

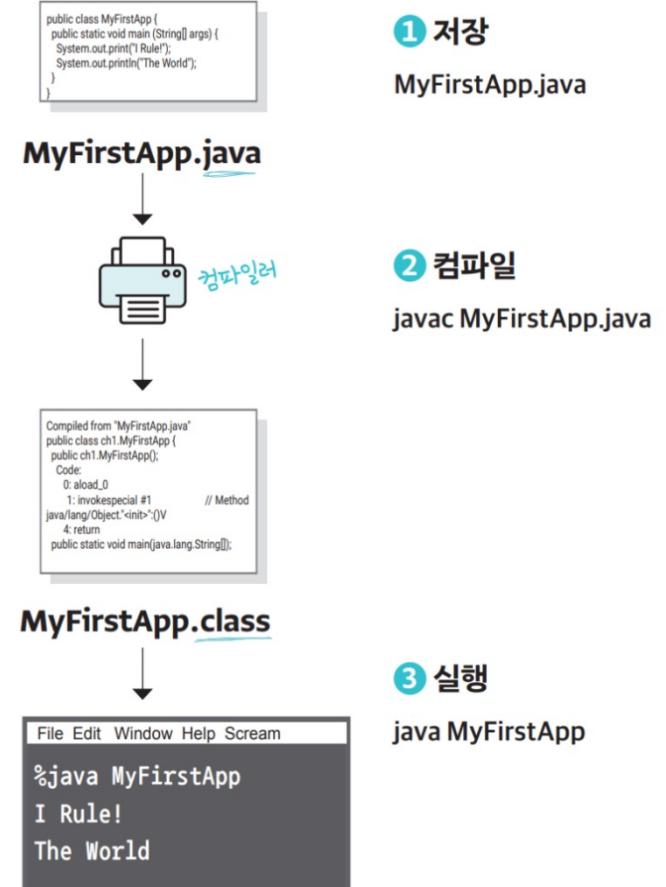


Java



시작하기 전에

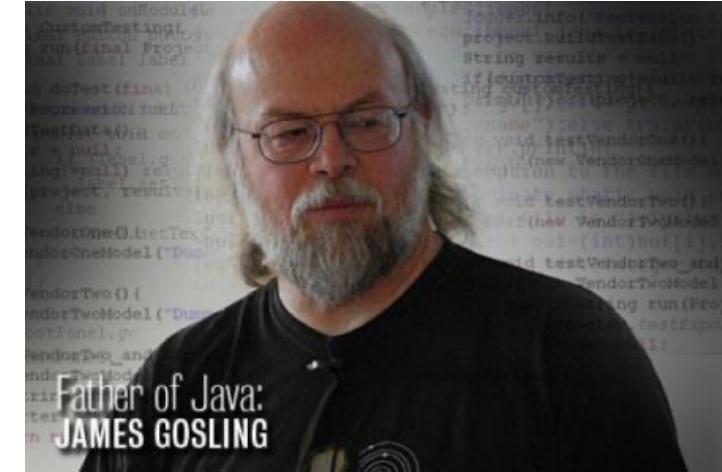
- 컴퓨터에서 실행하는 프로그램(program)은 특정 목적을 수행하도록 프로그래밍 언어로 작성된 소스를 기계어로 컴파일한 것.
- 기계어 (Machine language)
 - CPU가 직접 해독하고 실행할 수 있는 비트 단위로 쓰인 컴퓨터의 언어 (0과 1로 구성)
 - 사람과 기계 사이의 다리 역할을 하는 프로그래밍 언어 필요
- 프로그래밍 언어 (Programming language)
 - 컴퓨터 시스템을 구동시키는 소프트웨어를 작성하기 위한 형식 언어
 - 컴퓨터가 이해할 수 있는 명령을 작성하는 일련의 도구
- 소스 (Source File) : 프로그래밍 언어로 작성한 파일
- 컴파일 (Compile) : 소스 파일을 기계어 파일로 번역





자바 (Java)

- 1995년 썬 마이크로시스템즈(Sun Microsystems)에서 발표
- 현재 웹사이트 및 다양한 애플리케이션 개발의 핵심 언어
- 특징
 - 모든 운영체제에서 실행 가능 (플랫폼 독립성 - Cross Platform)
 - **객체 지향 프로그래밍** (OOP : Object-Oriented Programming)
 - 메모리 자동 정리 (Garbage Collection)
 - 풍부한 무료 라이브러리
 - 멀티쓰레드



Write Once, Run Anywhere(WORA)



자바 (Java)

- 1991. 그린프로젝트(Green Project)로 탄생한 오크(Oak): 자바의 전신
- 1993. 인터넷과 웹의 급속한 발전
- 1995. 인터넷 환경에 적합하도록 오크를 새롭게 설계 (Java 정식 발표)
- 1996. Java [JDK 1.0]
- 1998. Java [J2SE 1.2] (*Java 2 Standard Edition)
- 2004. Java [J2SE 5]
- 2006. Java [Java SE 6]
- 2014.03. Java SE 8
- 2017.09. Java SE 9
- 2023.03. Java SE 20
- 2023.09. Java SE 21 (LTS)
- 2024.03. Java SE 22

TIP

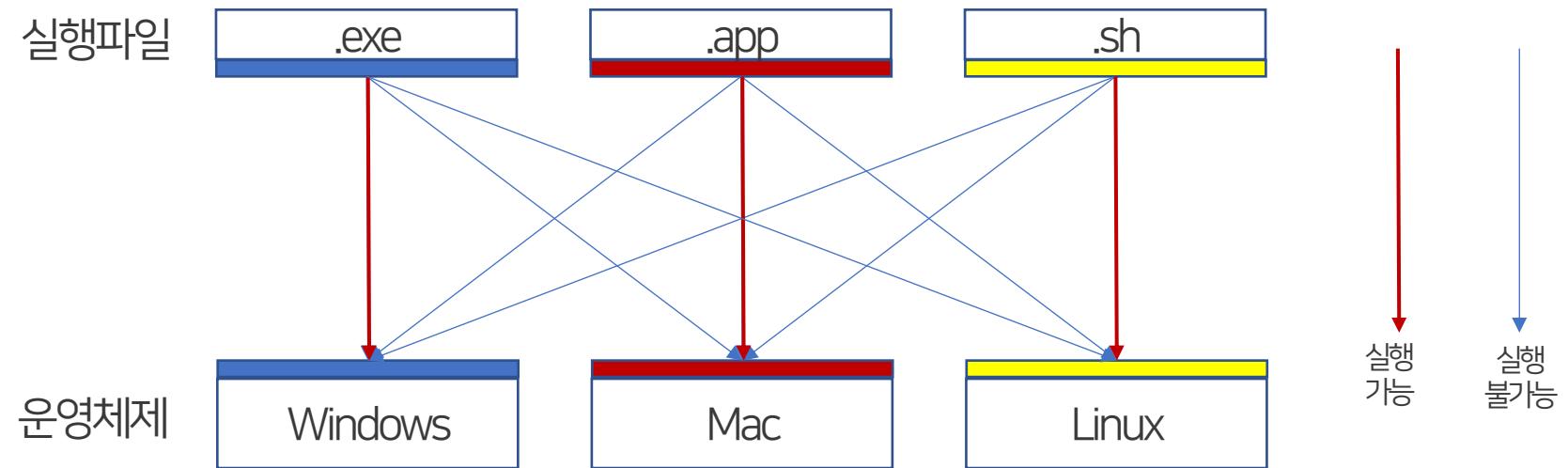
본인의 팀명(사무실 문 색상)을 따서 이름을 그린톡(Greentalk)으로 지었다가, 사무실 바깥의 오크나무를 보고 오크(Oak)로 바꿨다.

Oak라는 이름이 Oak Technologies에서 이미 사용 중이라는 사실을 알고, 커피를 좋아해서 java라는 이름을 사용했다.

