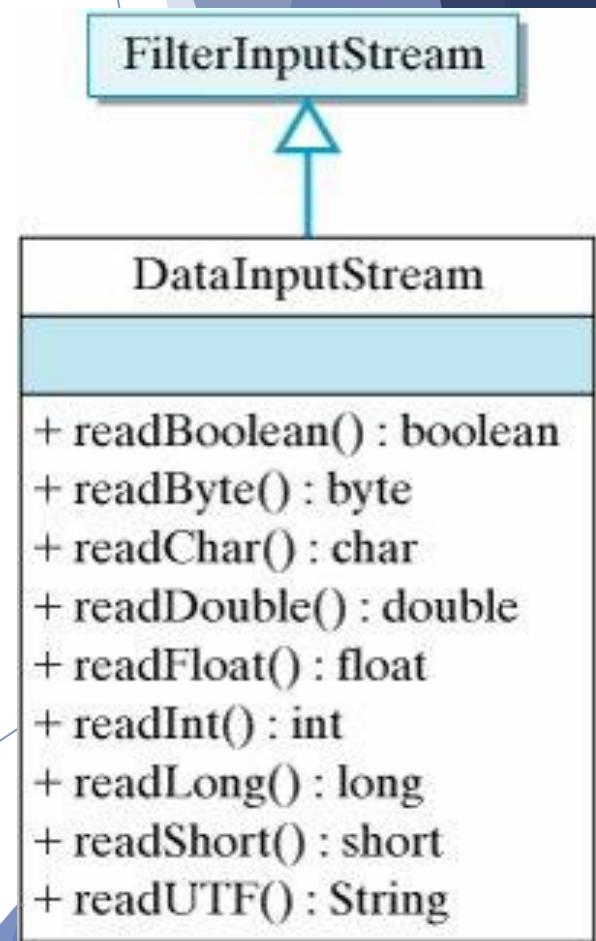
```
public class eggMonster {  
    public static void main(String[] args) {  
        String morningEgg;  
        String lunchEgg;  
        String dinnerEgg;  
  
        try {  
            BufferedReader keybd = new BufferedReader(new InputStreamReader(System.in));  
            System.out.print("아침에 먹은 계란 갯수: ");  
            morningEgg = keybd.readLine();  
            System.out.println("아침에 계란" + morningEgg + "개");  
            System.out.println();  
            System.out.print("점심에 먹은 계란 갯수: ");  
            lunchEgg = keybd.readLine();  
            System.out.println("점심에 계란" + lunchEgg + "개");  
            System.out.println();  
            System.out.print("저녁에 먹은 계란 갯수: ");  
            dinnerEgg = keybd.readLine();  
            System.out.println("저녁에 계란" + dinnerEgg + "개");  
        } catch (IOException e) {  
            e.printStackTrace();  
            System.exit(1);  
        }  
    }  
}
```

Stream and I/O

: 기본 타입 입출력 보조스트림

▶ DataInputStream과 DataOutputStream

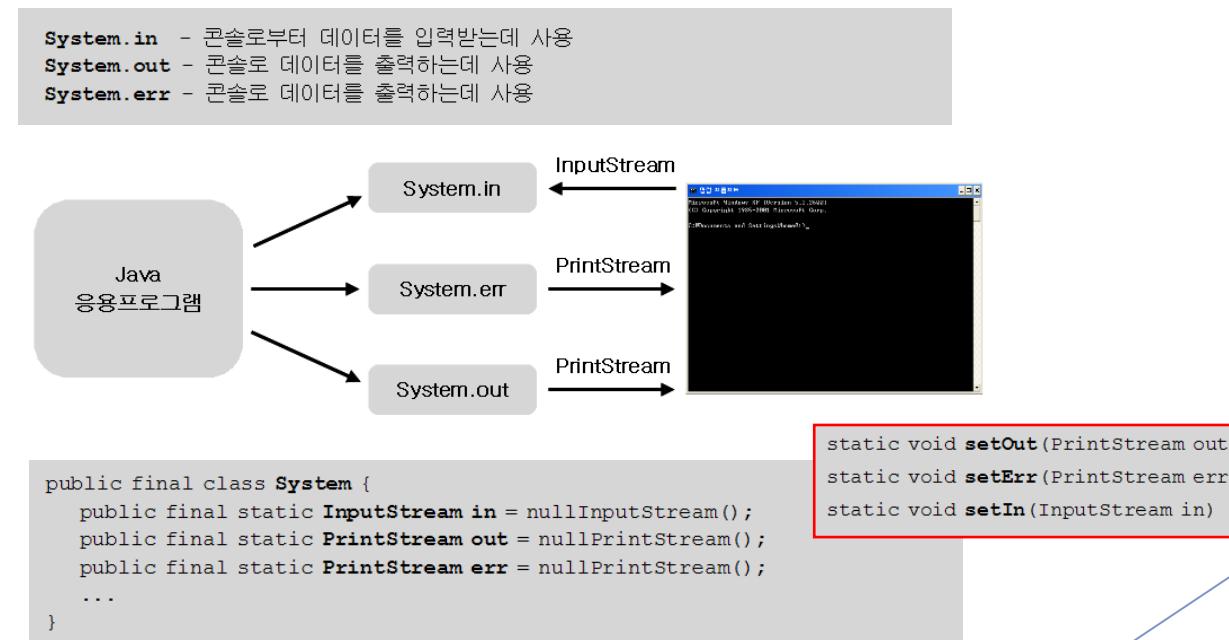
- ▶ 바이트 기반 스트림은 기본 데이터 타입(Primitive Types)을 입출력할 수 없다
- ▶ DataInputStream과 DataOutputStream을 이용하면 기본 타입 입출력이 가능하다
- ▶ 주의: DataInputStream으로 읽을 때는 DataOutputStream에서 출력한 순서대로 읽어야 한다
 - ▶ 각 데이터 타입의 크기가 모두 다르기 때문
 - ▶ 예) 출력 순서 : String -> int -> double
 입력 순서 : String -> int -> double



Stream and I/O

: 표준입출력

- ▶ 콘솔을 통한 데이터의 입출력을 ‘표준입출력’ 이라 한다
- ▶ JVM이 시작되면서 자동적으로 생성되는 스트림



Stream and I/O

: 파일 닫기

- ▶ 정상적으로 수행되었을 경우, Java가 알아서 파일을 **close** 해줌
- ▶ **close()** 메서드를 호출하는 이유
 - ▶ 비정상적으로 프로그램이 끝났을 경우 파일이 손상되는 것을 막아줌
 - ▶ 파일에 출력한 이후 **close** 해주어야 읽을 수 있음
 - ▶ **RandomAccessFile** 클래스 객체를 이용, 파일에서 출력과 읽어들이는 것을 동시에 할 수 있으나 일반적으로 **Transaction** 관점에서 바람직하지 않음

Stream and I/O

: StringTokenizer

- ▶ **java.util.StringTokenizer**
 - ▶ 문자열을 특정 구분자(Delimiter)로 분리해내는 기능 수행
- ▶ **StringTokenizer의 Constructor**
 - ▶ **StringTokenizer(String str)**
 - ▶ Default 생성자는 구분자로 “\t\n\r”을 가정
 - ▶ **StringTokenizer(String str, String delim)**

```
String tel = “010-9000-7777”;
StringTokenizer st = new StringTokenizer(tel, “#-”); // #과 -을 구분자로 사용
while(st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

Stream and I/O

: StringTokenizer - Example

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

public class StringTokenizerTest {

    public static void main(String[] args) {

        try {
            BufferedReader br = new BufferedReader(new FileReader("c:\\temp\\phone.txt"));
            String phoneList;

            while ((phoneList = br.readLine()) != null) {
                StringTokenizer st = new StringTokenizer(phoneList, " -.\t");
                while(st.hasMoreTokens()) {
                    System.out.println(st.nextToken());
                }
            } catch (IOException e) {
                e.printStackTrace();
                System.exit(1);
            }
        }
    }
}
```

phone.txt

둘리	010-2222-2222
고길동	011-0000-2345
도우너	010-5555-5555
마이콜	011-9999-7777

Stream and I/O

: Scanner

- ▶ Scanner

- ▶ Java 5부터 `java.util.Scanner` 제공
- ▶ `BufferedReader`와 `FileReader`, `StringTokenizer`의 조합 대신 사용
- ▶ 키보드 입력 사용의 예

```
Scanner skb = new Scanner(System.in);
System.out.print("Enter name: ");
String name = stdin.nextLine();
System.out.print("Enter age: ");
int age = stdin.nextInt();
```

- ▶ `File`에서 사용 예

```
File file = new File( "inputFile.txt" );
Scanner sf = new Scanner( file );
```

Stream and I/O

: Scanner - Example

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

public class ScannerTest {

    public static void main(String[] args) {
        try {
            File file = new File( "c:\\temp\\phone2.txt" );
            Scanner sf = new Scanner( file );
            String name;
            int number1;      // 전화번호 첫번째 자리
            int number2;      // 전화번호 두번째 자리
            int number3;      // 전화번호 세번째 자리
            while ( sf.hasNextLine() ) {
                name = sf.next();
                // 첫번째 Delimiter(0이 경우는 스페이스)까지 String으로 읽음
                number1 = sf.nextInt(); // Integer값으로 Parsing
                number2 = sf.nextInt();
                number3 = sf.nextInt();
                System.out.println( name + number1 + number2 + number3 );
            }
        } catch(FileNotFoundException ex) {
        }
    }
}
```

phone2.txt