TIP

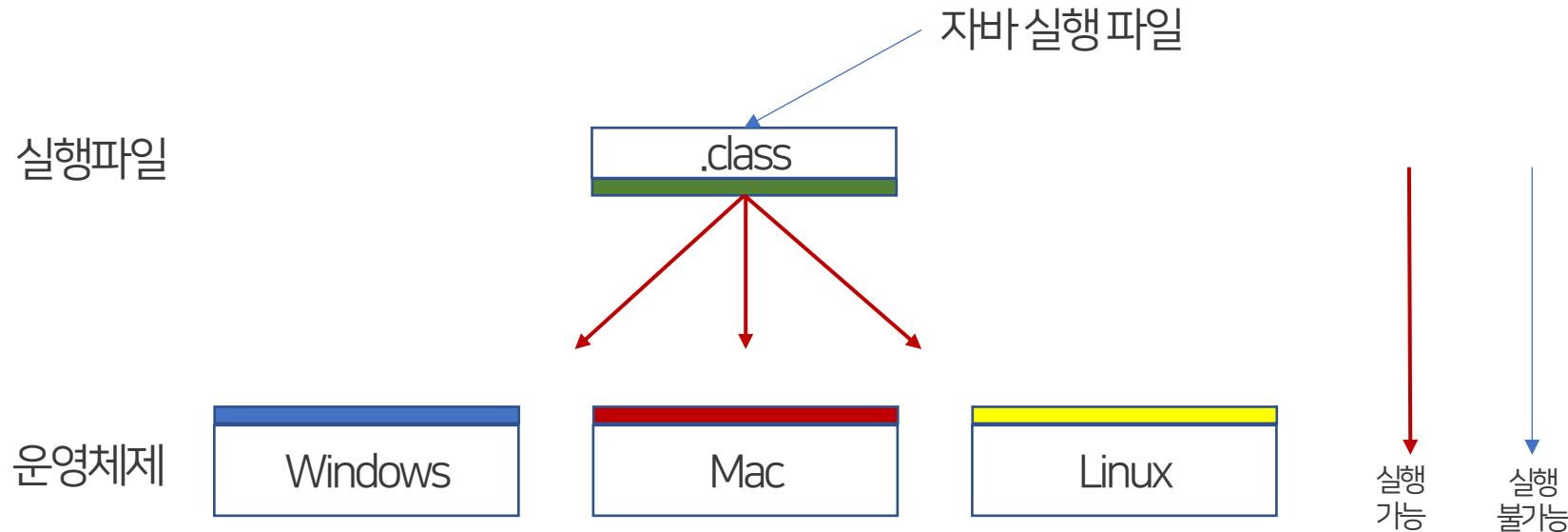
Java의 LTS 버전: 8, 11, 17, 21

플랫폼 종속 vs. 플랫폼 독립





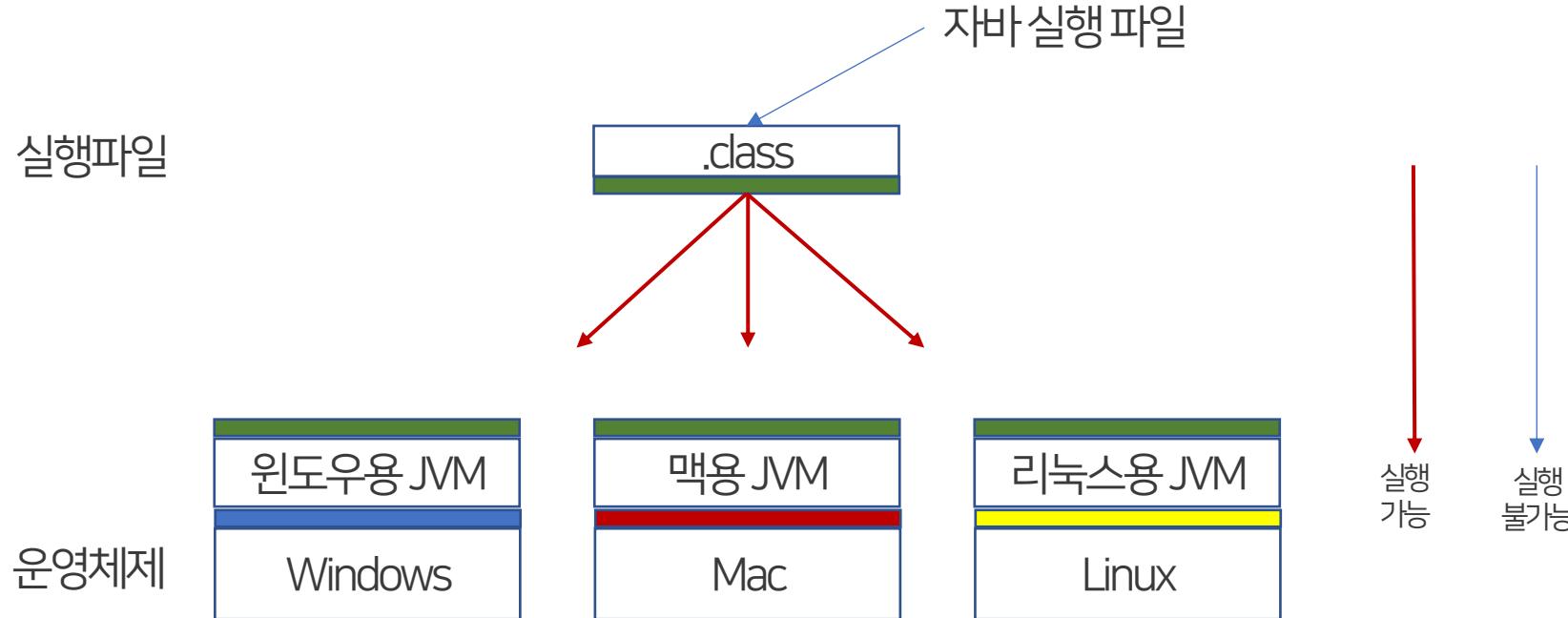
플랫폼 종속 vs. 플랫폼 독립



Q. 어떻게 운영체제와 관계없이 실행 가능할까?



플랫폼 종속 vs. 플랫폼 독립

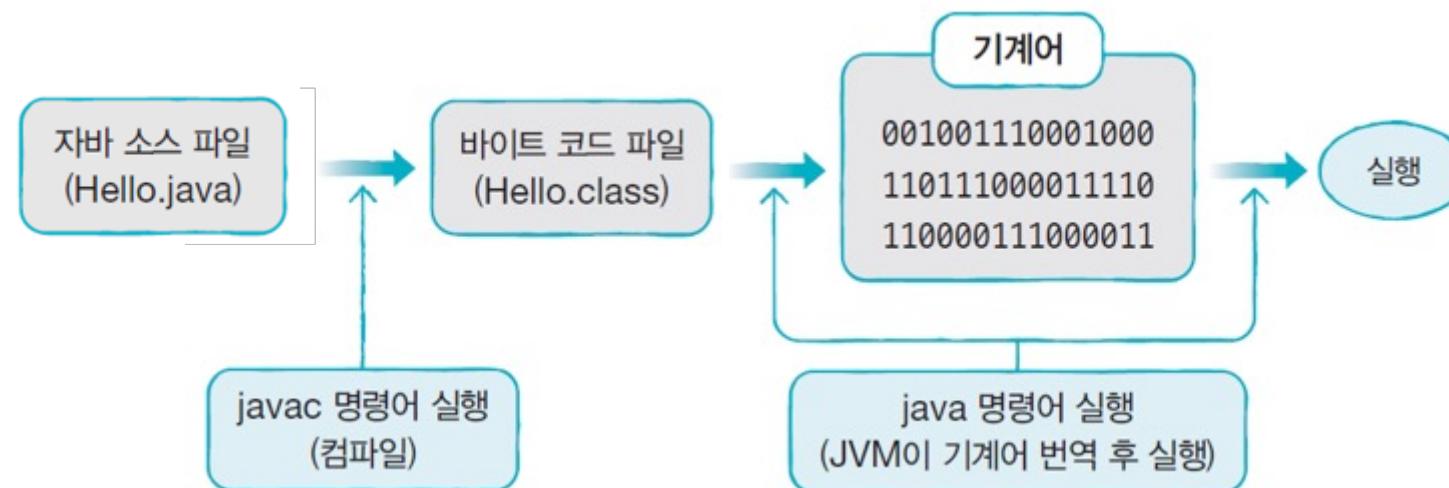


- JVM(Java Virtual Machine)은 바이트 코드 파일을 운영체제를 위한 완전한 기계어로 번역하고 실행하는 역할을 한다.



JVM (Java Virtual Machine)

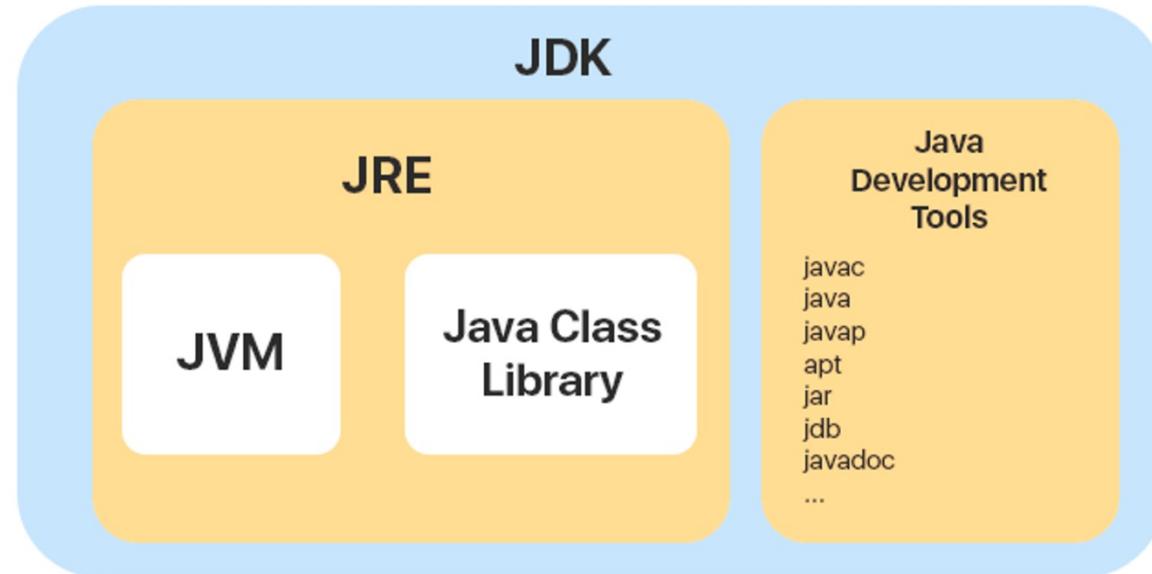
- Java 프로그램은 완전한 기계어가 아닌, 바이트 코드 파일(.class)로 구성
- 바이트 코드 파일은 운영체제에서 바로 실행할 수 없음
- 자바 가상기계(JVM: Java Virtual Machine)가 완전한 기계어로 번역하고 실행





JDK vs. JRE

- JDK: Java Development Kit (자바개발도구)
 - **JDK = JRE + 개발/디버깅 도구**
 - JDK를 설치하면 JRE, JVM이 자동으로 전부 설치
- JRE: Java Runtime Environment (자바 실행환경)
 - JRE = JVM + Java Class Library, Class Loader





JDK 설치하기

- 자바 개발 도구 (JDK : Java Development Kit): 자바 언어로 소프트웨어를 개발할 때 필요한 환경 및 도구를 제공하는 역할
- JDK 종류
 - Open JDK: <https://openjdk.java.net> (개발, 학습용 및 상업용 모두 무료로 사용)
 - Oracle JDK: <https://www.oracle.com> (개발, 학습용 무료로 사용, 상업용 목적은 연간 사용료 지불, 장기 기술지원 LTS)
- 학습용은 모두 무료이므로 안정적인 Oracle JDK를 사용하는 것을 권장

Java SE 11. 0. 2 (LTS)

주 버전 개선 버전 업데이트 버전 장기 지원 서비스 버전



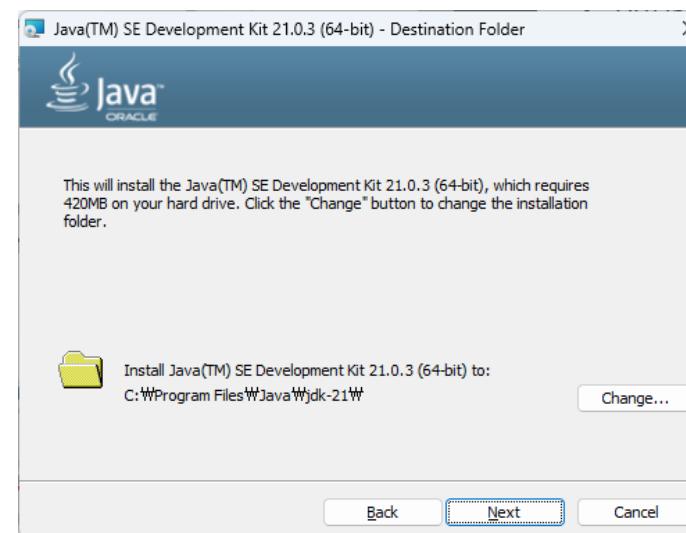
JDK 설치하기 (Windows)

- <https://www.oracle.com/kr/java/technologies/downloads/>
- 원하는 방식으로 설치
 - Compressed Archive는 압축 파일
 - installer는 프로그램을 바로 설치
 - MSI는 윈도우의 소프트웨어 설치, 유지, 제거를 위한 엔진으로 설치

JDK Development Kit 21.0.3 downloads

Linux macOS Windows

Product/file description	File size
x64 Compressed Archive	185.78 MB
x64 Installer	164.16 MB
x64 MSI Installer	162.91 MB





JDK 설치 확인하기 (Windows)

- 설치 완료 후, Terminal 창 새로 열기
 - Terminal 창: 명령 프롬프트, Windows PowerShell, Git bash
- java --version 입력 후, 정상적으로 설치되었는지 java의 버전 확인

```
Administrator@DESKTOP-A2ITP0C MINGW64 ~
$ java --version
java 21.0.3 2024-04-16 LTS
Java(TM) SE Runtime Environment (build 21.0.3+7-LTS-152)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.3+7-LTS-152, mixed mode, sharing)

Administrator@DESKTOP-A2ITP0C MINGW64 ~
$ |
```



JDK 설치하기 (macOS)

- <https://www.oracle.com/kr/java/technologies/downloads/>
- ARM64는 실리콘 칩, x64는 인텔 칩
- 원하는 방식으로 설치
 - Compressed Archive는 압축 파일
 - DMG Installer는 프로그램을 바로 설치



JDK Development Kit 21.0.3 downloads

Linux macOS Windows

Product/file description	File size
ARM64 Compressed Archive	182.13 MB
ARM64 DMG Installer	181.44 MB
x64 Compressed Archive	184.37 MB
x64 DMG Installer	183.69 MB



JDK 설치 확인하기 (macOS)

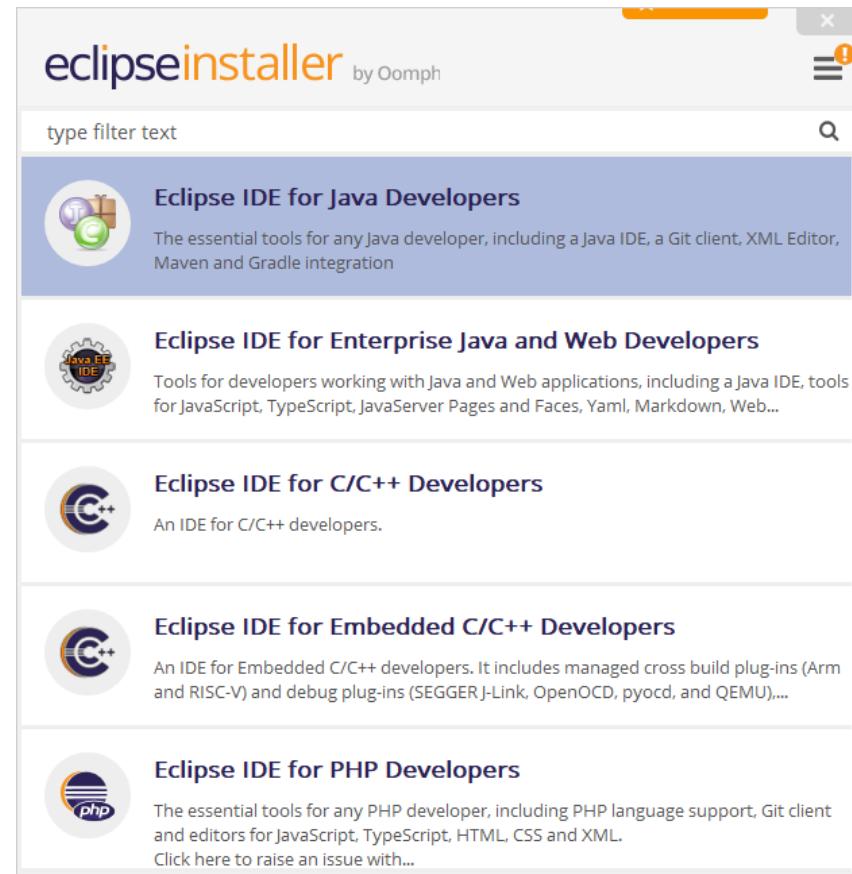
- 설치완료 후, Terminal 창 새로 열기
- java --version 입력 후, 정상적으로 설치되었는지 java의 버전 확인

```
Last login: Tue May 28 19:25:18 on ttys000
[inkyu@INKYUui-MacBookAir ~ % java --version
java 22.0.1 2024-04-16
Java(TM) SE Runtime Environment (build 22.0.1+8-16)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.1+8-16, mixed mode, sharing)
inkyu@INKYUui-MacBookAir ~ %
```



이클립스 설치하기

- <https://www.eclipse.org/downloads/>





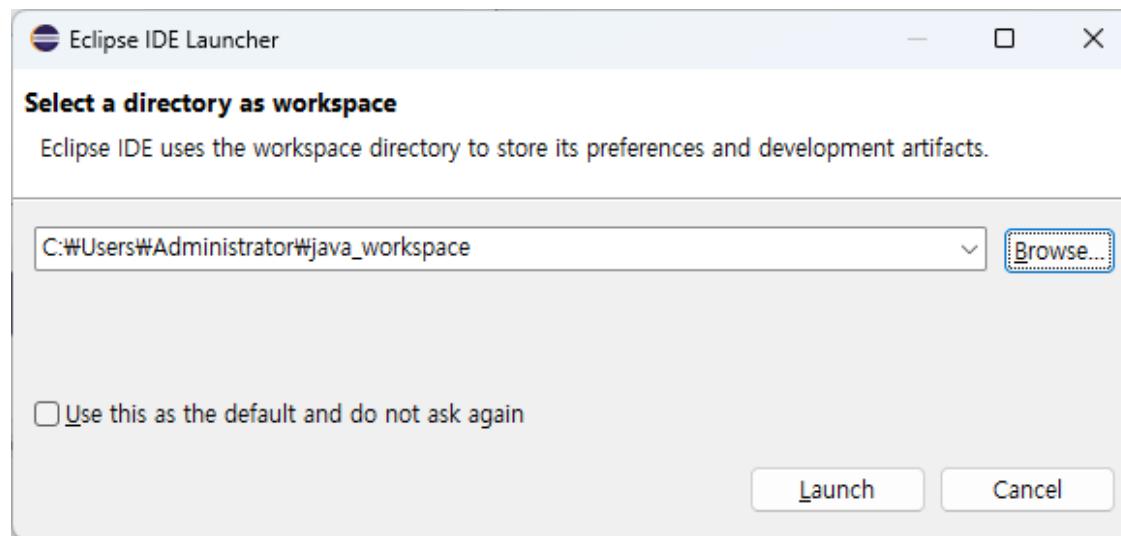
이클립스

- 통합 개발 환경(IDE: Integrated Development Environment)
 - 프로젝트 생성, 자동 코드 완성, 디버깅 등과 같이 개발에 필요한 여러 가지 기능을 통합적으로 제공해주는 툴
- 이클립스는 무료 오픈 소스 IDE로 기본적으로 자바 프로그램을 개발할 수 있도록 구성되어 있다.
 - 플러그인(plugin)을 설치하면 웹 애플리케이션 개발, C, C++ 애플리케이션 개발 등 다양한 개발 환경을 구축할 수 있다.
- 학습자뿐만 아니라 고급 개발자에 이르기까지 광범위하게 사용된다.