```
둘리 010 2222 2222
고길동 011 0000 2345
```

Stream and I/O

: Scanner vs BufferedReader

- ▶ Scanner 클래스가 있는데도, BufferedReader 클래스를 InputStream, InputStreamReader, FileReader, StringTokenizer 등과 조합해서 사용하는 이유
 - ▶ 하나의 업무를 하나의 클래스에서 수행하여 클래스의 변경 없이 다양한 경우에 조합해서 사용 가능
 - ▶ Scanner 클래스로 처리 가능한 경우는 Scanner 클래스 사용을 추천

	Scanner	BufferedReader
Parsing	<ul style="list-style-type: none">▪ Parsing 자료형에 따라 nextInt(), nextFloat(), next() 등 사용	<ul style="list-style-type: none">▪ Read(), readLine() 등의 메쏘드로는 Parsing 지원하지 않음▪ StringTokenizer와 Wrapper 클래스 메쏘드 사용 Ex) Integer.parseInt(Delimitedtoken)
EOF/EOS*	<ul style="list-style-type: none">▪ nextLine(), nextInt()는 exception을 throw 함▪ hasNextLine(), hasNextInt() 등을 사용해서 미리 확인할 필요	<ul style="list-style-type: none">▪ readLine() - null 값은 return▪ read() - “-1” 값을 return

* EOF/EOS - End of File / End of Stream

Java Network and Thread