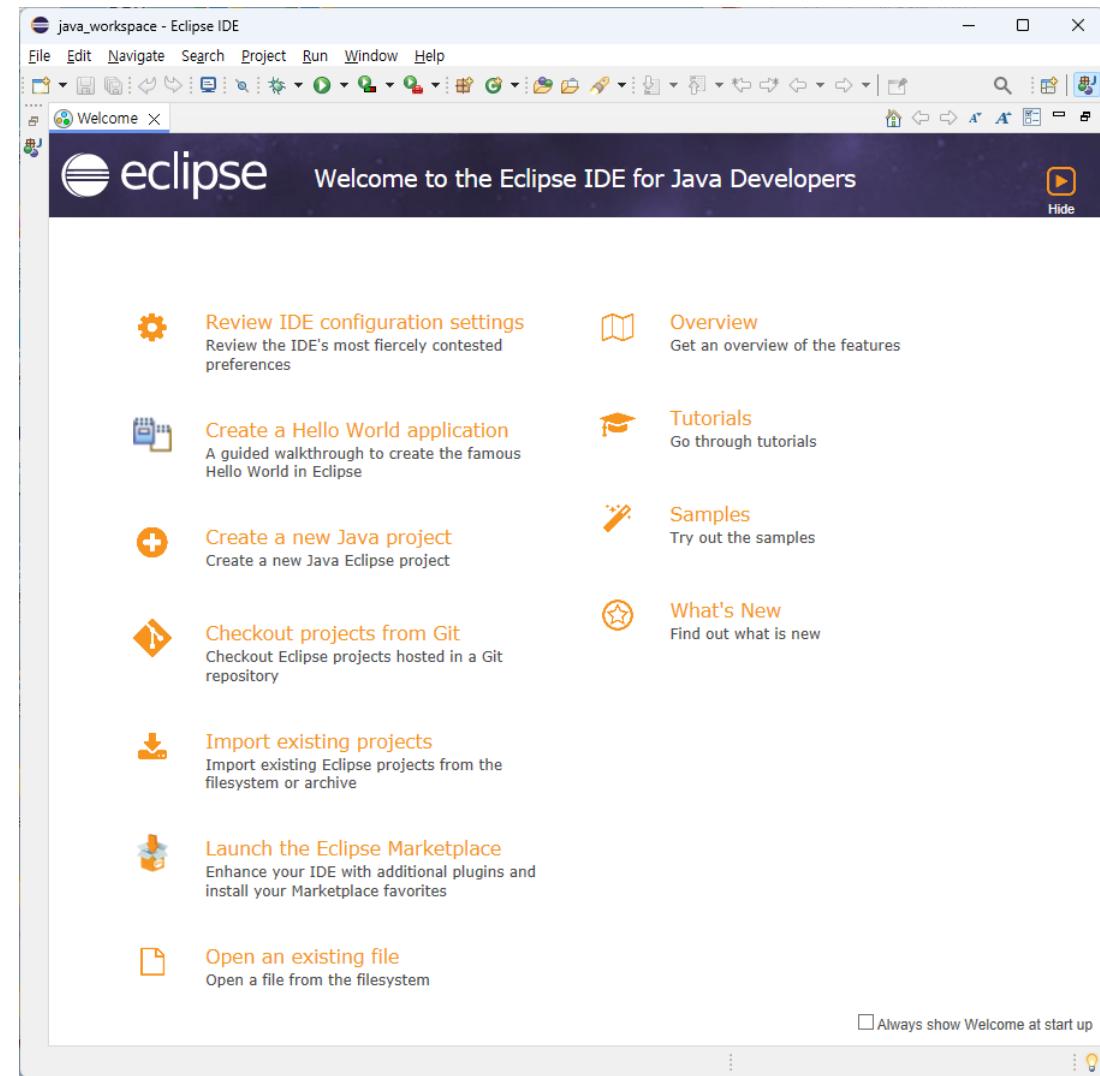
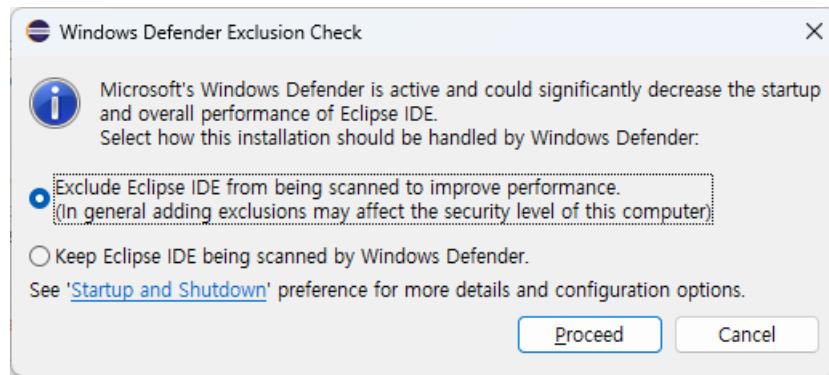
이클립스 구성 및 환경 설정하기

- 작업위치(workspace) 지정
 - 해당 디렉터리 내부에 작성한 코드, 프로젝트 폴더들이 저장된다.
 - 개발 환경 정보와 관련된 메타 데이터가 저장된 폴더(metadata)가 저장된다.





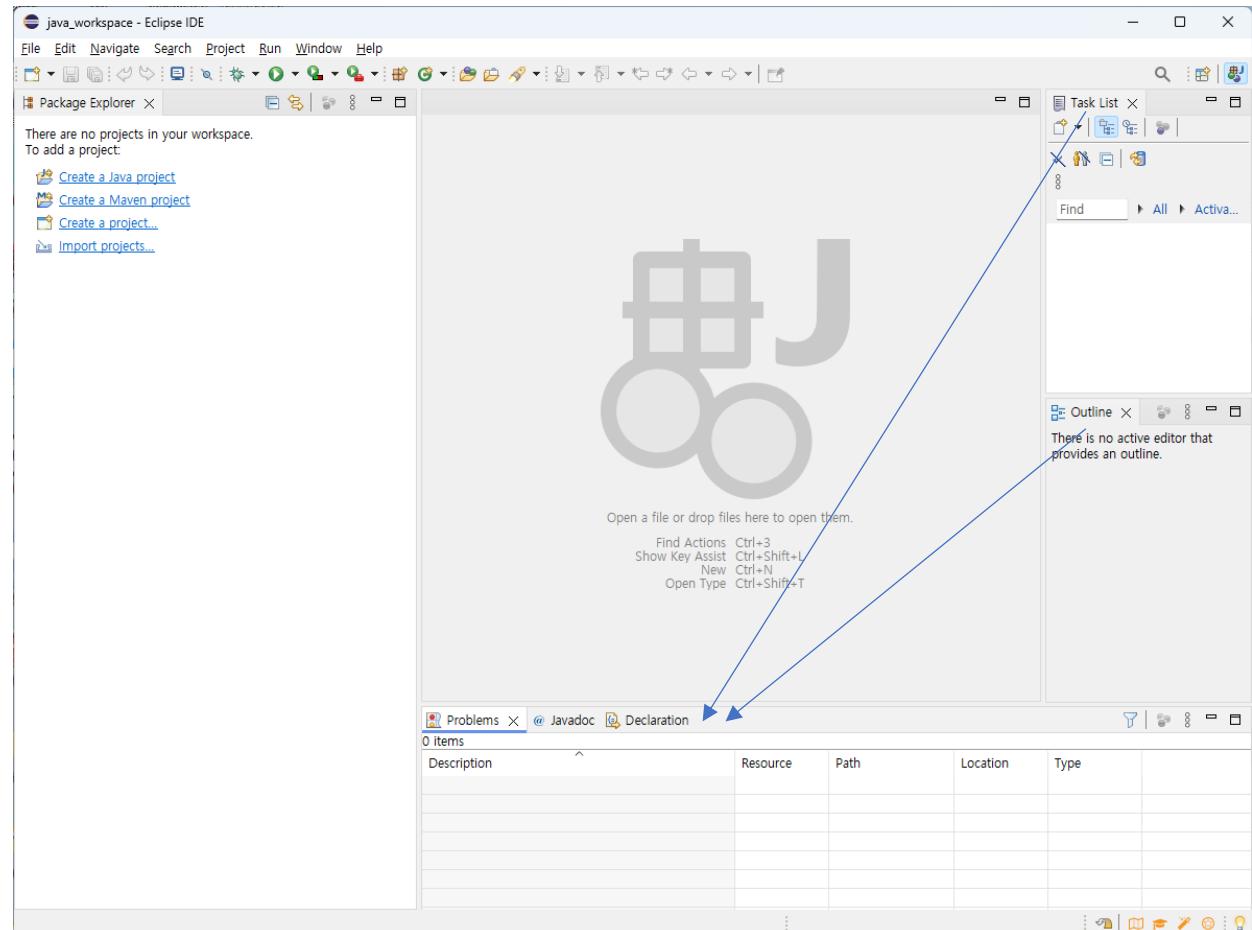
이클립스 구성 및 환경 설정하기



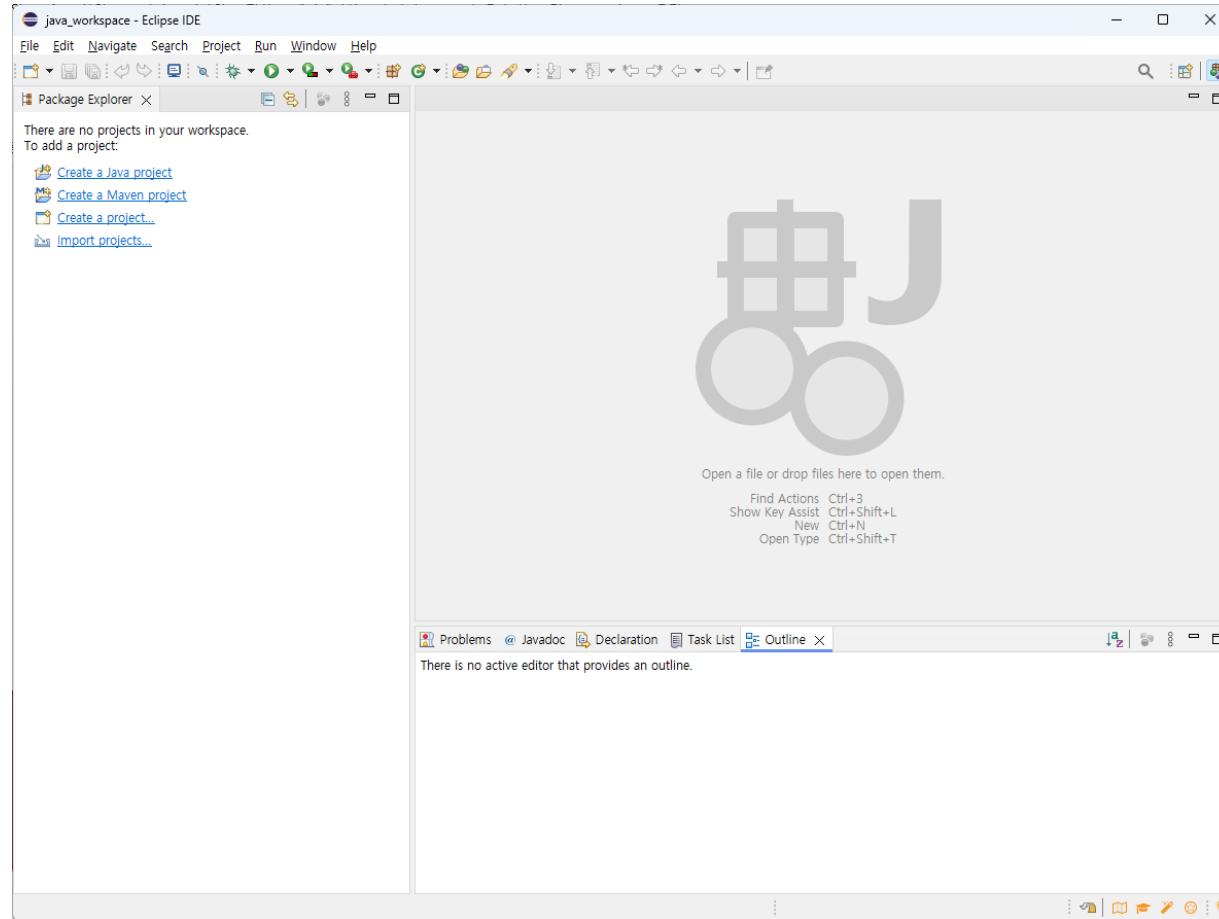


이클립스 구성 및 환경 설정하기

- 퍼스펙티브(Perspective)
 - 개발자가 특정 작업을 수행할 때 필요한 뷰(Views)와 편집기(Editors)의 배치를 정의하는 일종의 작업 환경
- 퍼스펙티브 설정
 - 우측 탭들을 하단으로 드래그앤파드랍



이클립스 구성 및 환경 설정하기





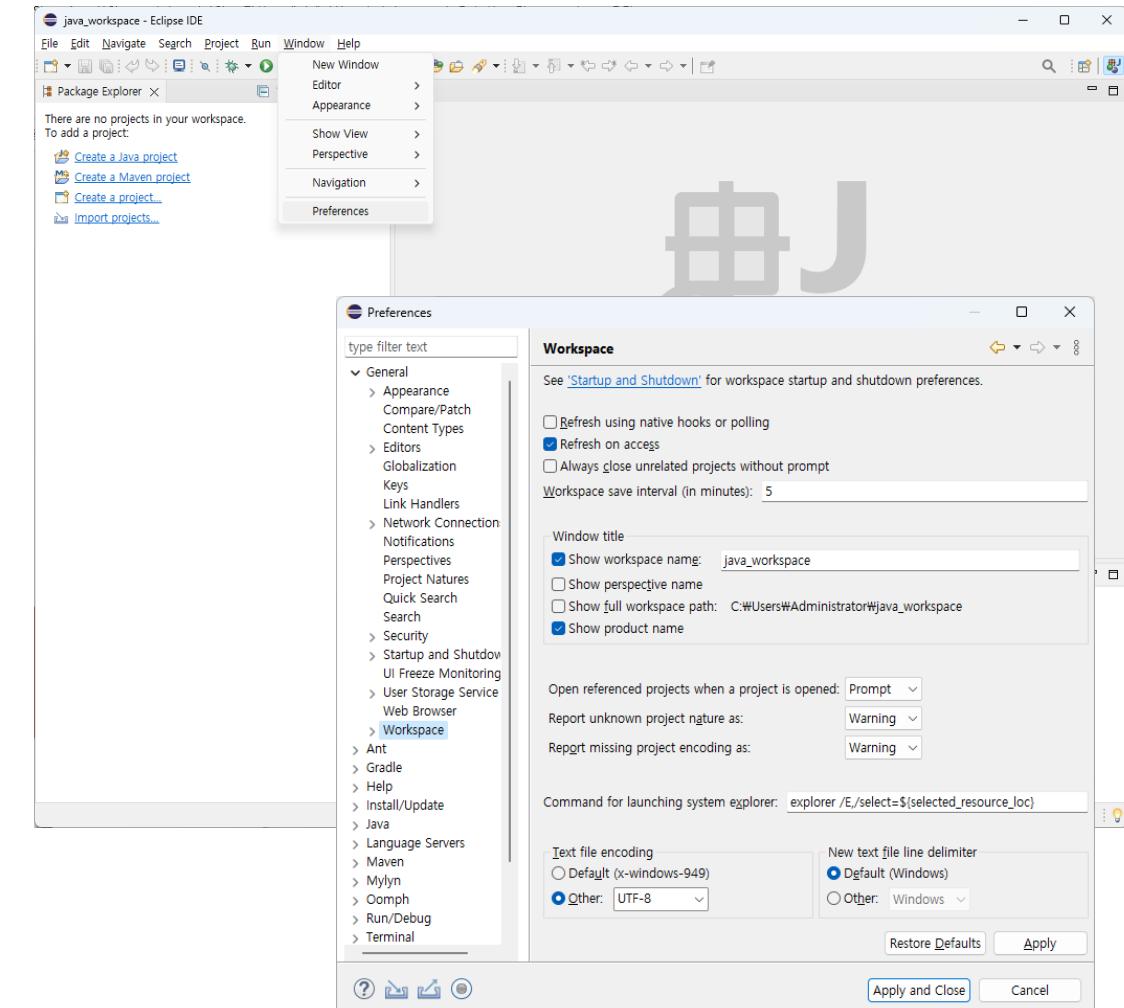
이클립스 구성 및 환경 설정하기

- Package Explorer
 - 프로젝트와 소스파일을 계층구조로 표시하는 뷰
 - 프로젝트 탐색
 - 폴더 및 파일 관리
 - 파일 열기 및 편집
- Declaration
 - 선택한 변수, 메서드, 클래스의 선언부 표시
 - 선언부 보기
 - 정의 위치로 이동
- Console
 - 프로그램의 출력을 표시
 - 표준 출력 및 오류
 - 디버그 정보 출력
- Problems
 - 프로젝트에서 발생한 오류와 경고 표시
 - 여러 및 경고 표시
 - 문제 위치로 이동
- Task List
 - 코드 내 TODO 주석으로 표시된 작업 목록 관리
 - 할 일 관리
 - 작업 우선 순위
 - 작업 위치로 이동
- Servers
 - 서버 관련 작업을 관리
 - 서버 관리, 서버 상태 모니터링
 - 배포 관리
- Javadoc
 - 선택한 클래스나 메서드의 Javadoc 주석 표시
 - API 문서 조회
 - 문서화된 설명 표시
- Outline
 - 현재 편집 중인 파일의 계층 구조 표시
 - 파일 구조 보기
 - 빠른 탐색
 - 코드 네비게이션



이클립스에서 다양한 설정 변경하기

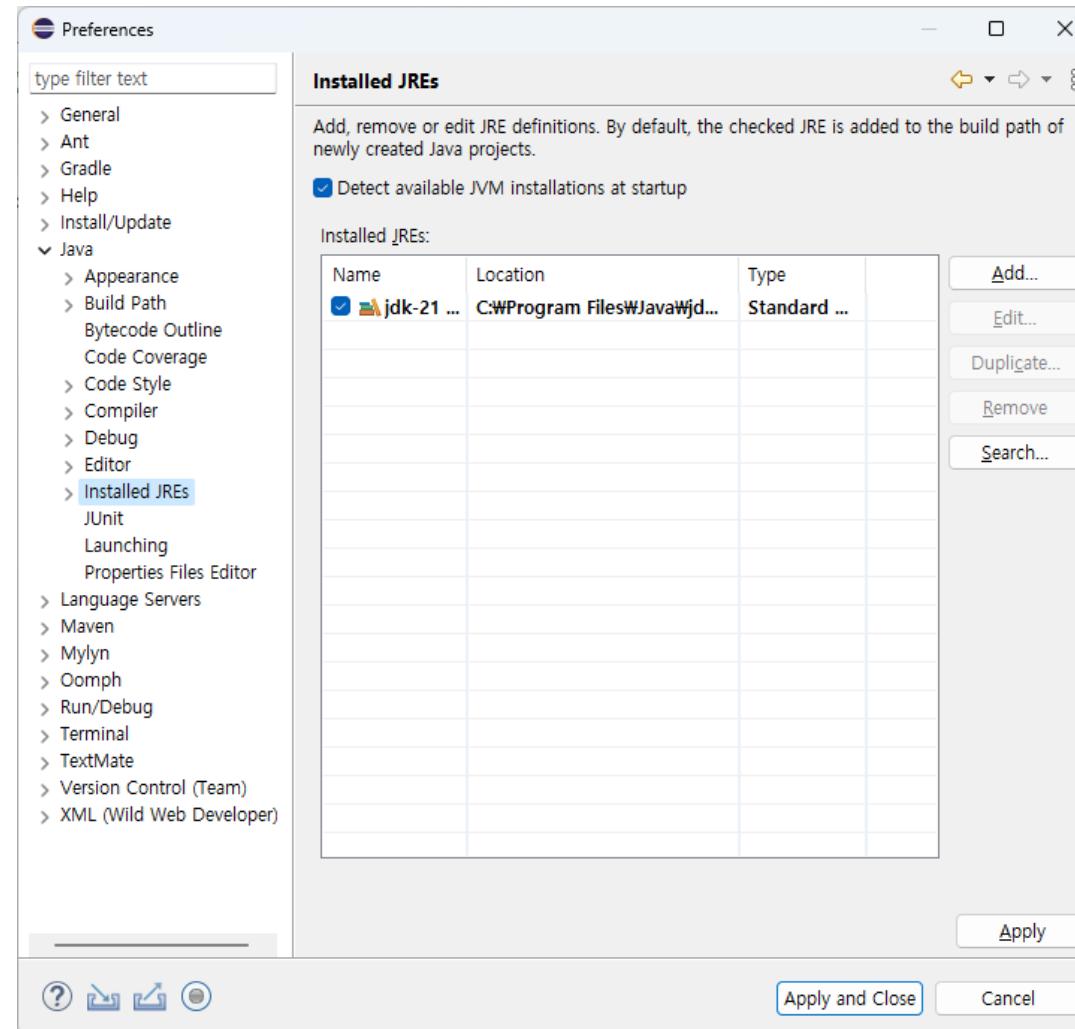
- 텍스트 인코딩 설정 (Workspace 단위)
 - Window - Preferences
 - General - Workspace - Text file encoding [UTF-8 설정]
- UTF-8:
 - 1바이트(8비트) 코드 단위를 사용.
 - 전 세계의 거의 모든 언어를 인코딩하는 전 세계 표준 인코딩 방식
 - 파일마다 인코딩하는 방법이 다르면 글자가 깨지는 현상이 발생





이클립스에서 다양한 설정 변경하기

- JRE 버전 확인
 - Window - Preferences
 - Java - Installed JREs





이클립스 단축키

주요
Function Key

F2	패키지/클래스 이름 바꾸기 (rename)
F3	클래스 및 함수 정의로 이동
F4	클래스 정의 확인 (상속관계 포함)

기능

[ctrl + shift + O]	자동 임포트
[ctrl + /]	한줄주석 설정 및 해제 (동일 방식)
[ctrl + shift + /] [ctrl + shift + W]	블록주석 설정 및 해제
(클래스 이름 선택 후) [ctrl + t]	상속관계 표현 (한번 더 ctrl + t 슈퍼타입 확인)

편집

[ctrl + shift + f]	자동 정렬
[alt + (+)] [alt + (-)]	블록 선택 시: 블록전체를 위아래로 이동 블록 미선택 시: 커서위치 라인을 위아래로 이동
(한글(ㄱ~ㅎ) 입력 후) [한자키]	특수문자 입력

보기

[ctrl + (+)] [ctrl + (-)]	폰트 확대 및 축소
[ctrl+shift + (-)] [ctrl+shift + (l)]	화면 가로 나누기 및 세로 나누기
[alt + (←)] [alt + (→)]	이전 자바파일 히스토리로 이동 (자바파일)



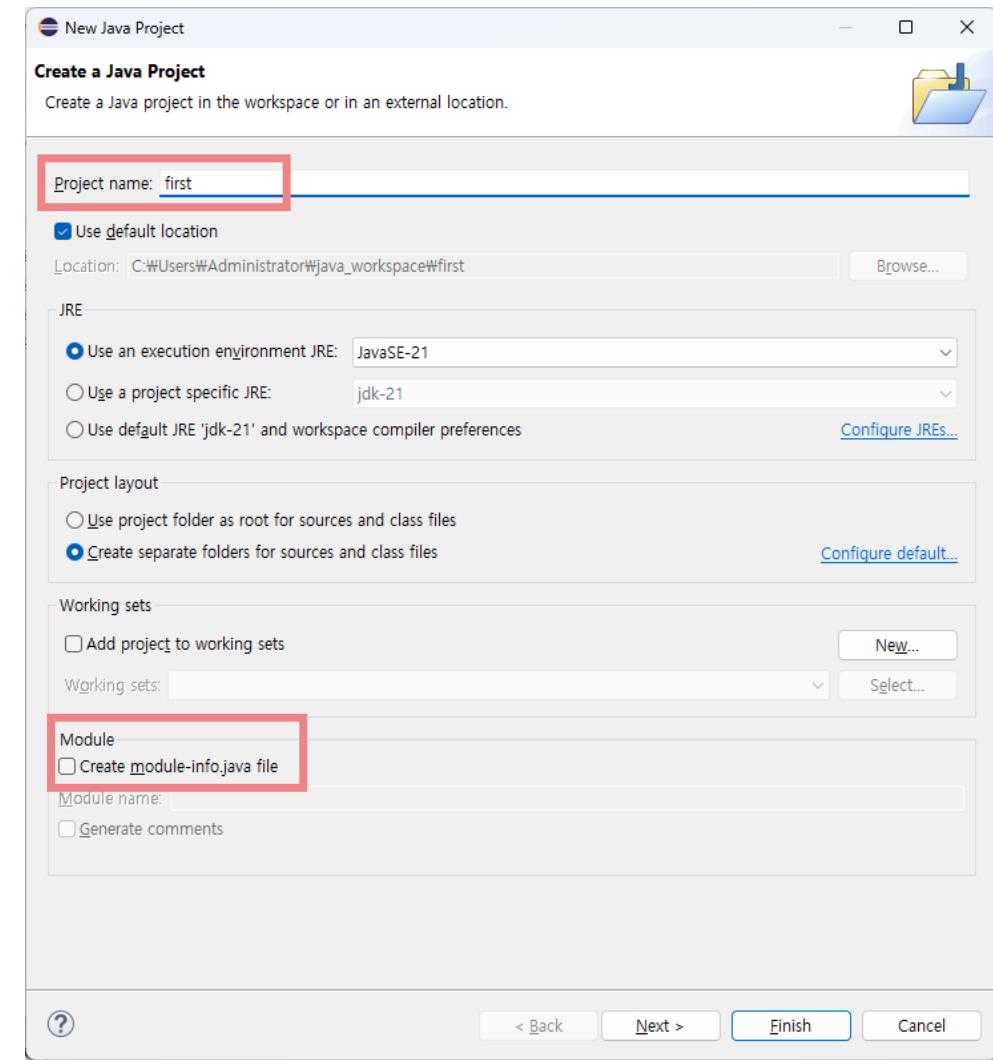
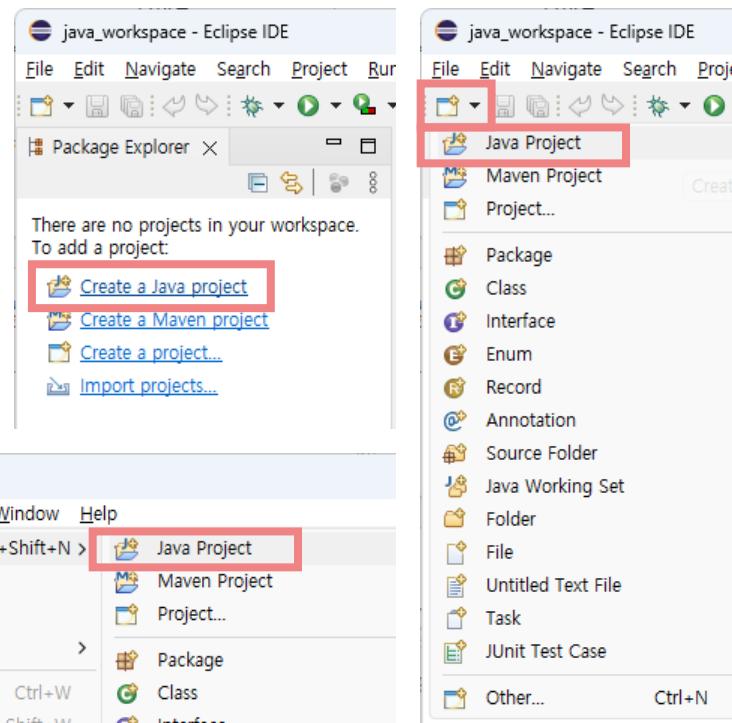
Java Project 시작하기