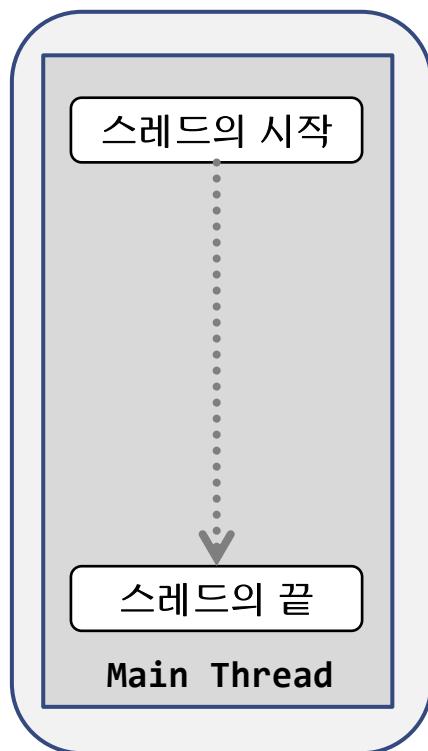
Java Network and Thread

Thread

Java Thread

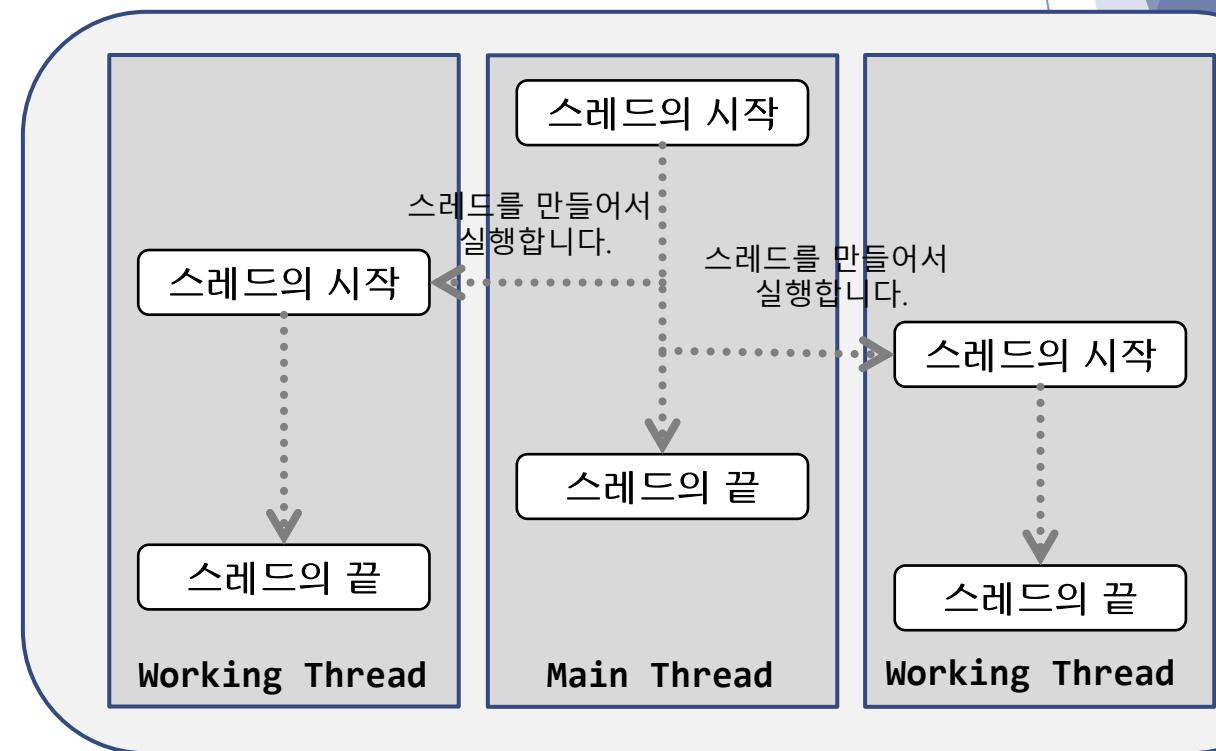
: Thread: 프로그램의 실행 흐름

- **싱글스레드(single thread) 프로그램:**
스레드가 하나뿐인 프로그램



Process: 실행중인 하나의 프로그램
Thread: 프로세스 내의 하나의 작업 흐름

- **멀티스레드(multi thread) 프로그램:**
스레드가 둘 이상인 프로그램



Java Thread

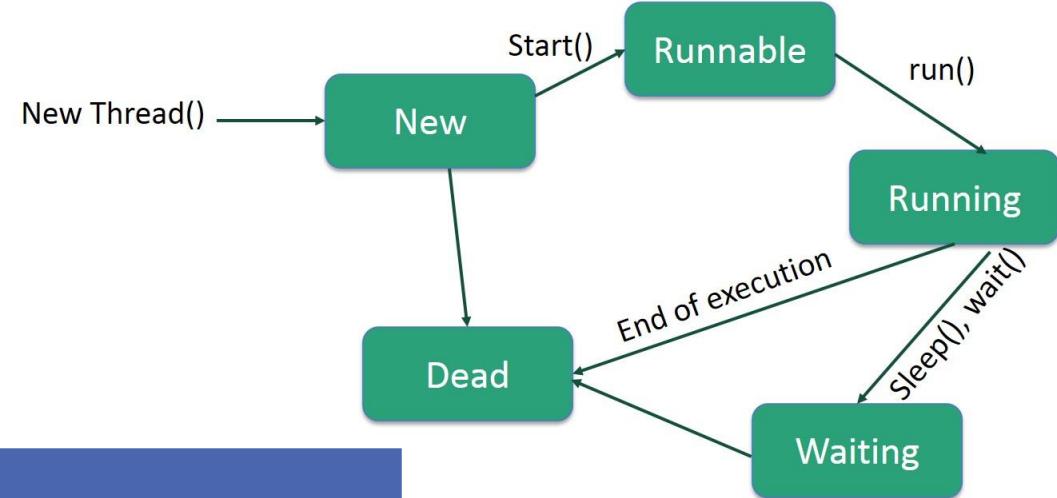
: Thread의 상태와 Multi Thread 프로그램의 작성 방법

▶ Thread의 상태

상태	열거 상수	설명
객체 생성	NEW	쓰레드 객체가 생성. 아직 <code>start()</code> 메서드가 호출되지 않은 상태
실행 대기	RUNNABLE	실행 상태로 언제든지 갈 수 있는 상태
일시 정지	WAITING	다른 쓰레드가 통지할 때까지 기다리는 상태
	TIMED_WAITING	주어진 시간 동안 기다리는 상태
	BLOCKED	사용하고자 하는 객체의 락이 풀릴 때까지 기다리는 상태
종료	TERMINATED	실행을 마친 상태

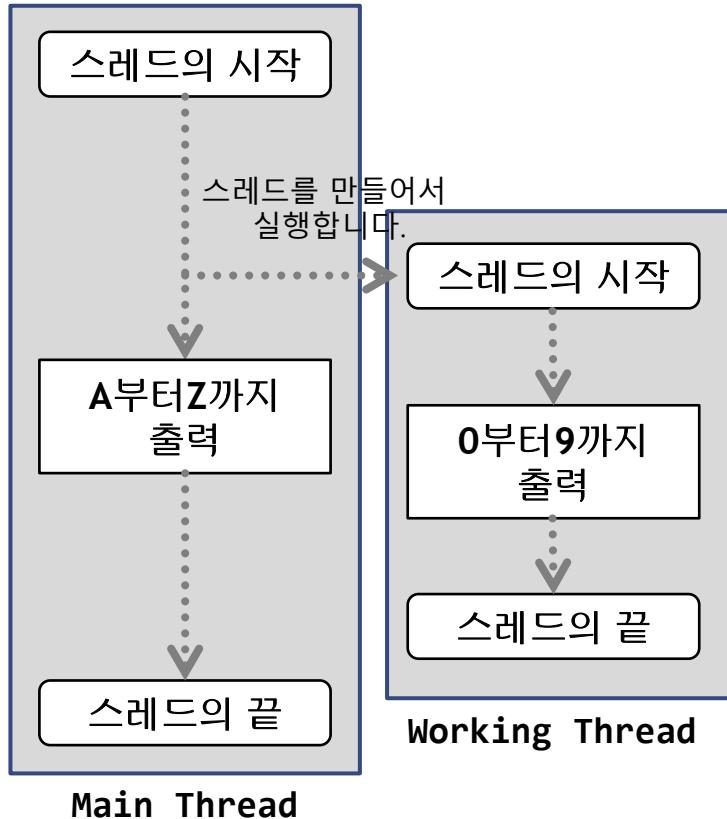
▶ Multi Thread 프로그램 작성 방법

- ▶ `java.lang.Thread` 클래스를 이용하는 방법
- ▶ `java.lang.Runnable` 인터페이스를 이용하는 방법



Java Thread

: java.lang.Thread 클래스를 이용하는 방법



main 메소드를 포함하는 클래스

```
1 public class Multithread {  
2     public static void main(String args[]) {  
3         Thread thread = new DigitThread(); // 스레드를 생성  
4         thread.start(); // 스레드를 시작  
5         for (char ch = 'A'; ch <= 'Z'; ch++) {  
6             System.out.print(ch);  
7         }  
8     }  
9 }
```

숫자를 출력하는 스레드 클래스

```
1 public class DigitThread extends Thread {  
2     public void run() {  
3         for (int cnt = 0; cnt < 10; cnt++) {  
4             System.out.print(cnt);  
5         }  
6     }  
7 }
```

연습문제

: java.lang.Thread 클래스 연습

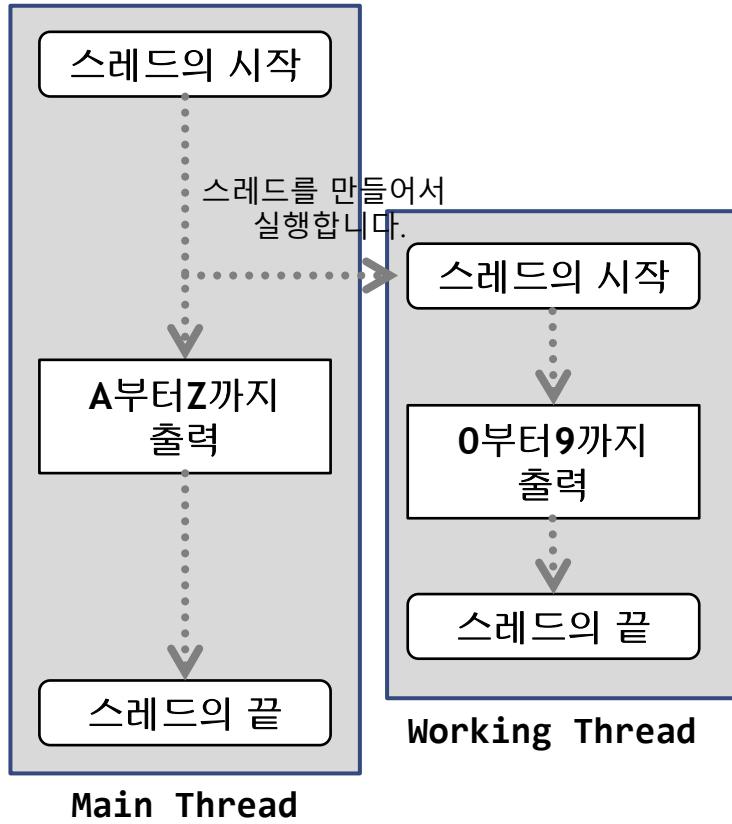
- ▶ 다음 코드를 참고하여 멀티스레드 프로그램을 작성하고 테스트 한다.