- STEP1. Java Project 생성 [프로젝트 폴더 생성]
- STEP2. 패키지 생성 [하위 폴더 생성]
- STEP3. Java Class 생성 [소스 파일 작성 .java 파일]
- STEP4. 저장(컴파일) [바이트 코드 생성 .class 파일]
- STEP5. 실행

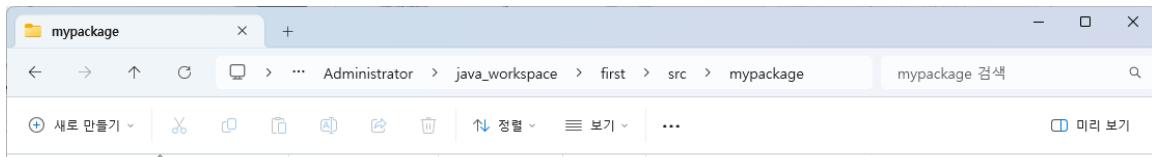
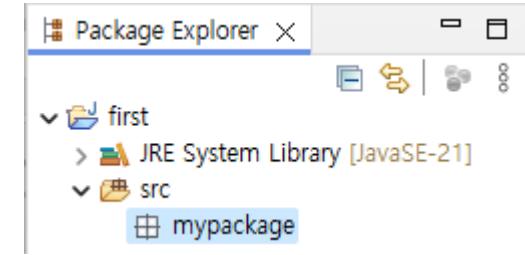
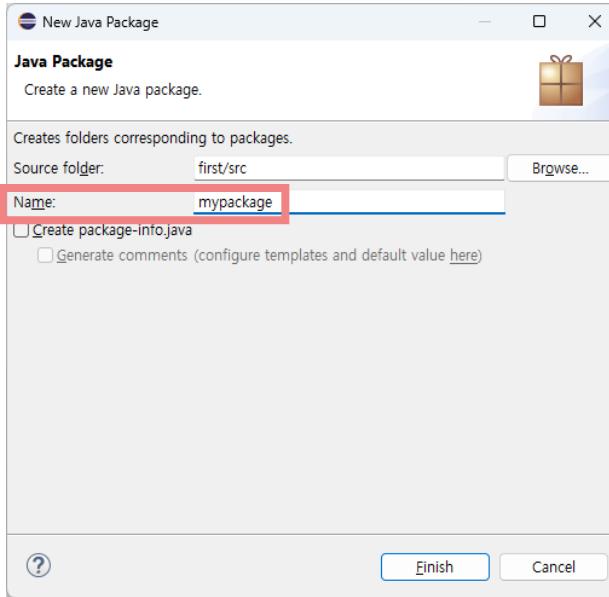
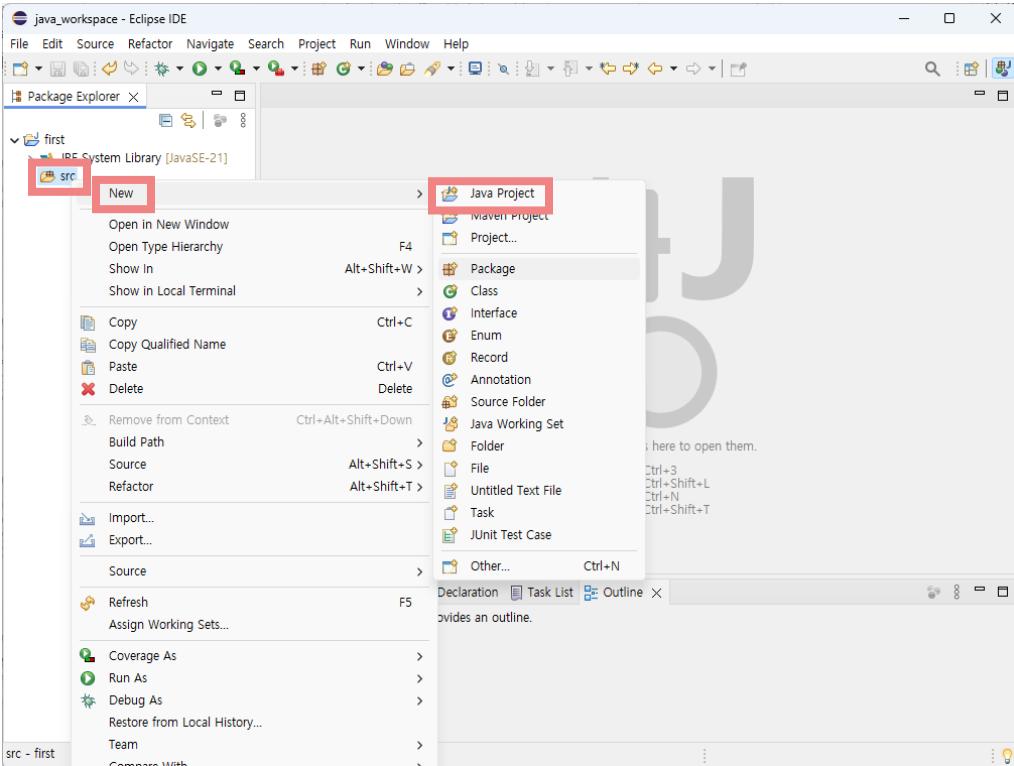
STEP1. Java Project 생성 [프로젝트 폴더 생성]

5

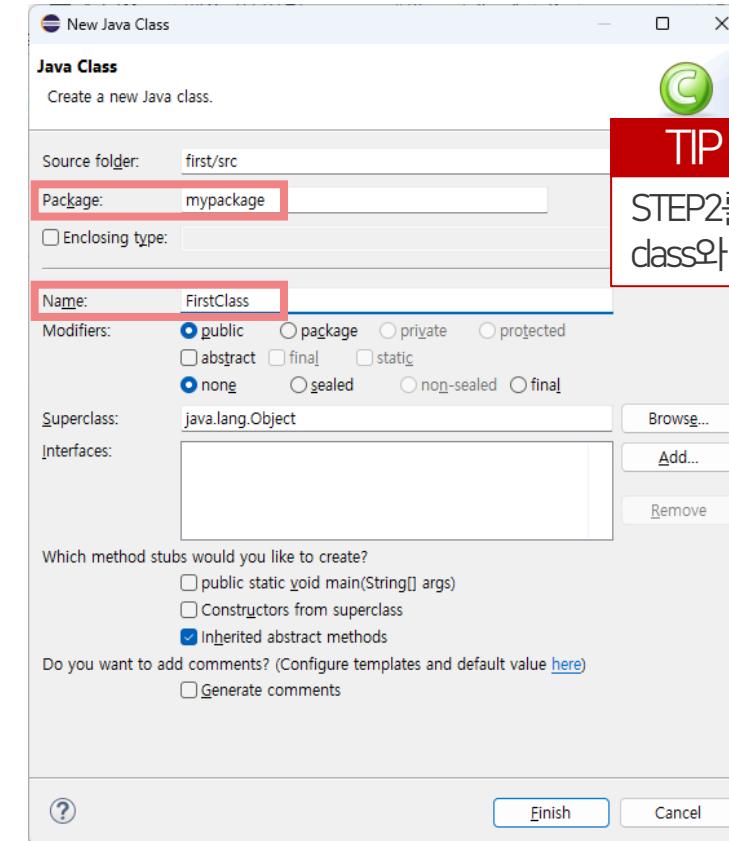
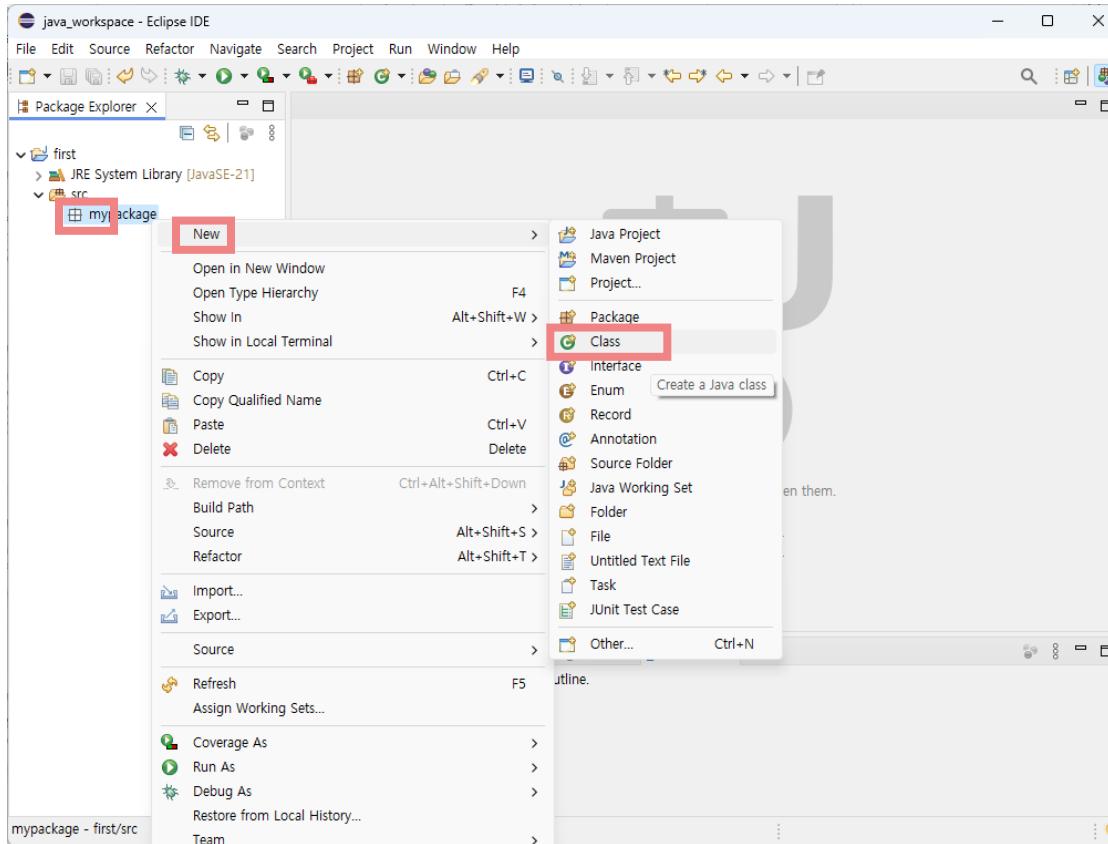
- Workspace : 여러 개의 프로젝트를 한 묶음으로 관리하는 곳
- Project : 하나의 실행
- Package : 비슷한 계열의 Class를 모아둔 파일의 묶음
- Class : 비슷한 유형의 메소드(함수)와 변수를 모아놓은 소스코드



STEP2. 패키지 생성 [하위 폴더 생성]

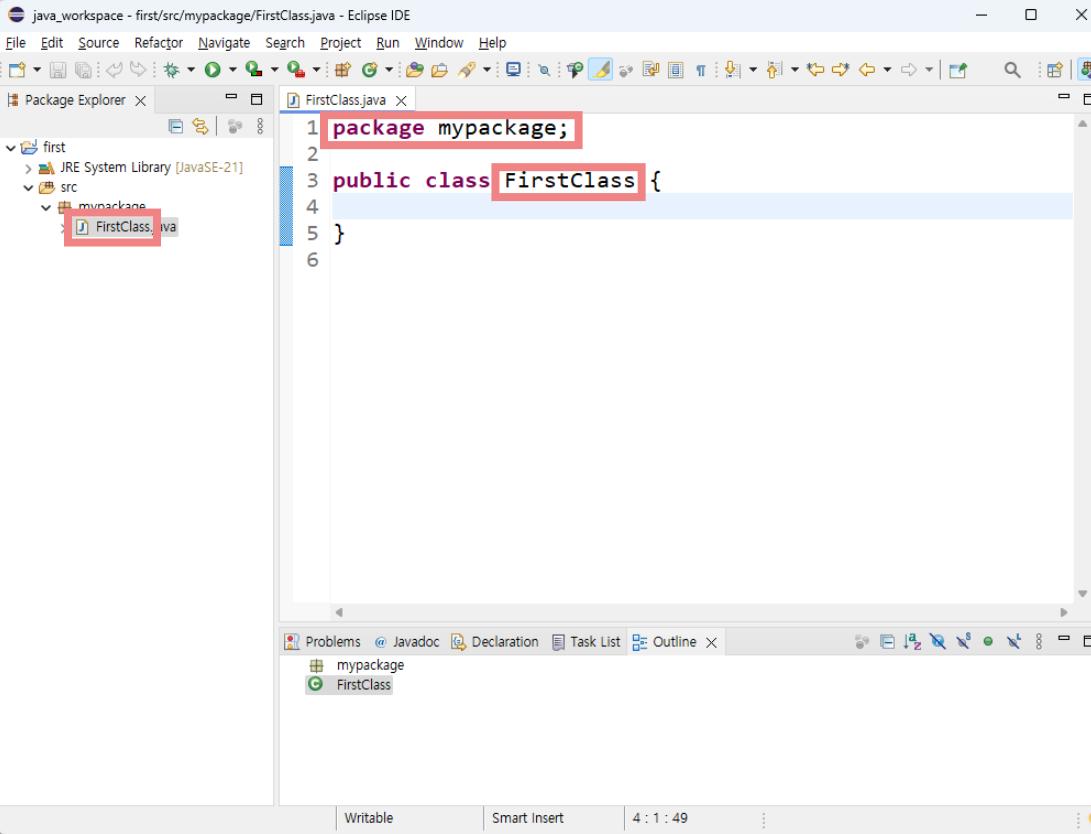


STEP3. Java Class 생성 [소스 파일 작성 .java 파일]



STEP2를 생략하고
class와 package 동시 생성 가능

STEP3. Java Class 생성 [소스 파일 작성 .java 파일]



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** java_workspace - first/src/mypackage/FirstClass.java - Eclipse IDE
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard Eclipse toolbar with various icons for file operations.
- Package Explorer:** Shows the project structure:
 - first
 - JRE System Library [JavaSE-21]
 - src
 - mypackage
 - FirstClass.java
- Editor:** Displays the code for FirstClass.java:

```
1 package mypackage;
2
3 public class FirstClass {
4
5 }
```
- Outline View:** Shows the declaration of the FirstClass class.
- Bottom Status Bar:** Writable, Smart Insert, 4 : 1 : 49

STEP3. Java Class 생성 [소스 파일 작성 .java 파일]

The screenshot shows the Eclipse IDE interface with the following details:

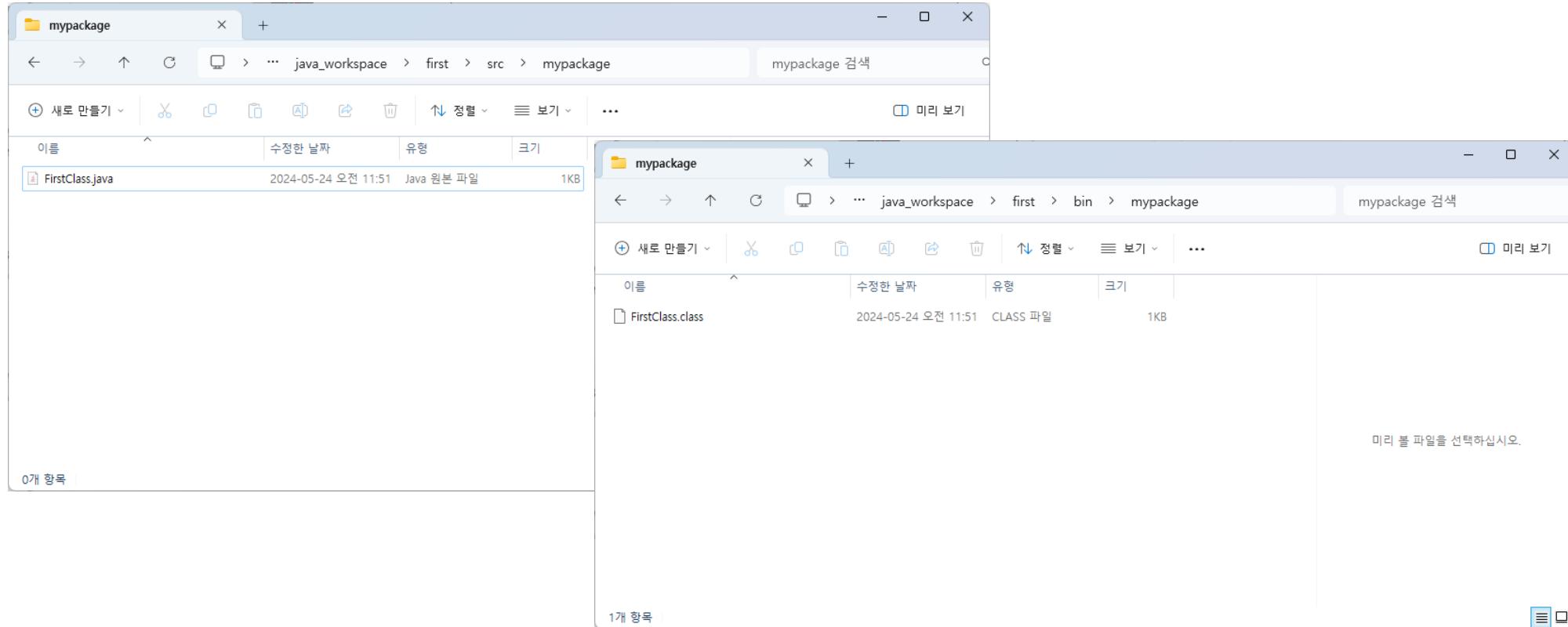
- Title Bar:** java_workspace - first/src/mypackage/FirstClass.java - Eclipse IDE
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard Eclipse toolbar icons.
- Package Explorer:** Shows the project structure:
 - first
 - > JRE System Library [JavaSE-21]
 - src
 - > mypackage
 - > FirstClass.java
- Editor:** Displays the Java code for FirstClass.java:

```
1 package mypackage;
2
3 public class FirstClass {
4     public static void main(String[] args) {
5         System.out.println("첫 프로젝트 생성");
6     }
7 }
8
```
- Outline View:** Shows the class structure:
 - mypackage
 - FirstClass
 - main(String[]) : void
- Bottom Status Bar:** Writable, Smart Insert, 5 : 41 : 126, etc.

TIP

main	CTRL+SPACE
sysout	CTRL+SPACE

STEP4. 저장(컴파일) [바이트 코드 생성 .class 파일]





STEP5. 실행

```
java_workspace - first/src/mypackage/FirstClass.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X FirstClass.java X
1 package mypackage;
2
3 public class FirstClass {
4     public static void main(String[] args) {
5         System.out.println("첫 프로젝트 생성");
6     }
7
8 }
```

첫 프로젝트 생성

TIP
CTRL+F11

java_workspace - first/src/mypackage/FirstClass.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project **Run** Window Help

Run F11

Debug F11

Coverage Ctrl+Shift+F11

Run History >

Run As > 1 Java Application Alt+Shift+X,J

Run Configurations...

Debug History >

Debug As >

Debug Configurations...

Coverage History >

Coverage As >

Coverage Configurations...

Toggle Breakpoint Ctrl+Shift+B

Toggle Lambda Entry Breakpoint

Toggle Tracepoint

Toggle Line Breakpoint

Toggle Watchpoint

Toggle Method Breakpoint

Skip All Breakpoints Ctrl+Alt+B

Remove All Breakpoints

Add Java Exception Breakpoint...

Add Class Load Breakpoint...

Problems Ctrl+Shift+N

All References... Ctrl+Shift+N

All Instances... Ctrl+Shift+N

Instance Count... Ctrl+Shift+N

Watch Ctrl+Shift+N

Inspect Ctrl+Shift+N

Display Ctrl+Shift+D

Execute Ctrl+U

Force Return Alt+Shift+F

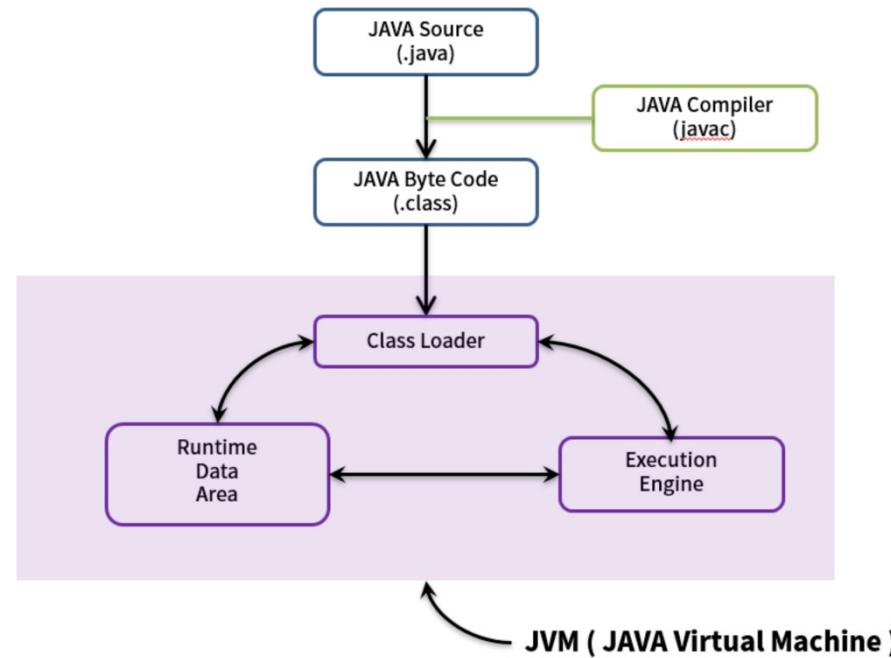
Console X C:\Program Files\Java\jdk-21\bin\javaw.exe (2024. 5. 24. 오후 1:34:40 – 오후 1:34:44) [pid: 1000]

Task List Outline



소스 코드의 실행 과정

1. 자바 소스 코드를 작성 [java]
2. 자바 컴파일러(Java Complier)가 소스 파일을 바이트 코드로 컴파일 [.class]
.class 파일은 컴퓨터가 읽을 수 있는 파일이 아니라 자바 가상 머신(JVM)에서 이해 가능한 코드
3. JVM 내의 클래스 로더(Class Loader)가 .class 를 메모리에 올리고 필요한 클래스들을 로딩
4. JVM 내의 실행 엔진(인터프리터와 JIT 컴파일러)이 컴퓨터가 읽을 수 있는 기계어 형태로 변환하여 실행





소스 코드의 실행 과정 [실습]

1. 바탕화면에 temp 폴더 생성
2. Hello.txt 파일 생성

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

줄 5, 열 2 | 118자 | 100% | Windows (CRLF) | UTF-8

3. .txt 확장자를 java로 변경
4. temp 폴더에서 Terminal 열고, javac Hello.java 입력

```
Administrator@DESKTOP-A2ITP0C MINGW64 ~/Desktop/temp  
$ javac Hello.java  
  
Administrator@DESKTOP-A2ITP0C MINGW64 ~/Desktop/temp  
$ |
```



소스 코드의 실행 과정 [실습]

5. temp 폴더에 .class 확장자의 파일이 생성된 것을 확인
6. Terminal에서 java Hello를 입력하여 실행

The screenshot shows a terminal window titled "MINGW64:/c/Users/Administrator/Desktop/temp". The command \$ java Hello is entered, followed by the output "Hello, World!". The terminal window has a standard title bar with minimize, maximize, and close buttons.

```
Administrator@DESKTOP-A2ITP0C MINGW64 ~/Desktop/temp
$ java Hello
Hello, World!

Administrator@DESKTOP-A2ITP0C MINGW64 ~/Desktop/temp
```



소스 코드의 기본 구조 분석

- 일반적인 소스 코드의 형태

```
/* 패키지 선언 */  
package commypackage.util;  
  
/* 패키지 가져오기 */  
import java.lang.*; // 해당 패키지 내 모든 클래스를 불러옴
```

```
/* 클래스 블록 */ 클래스 이름  
public class Main { 클래스 시작 중괄호  
    /* 메소드 블록 */ 메서드 이름  
    public static void main(String[] args) {  
        /* 명령문(statement) */  
        int result = add(1, 2);  
        System.out.println(result);  
    }  
}
```

```
/* 메소드 블록 */  
public static int add(int a, int b) {  
    return a + b;  
}
```

클래스 끝 중괄호

TIP

패키지를 지정하지 않은 경우 클래스 외부에는 아무것도 오지 않음

- 주석

/* 주석 내용 */ : 여러 줄 주석
// 주석 내용 : 한 줄 주석

- 클래스

```
public class Test {  
    ...  
}
```

- public: 접근지정자 키워드 (파일 당 최대 1개)
- class : 클래스 선언 키워드
- public 클래스 이름은 소스 파일 이름과 동일하게 작성

- 메소드

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

- public : 접근지정자 키워드
- static: 정적 메소드 키워드
- void : 반환 타입 지정
- 메소드 이름
- (매개변수)



소스 코드의 기본 구조 분석

```
/* 패키지 선언 */
package com.mypackage.util;

/* 패키지 가져오기 */
import java.lang.*; // 해당 패키지 내 모든 클래스를 불러옴

/* 클래스 블록 */
public class Main {
    /* 메소드 블록 */
    public static void main(String[] args) {
        /* 명령문(statement) */
        int result = add(1, 2);
        System.out.println(result);
    }

    /* 메소드 블록 */
    public static int add(int a, int b) {
        return a + b;
    }
}
```