```
1 public class Multithread {  
2     public static void main(String args[]) {  
3           
4             Thread thread1 = new DigitThread();  
5             Thread thread2 = new DigitThread();  
6             Thread thread3 = new AlphabetThread();  
7           
8             thread1.start();  
9             thread2.start();  
10            thread3.start();  
11     }  
12 }
```

3개의 스레드를
생성해서 시작

Java Thread

: java.lang.Runnable 인터페이스를 이용하는 방법



main 메소드를 포함하는 클래스

```
1 public class Multithread {
2     public static void main(String args[]) {
3         Thread thread = new Thread(new DigitRunnableImpl());
4                                         // 스레드를 생성
5         thread.start();                      // 스레드를 시작
6         for (char ch = 'A'; ch <= 'Z'; ch++) {
7             System.out.print(ch);
8         }
9     }
10 }
```

숫자를 출력하는 스레드 클래스

```
1 public class DigitRunnableImpl implements Runnable {
2     public void run() {
3         for (int cnt = 0; cnt < 10; cnt++) {
4             System.out.print(cnt);
5         }
6     }
7 }
```

연습문제

: java.lang.Runnable 인터페이스 연습

- ▶ 다음 코드를 참고하여 멀티스레드 프로그램을 작성하고 테스트한다 (**Runnable** 이용)