소스 파일 생성

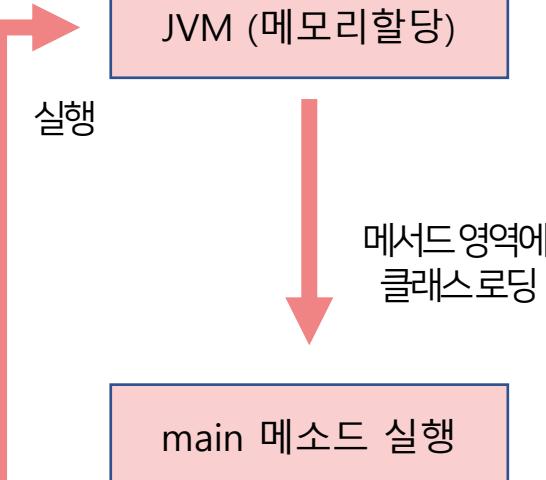
컴파일
(아클립스 저장시 자동 컴파일)

Main.class

바이트코드 생성

TIP

소스파일은 src 폴더내에 존재(java)
바이트코드는 bin 폴더에 존재(class)



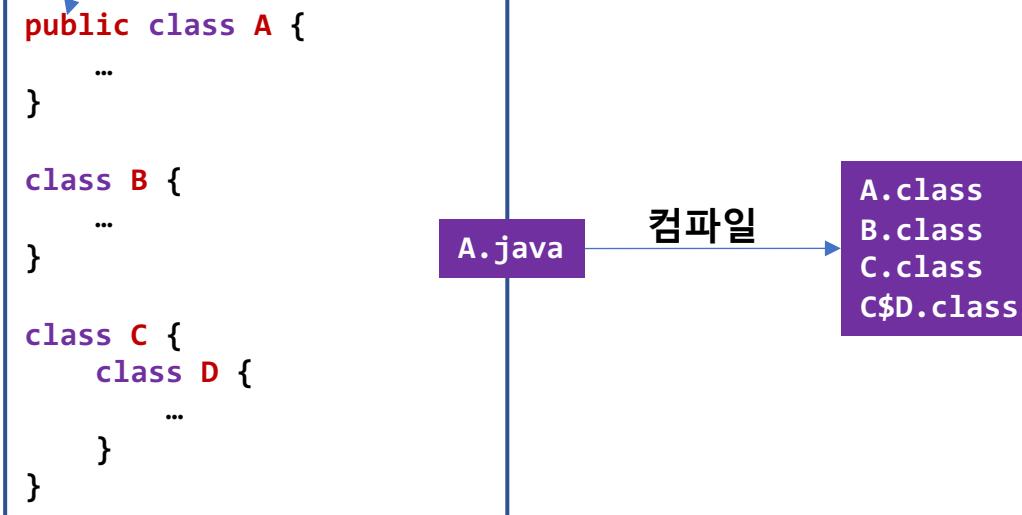
os(운영체제)

메소드 영역 | 스택 영역 | 힙 영역



컴파일과 바이트 코드 생성

하나의 소스파일(.java)에는 최대
하나의 public class만 선언 가능



CHECK 1

소스파일별 바이트코드 생성이 아닌 **클래스별 바이트코드(.class)가 생성됨**
(class 키워드 개수와 동일하게 생성)

CHECK 2

소스파일에는 public class가 최대 1개만 존재할 있으며
public class의 이름이 파일 이름과 동일하여야 함

CHECK 3

- 외부클래스(external class): **클래스이름.class**
- 내부클래스(inner class): **외부클래스이름\$내부클래스이름.class**



컴파일과 바이트 코드 생성 [실습]

The screenshot shows the Eclipse IDE interface. The top window is titled "workspace - practice01/src/practice01/A.java - Eclipse IDE". The code editor displays the following Java code:

```
1 package practice01;
2
3 public class A {
4
5 }
6
7 class B {
8
9 }
10
11 class C {
12
13     class D {
14
15     }
16
17 }
```

The "Package Explorer" view on the left shows a project structure with packages "practice01" and "src", and a source folder "src" containing "practice01". The file "A.java" is selected. The "Outline" view at the bottom shows nodes for classes A, B, C, and D.



프로젝트 이름: practice01
패키지 이름: practice01
클래스 이름: A.java



콘솔 출력 메서드와 문자열 출력 [실습]

TIP

문자열은 쌍따옴표(" ")안에만 표기 가능

TIP

String 자료형과의 '+' 연산

- String + String = String
- String + 기본자료형 = String
- 기본자료형 + String = String

콘솔 출력 System.out.xxx() 메서드와 문자열(String)

- **println()** 메서드: 괄호안의 내용 출력 + 줄바꿈

```
System.out.println("화면출력");
System.out.println("화면"+"출력");
System.out.println(3.8);
System.out.println(3+5);
System.out.println("화면"+3);
System.out.println("화면"+3+5);
System.out.println(3+5+"화면");
```

화면출력
화면출력
3.8
8
화면3
화면35
8화면

```
int a = 3;
String b = "화면";

System.out.println(a);
System.out.println(b);
System.out.println(b+"출력");
System.out.println(a+b+"출력");
```

3
화면
화면출력
3화면출력



콘솔 출력 메서드와 문자열 출력 [실습]

- **print() 메서드:** 괄호안의 내용 출력

```
System.out.print("화면");
System.out.print("출력");
System.out.print(3);
```

화면출력3

- **printf() 메서드:** 자료형 포맷에 따라 출력

```
System.out.printf("%d\n",30);(10진수)
System.out.printf("%o\n",30);(8진수)
System.out.printf("%x\n",30);(16진수)
```

30
36
1e

출력
5.800000
5.80
5.80
4와 5.80

```
System.out.printf("%s\n","출력");
System.out.printf("%f\n",5.8);
System.out.printf("%4.2f\n",5.8);
System.out.printf("%5.2f\n",5.8);
System.out.printf("%d와 %4.2f\n",4,5.8);
```



Java API Document

- 이미 제공되는 유용한 클래스들에 대한 사용 방법을 문서화하여 제공해주는 도움말과 같은 것

<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/module-summary.html>

The screenshot shows the Java API Documentation interface. The top navigation bar includes tabs for OVERVIEW, MODULE (which is highlighted in orange), PACKAGE, CLASS, USE, TREE, PREVIEW, NEW, DEPRECATED, INDEX, and HELP. To the right of the tabs, it says "Java SE 21 & JDK 21". Below the tabs, there's a search bar with a magnifying glass icon and the word "Search". The main content area has a header "MODULE: DESCRIPTION | MODULES | PACKAGES | SERVICES". On the left, there's a sidebar with a "Packages" section. Under "Exports", there's a table with columns for "Package" and "Description". The table lists several packages:

Package	Description
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.annotation	Provides library support for the Java programming language annotation facility.
java.lang.constant	Classes and interfaces to represent <i>nominal descriptors</i> for run-time entities such as classes or method handles, and classfile entities such as constant pool entries or <code>invokedynamic</code> call sites.
java.lang.foreign	Provides low-level access to memory and functions outside the Java runtime.
java.lang.invoke	The <code>java.lang.invoke</code> package provides low-level primitives for interacting with the Java Virtual Machine.
java.lang.module	Classes to support module descriptors and creating configurations of modules by means of resolution and service binding.
java.lang.ref	Provides reference-object classes, which support a limited degree of interaction with the garbage collector.
java.lang.reflect	Provides classes and interfaces for obtaining reflective information about classes and objects.

At the bottom of the page, there's a footer with the URL "docs.oracle.com/en/java/javase/21/docs/api/.../package-summary.html".



식별자

- 식별자: Java 코드 내에서 사용되는 각각의 단어
- 식별자 지정 규칙
 - Java는 대소문자를 구분한다.
 - 첫 문자는 반드시 영문자 또는 _, \$ 여야 한다.
 - 그 다음 문자부터는 숫자와 문자를 혼합하여 사용 가능하다.
- 식별자의 종류
 1. 시스템 정의 식별자: Java 시스템이 필요에 의해 먼저 정의해둔 식별자로 "예약어", "키워드"라고 부른다.
 2. 사용자 정의 식별자: 개발자가 필요에 의해 정의한 식별자로 "클래스명, 변수명, 메서드명"을 지정할 때 사용한다.



식별자

- 시스템 정의 식별자의 종류
 - Java 자체에서 특별한 의미를 가지는 식별자로, 시스템이 먼저 지정했기 때문에 사용자 지정 식별자로 사용이 불가하다.

분류	예약어
기본데이터타입	boolean, byte, char, short, int, long, float, double
접근지정자	private, protected, public
클래스관련	class, abstract, interface, extends, implements, enum
객체관련	new, instanceof, this, super, null
메소드관련	void, return
제어문관련	if, else, switch, case, default, for, do, while, break, continue
논리값	true, false
예외처리관련	try, catch, finally, throw, throws
기타	transient, volatile, package, import, synchronized, native, final, static, strictfp, assert



식별자

- 사용자 정의 식별자의 종류

분류	정의 규칙	사용 예
클래스	<ul style="list-style-type: none">첫 문자는 항상 대문자하나 이상의 단어가 합쳐질 때는 각 단어의 첫 문자들만 대문자로 표현(Pascal 표기)의미있는 명사형으로 지정	<pre>public class Student { ... }</pre>
변수와 메서드	<ul style="list-style-type: none">첫 문자는 항상 소문자하나 이상의 단어가 합쳐질 때는 각 단어의 첫 문자들만 대문자로 표현(Camel 표기)변수는 의미있는 명사형으로, 메서드는 의미있는 동사형으로 지정	<pre>String userId; public String getName() { ... }</pre>
상수	<ul style="list-style-type: none">모든 문자가 대문자하나 이상의 단어가 합쳐질 때는 under score(_)를 사용하여 연결의미있는 명사형으로 지정	<pre>final int BIRTH_YEAR = 1999;</pre>



데이터 타입

- Java가 처리할 수 있는 데이터의 종류를 Data Type이라고 하며, 크게 2가지로 나눈다.
 - 기본 데이터 타입 (Primitive Data Type, PDT) : 정수형, 실수형, 논리형, 문자형
 - 참조 데이터 타입 (Reference Data Type, RDT) : 기본 데이터 타입을 제외한 모든 데이터 타입



리터럴

- 리터럴(literal)은 Java 언어가 처리하는 실제 값을 의미한다.
 1. 문자
 - 하나의 문자를 의미하며, 반드시 작은따옴표로 표현한다.
 2. 문자열
 - 하나 이상의 문자나열을 의미하며, 반드시 큰따옴표로 표현한다.
 3. 정수
 - 음수, 양수, 0으로 구성된 정수 데이터이다.
 - 10진수, 2진수, 8진수, 16진수로 표현 가능하다.
 4. 실수
 - 소수점을 가진 실수 데이터이다.
 5. 논리
 - 참/거짓을 표현할 때 사용하는 논리 데이터이다.



변수

- 컴퓨터의 메모리(RAM)는 수많은 주소들로 구성되어 있는 데이터 저장 공간이다.
- 프로그램은 데이터를 메모리에 저장하고 읽는 작업이 반복되는데, 데이터를 어디에 어떤 방식으로 저장할지 결정하기 위해서는 변수를 사용해야 한다.
- 변수(Variable)은 하나의 값을 저장할 수 있는 메모리 주소에 붙여진 이름이다.
- 변수를 통해 프로그램은 메모리 주소에 값을 저장하고 읽을 수 있다.

주기억동 기억장소마을



- Java의 변수는 다양한 타입의 값을 저장할 수 없다.
- 즉, 정수형 변수에는 정수값만 저장할 수 있고, 실수형 변수에는 실수값만 저장 가능하다.



변수

- 다음과 같은 순서로 변수를 사용할 수 있다.

- 변수 선언
- 값 할당(초기화)
- 값 변경



변수 선언

- 변수를 사용하기 위해서는 변수를 선언하는 작업이 필요하다.
- 변수 선언 : 어떤 타입의 데이터를 어떤 이름으로 저장할지를 결정하는 것

데이터타입 변수명 ;

- 변수 이름 필수 규칙 : 첫번째 글자가 문자여야하고, 중간부터는 문자, 숫자, \$와 _를 포함할 수 있다.
- 일반적으로는 첫번째 글자를 소문자로 작성하는 Camel 케이스 방식을 사용하는 것이 관례이다.
- Java에서 사용되는 키워드는 변수명으로 사용이 불가하다.

- 변수를 선언함과 동시에 값을 저장할 수 있는데, 이 때 대입 연산자(=)을 사용한다.
- 어떤 값을 저장하고 있는지 쉽게 알 수 있도록 의미 있는 변수 이름이 좋다.
- 변수는 자신이 선언된 블록(중괄호 기준) 안에서만 사용 가능하다.