```
1 public class Multithread {  
2     public static void main(String args[]) {  
3         Thread thread1 = new Thread(new DigitRunnableImpl());  
4         Thread thread2 = new Thread(new DigitRunnableImpl());  
5         Thread thread3 = new Thread(new AlphabetRunnableImpl());  
6         thread1.start();  
7         thread2.start();  
8         thread3.start();  
9     }  
10 }
```

3개의 스레드를
생성해서 시작

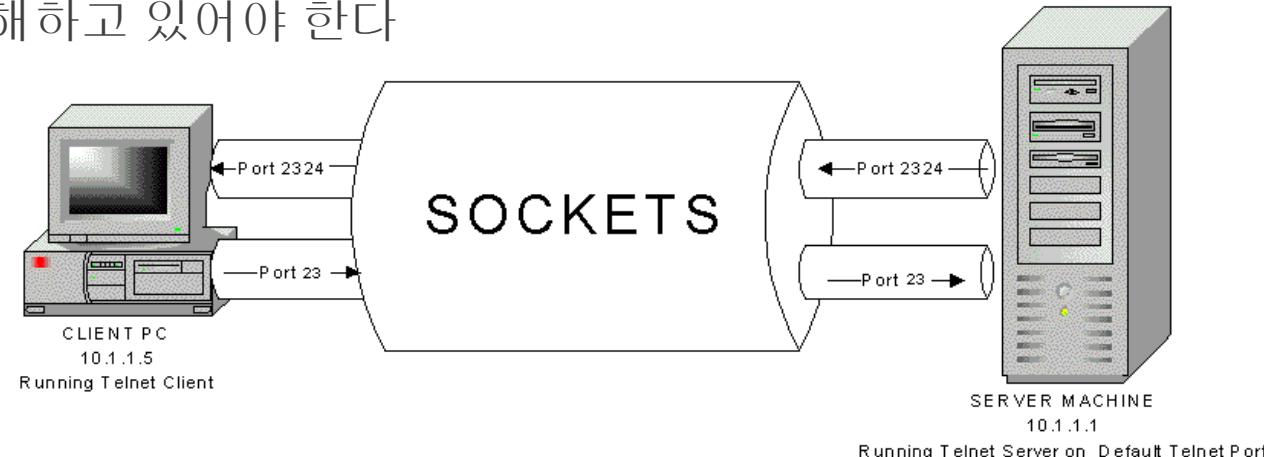
Java Network and Thread

TCP Socket Programming

TCP Socket Programming

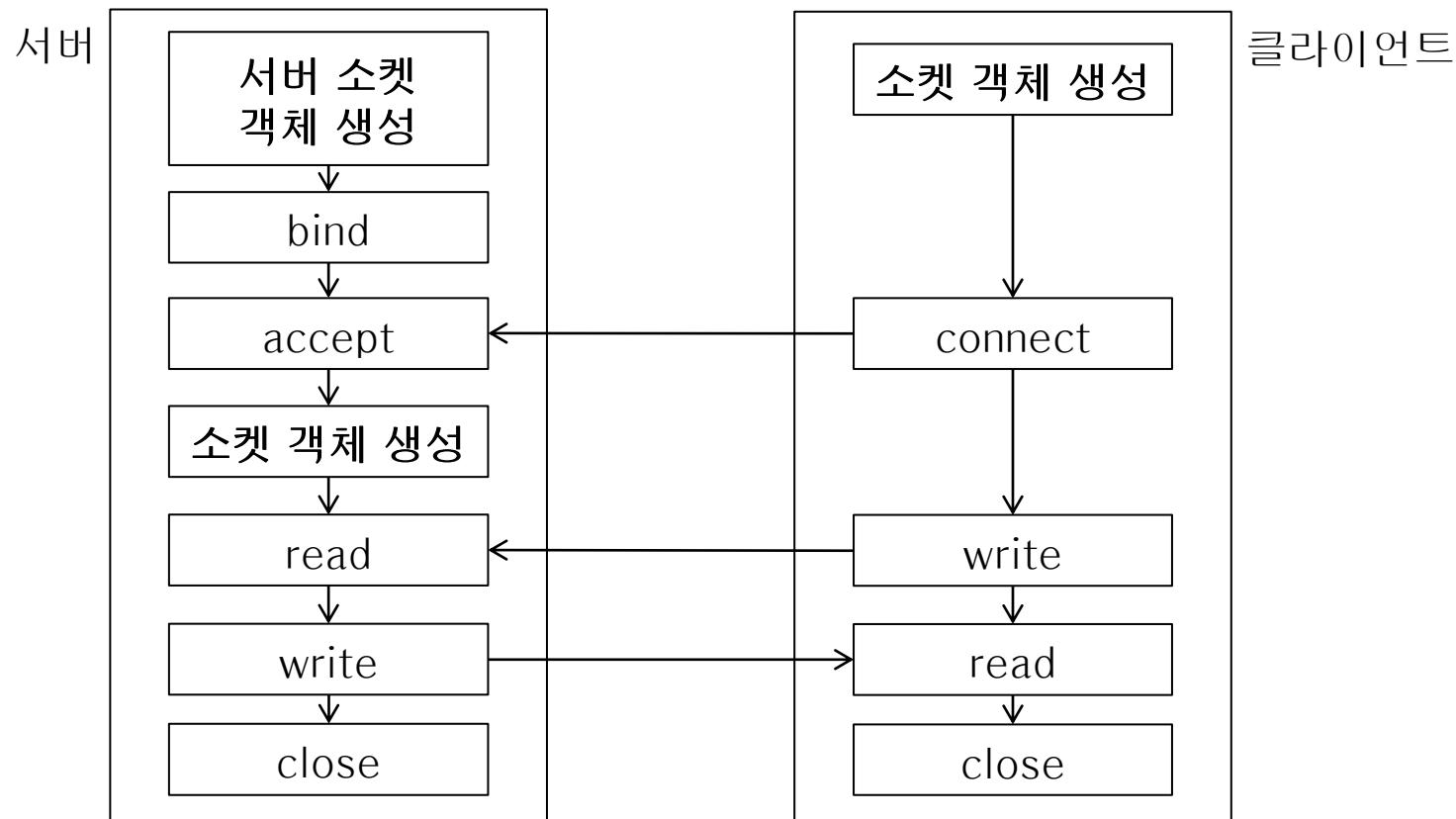
: TCP 소켓의 특징

- ▶ 스트림(stream) 통신 프로토콜
- ▶ 양쪽의 소켓이 연결된 상태에서 통신이 가능하다(연결지향 프로토콜)
- ▶ 신뢰성 있는 데이터 통신
- ▶ 한번 연결되면 끊어질 때까지 송신한 데이터는 차례대로 목적지의 소켓에 전달
- ▶ 자바는 `java.net` 패키지에 관련 클래스를 제공
 - ▶ TCP 소켓 프로그래밍을 쉽게 할 수 있도록 돋는다
- ▶ 라이브러리의 사용법과 작동 순서를 명확하게 이해하고 있어야 한다
- ▶ `ServerSocket`과 `Socket` 클래스를 사용한다



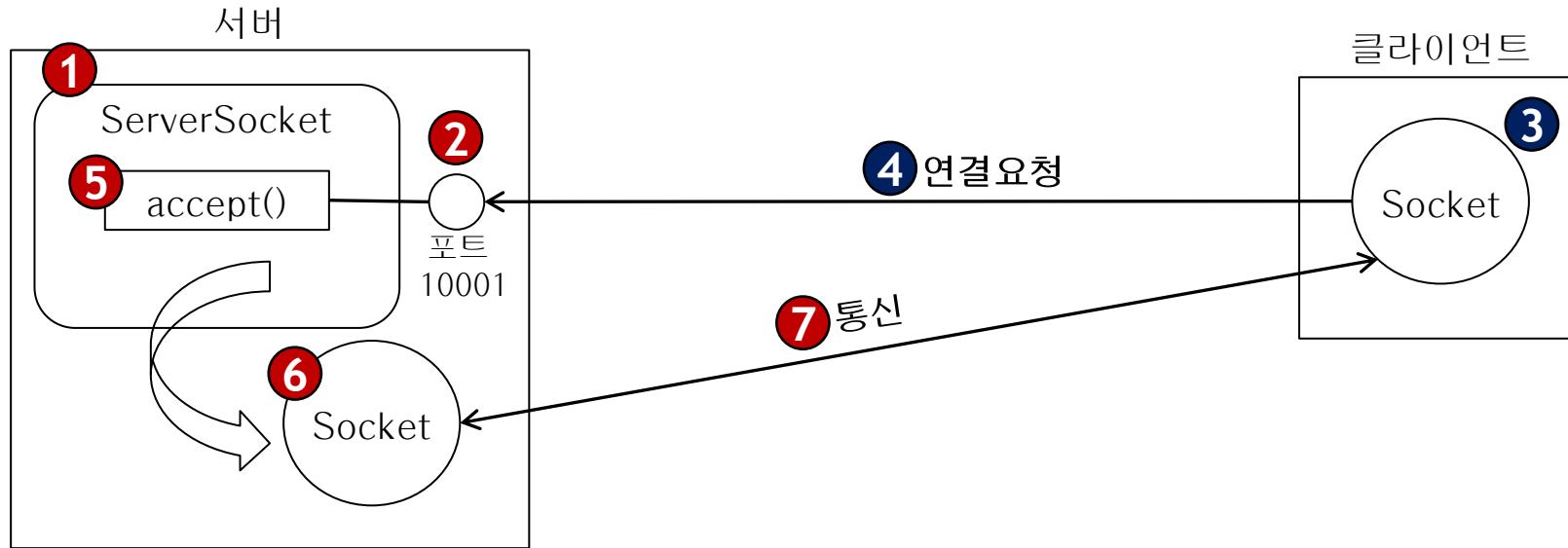
TCP Socket Programming

: TCP 소켓 프로그래밍 절차



TCP Socket Programming

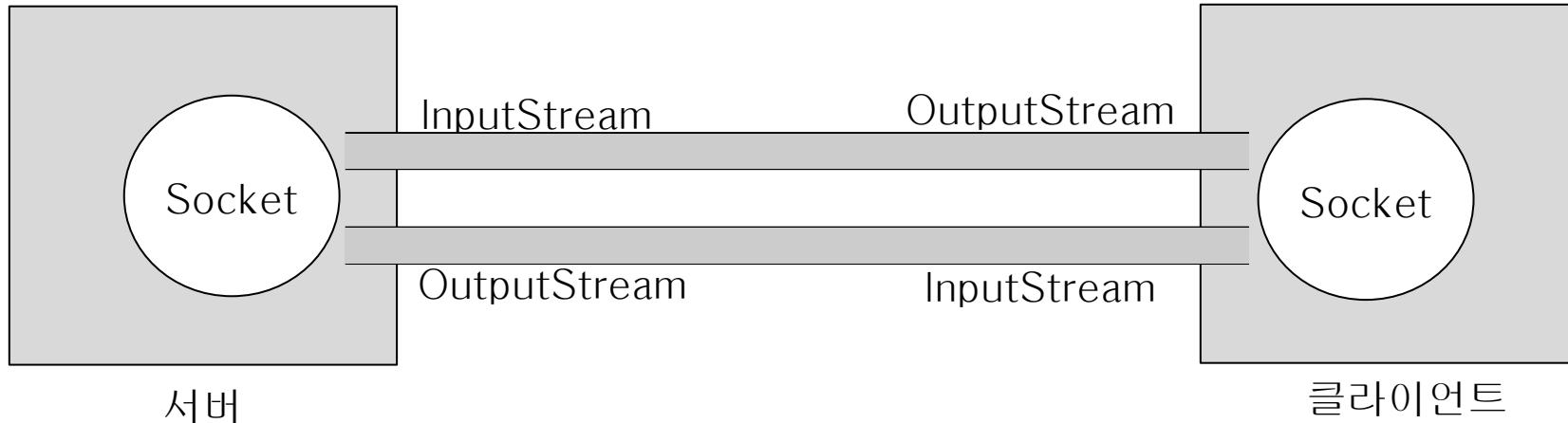
: ServerSocket과 Socket



- ▶ **ServerSocket** : 클라이언트의 연결 요청을 기다리면서 연결 요청에 대한 수락을 담당
- ▶ **Socket** : 클라이언트와 통신을 직접 담당한다

TCP Socket Programming

: Socket 객체의 데이터 통신



- ▶ 양쪽의 **Socket** 객체로부터 **InputStream**, **OutputStream**을 얻어와 데이터 통신에 사용