TIP

변수명 : 소문자 시작이 관례
score
mathScore
greatStudentName
클래스명 : 대문자 시작이 관례
Week.java
MemberGrade.java



변수 선언과 대입

- 변수의 선언은 저장할 데이터의 형식(Type)과 이름만을 결정한 것이지, 메모리에 저장된 것은 아니다.
 - 변수에 최초로 값이 대입될 때 메모리에 할당되고, 해당 메모리에 값이 저장된다.
-
- 변수에 최초로 값을 대입하는 행위를 [변수 초기화]라고 하고, 이 때의 값을 초기값이라고 한다.
 - 변수의 선언과 동시에 초기화를 할 수 있다.

```
int score; // 변수 선언  
score = 90; // 변수 초기화
```

TIP

int: 정수형 데이터 타입

```
int myScore = 95; // 변수의 선언과 초기화를 동시에
```

- 같은 타입의 변수는 봄매[]를 이용해 한꺼번에 선언도 가능하다.
- 초기화되지 않은 변수를 사용할 경우에러: **The local variable score may not have been initialized**



변수의 사용 범위

- 변수는 블록({ }) 내부에서 선언되고 사용된다. 이를 블록 scope라고 한다.
 - 지역변수(local variable) : 메소드 블록 내부에서 선언되어, 메소드 블록 내부에서 사용되는 변수
 - 멤버 변수(member variable) : 클래스 블록 내부에서 선언되어, 클래스 블록 내부에서 사용되는 변수 (필드, 속성)
 - 인스턴스 변수(instance variable) : static이 붙지 않은 변수로 외부에서 사용할 때에는 반드시 new 연산자로 객체를 생성해서 사용
 - 클래스 변수(static variable) : 객체를 생성하지 않고도 접근이 가능한 변수



상수

- 값이 변경될 수 없는 것을 상수라고 하며, final 키워드를 사용하면 변수를 상수로 만들 수 있다.
- 일반적으로 변수명과 구분하기 위해 상수명은 대문자로 설정한다.

```
final int BIRTH_YEAR = 1999;
```



변수 값 교환하기(Swap) [실습]

```
package practice01;

public class VariableExchange {
    public static void main(String[] args) {
        int x = 3;
        int y = 5;
        System.out.println("x: " + x + ", y: " + y);

        // x와 y값을 교환하기
        x = y;
        y = x;
    }
}
```



데이터 타입

- 변수는 선언 시 정해지는 타입에 따라 저장할 수 있는 값의 종류와 허용 범위가 달라진다.
- Java에서 기본 데이터 타입(Primitive Data Type) 8개를 제공한다.
- Java의 자료형(Data Type)
 - 기본 자료형
 - 정수 : byte, char, short, int, long
 - 실수 : float, double
 - 논리(true, false) : boolean
 - 참조 자료형
 - 배열, 클래스 / 인터페이스

TIP

일반적으로 기본 자료형은 소문자로 시작하고, 참조 자료형은 대문자로 시작

TIP

기본 자료형은 값을 stack 메모리에 저장(직접 값을 보유)
참조 자료형은 값은 heap 메모리에 저장(참조하고 있는 메모리 주소를 보유)

TIP

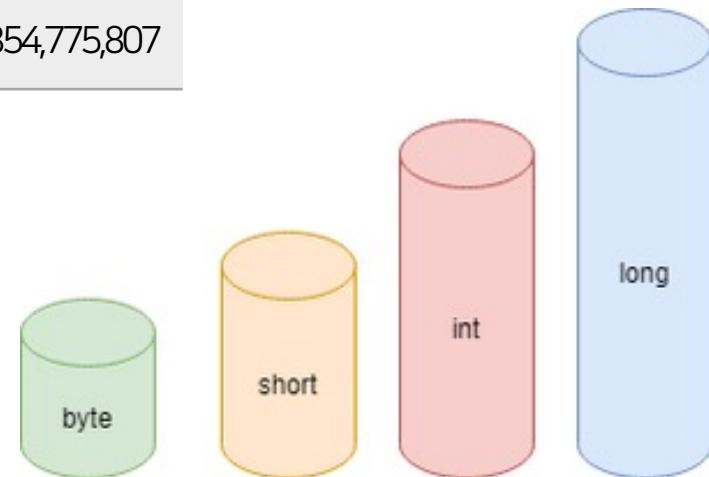
참조 자료형은 직접 정의할 수 있기 때문에 무한개가 존재



정수 데이터 타입

타입	메모리 크기		저장되는 값의 허용 범위	
byte	1byte	8bit	$-2^7 \sim (2^7 - 1)$	-128~127
short	2byte	16bit	$-2^{15} \sim (2^{15} - 1)$	-32,768~32,767
char	2byte	16bit	$0 \sim (2^{16} - 1)$	0~65535
int	4byte	32bit	$-2^{31} \sim (2^{31} - 1)$	-2,147,483,648~2,147,483,647
long	8byte	64bit	$-2^{63} \sim (2^{63} - 1)$	-9,223,372,036,854,775,808~9,223,372,036,854,775,807

종류	byte	short	int	long
메모리 사용 크기 (단위: bit)	8	16	32	64





정수 데이터 타입 [실습]

- 기본적으로 컴파일러는 정수 리터럴을 int 타입 값으로 간주한다.
- 따라서 int 타입의 허용 범위를 초과하는 리터럴은 뒤에 [I 또는 L]을 붙여 long 타입의 값임을 표시해야 한다.

```
long var1 = 10;  
long var2 = 20L;  
long var3 = 2_147_483_647;  
long var4 = 2147483648L;
```

```
System.out.println(var1);  
System.out.println(var2);  
System.out.println(var3);  
System.out.println(var4);
```

TIP

리터럴: 코드상에서 데이터를 표현하는 방식(데이터값 그 자체)



정수 데이터 타입

2진수: 0b 또는 0B로 시작하고 0과 1로 구성됩니다.

0b1011

$$\rightarrow 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \rightarrow 11$$

0b10100

$$\rightarrow 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \rightarrow 20$$

8진수: 0으로 시작하고 0~7 숫자로 구성됩니다.

013

$$\rightarrow 1 \times 8^1 + 3 \times 8^0 \rightarrow 11$$

0206

$$\rightarrow 2 \times 8^2 + 0 \times 8^1 + 6 \times 8^0 \rightarrow 134$$

10진수: 소수점이 없는 0~9 숫자로 구성됩니다.

12

365

16진수: 0x 또는 0X로 시작하고 0~9 숫자와 A, B, C, D, E, F 또는 a, b, c, d, e, f로 구성됩니다.

0xB3

$$\rightarrow 11 \times 16^1 + 3 \times 16^0 \rightarrow 179$$

0x2A0F

$$\rightarrow 2 \times 16^3 + 10 \times 16^2 + 0 \times 16^1 + 15 \times 16^0 \rightarrow 10767$$



문자 데이터 타입

- 하나의 문자는 작은 따옴표로 감싸 표시하며, 이를 문자 리터럴이라 한다.
 - 문자 리터럴은 유니코드로 변환되어 저장된다.
 - 유니코드 : 세계 각국의 문자를 0~65535 숫자로 매팅한 국제 표준 규약
-
- Java는 유니코드를 저장할 수 있도록 char 타입을 제공한다.

```
char var1 = 'A'; // 'A' 문자 : 65로 대입  
char var2 = '가'; // '가' 문자 : 44032로 대
```

- 유니코드가 정수이므로 char 타입도 정수 타입에 속한다.
- 따라서 작은 따옴표로 감싼 문자가 아니라 유니코드 숫자를 직접 대입할 수도 있다.
- ''와 같이 빈 문자열은 컴파일 에러가 발생한다.
- 공백 문자는 유니코드:32 또는 공백 하나를 포함한 문자(' ')을 사용해야 한다.

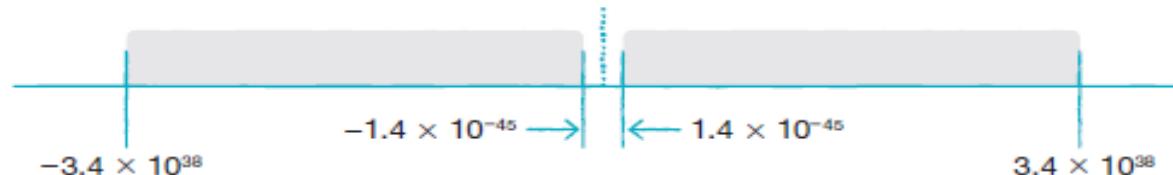


실수 데이터 타입

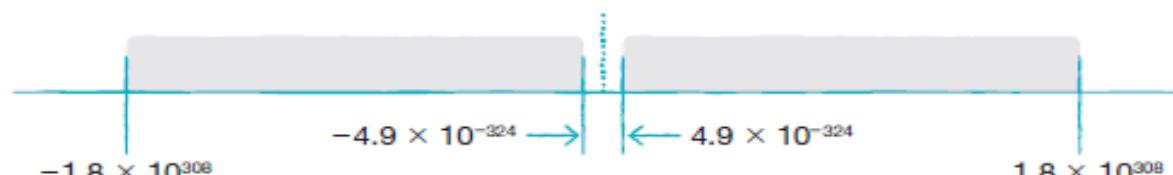
타입	메모리 크기	저장되는 값의 허용 범위 (양수 기준)
float	4byte	$1.4 \times 10^{-45} \sim 3.4 \times 10^{-38}$ 소수점이하 7자리
double	8byte	$4.9 \times 10^{-324} \sim 1.8 \times 10^{-308}$ 소수점이하 15자리

- 기본적으로 컴파일러는 실수 리터럴을 double 타입 값으로 간주한다.
- 따라서 float 타입의 허용 범위를 초과하는 리터럴은 뒤에 [f 또는 F]을 붙여 float 타입의 값임을 표시해야 한다.

float



double





논리 데이터 타입

- 참과 거짓을 의미하는 논리 리터럴은 true와 false이다.
- 논리 리터럴은 boolean 타입 변수에 대입 가능하다.
- boolean 타입 변수는 주로 두 가지 상태값을 저장할 필요가 있을 때 사용하며, 상태값에 따라 조건문과 제어문의 실행 흐름을 변경하는데 사용된다.
- 비교 및 논리 연산의 결과값은 true 또는 false를 가지므로, boolean 타입 변수에 대입할 수 있다.



문자열 데이터 타입 [실습]

- 한글자의 문자는 작은 따옴표로 감싼 char 타입이지만, 여러 개의 문자들은 큰 따옴표로 감싼 문자열 타입을 사용해야 한다.
- 문자열 타입은 String 타입이며, Java에서 제공하는 기본 데이터 타입(Primitive Data Type)이 아니라 참조 데이터 타입이다.

```
String name = "홍길동";
String job = "프로그래머";
System.out.println(name);
System.out.println(job);
```

```
String str = "나는 \"자바\"를 배웁니다.";
System.out.println(str);
```

```
str = "번호\t이름\t직업";
System.out.println(str);
```

```
System.out.print("나는\n");
System.out.print("자바를\n");
System.out.print("배웁니다.");
```



문자열 데이터 타입

- 문자열 내부의 [\,₩]는 이스케이프 문자를 뜻함
 - 이스케이프 문자를 사용하면 특정 문자를 포함시키거나, 문자열의 출력을 제어할 수 있음

문자열 내부에 "문자 포함"

```
String str = "나는 \"자바\"를 좋아합니다.";  
System.out.println(str);
```

→ 나는 "자바"를 좋아합니다.

문자열 출력 제어(Tap)

```
String str = "번호\t이름\t나이";
System.out.println(str);
```

문자열 출력 제어 (줄바꿈)

```
String str = "홍길동\n갑자바";  
System.out.println(str);
```

→ 흥길동
갈자바

이스케이프문자	출력용도
\t	탭만큼띄움
\n	줄바꿈(라인피드)
\r	캐리지리턴
\", \', \\	";\" 출력
\u16진수	16진수유니코드에 해당하는 문자 출력



문자열 데이터 타입 [실습]

- Java 13부터는 텍스트 블록 문법을 제공한다.
- 큰따옴표 3개로 감싸면 작성된 그대로 문자열로 저장된다.
- 줄바꿈을 하지 않고 이어서 작성하고 싶으면 맨 끝에 \를 붙여주면 된다.

```
String str1 = "" +  
    "{\n" +  
    "\t\"id\": \"winter\", \n" +  
    "\t\"name\": \"눈송이\", \n" +  
    "}";  
System.out.println(str1);  
  
String str2 = """"  
{  
    "id": "winter",  
    "name": "눈송이"  
}  
""";  
System.out.println(str2);  
  
String str3 = """"  
나는 자바를 \  
학습합니다,  
나는 자바 고수가 될 겁니다.  
""";  
System.out.println(str3);
```



[정리] 데이터 타입

- 정수 타입: 정수를 저장할 수 있는 타입으로 byte, short, int, long 타입
- 실수 타입: 실수를 저장할 수 있는 타입으로 float, double 타입을 말함
- char 타입: 작은따옴표(')로 감싼 하나의 문자 리터럴을 저장할 수 있는 타입
- boolean 타입: 참과 거짓을 의미하는 true와 false를 저장할 수 있는 타입
- String 타입: 큰따옴표(")로 감싼 문자열을 저장할 수 있는 타입



자동 타입 변환 (Promotion)

- 자동 타입 변환은 자동으로 데이터 타입의 변환이 일어나는 것을 말한다.
- 자동 타입 변환은 값의 허용 범위가 작은 타입이 허용 범위가 큰 타입으로 대입될 때 발생한다.
- 기본 타입을 허용 범위 순으로 나열하면 아래와 같다.
 - byte < short, char < int < long < float < double
- 즉, int 타입이 byte 타입보다 허용 범위가 더 크기 때문에 자동 타입 변환이 된다.

```
byte byteValue = 10;  
int intValue = byteValue;
```

00001010

00000000

00000000

00000000

00001010



자동 타입 변환 (Promotion)

- 같은 원리로 정수 타입이 실수 타입으로 대입될 경우에는
실수 타입이 정수 타입보다 허용 범위가 더 크기 때문에 무조건 자동 타입 변환이 된다.

```
long longValue = 5_000_000_000L;
```

```
float floatValue = longValue;
```

```
double doubleValue = longValue;
```

```
char charValue = 'A';
```

```
int intValue = charValue;
```

- 자동 타입 변환에 한 가지 예외가 있는데, char 타입보다 허용 범위가 작은 byte 타입은 char 타입으로 자동 변환될 수 없다.
왜냐하면 char 타입의 허용 범위는 음수를 포함하지 않는데, byte 타입은 음수를 포함하기 때문이다.

```
byte byteValue = 65;
```

```
char charValue = byteValue;
```



자동 타입 변환 (Promotion) [실습]

```
// 자동 타입 변환
byte byteValue = 10;
int intValue = byteValue;
System.out.println("intValue: " + intValue);

char charValue = '가';
intValue = charValue;
System.out.println("가의 유니코드: " + intValue);

intValue = 50;
long longValue = intValue;
System.out.println("longValue: " + longValue);

longValue = 100;
float floatValue = longValue;
System.out.println("floatValue: " + floatValue);

floatValue = 100.5F;
double doubleValue = floatValue;
System.out.println("doubleValue: " + doubleValue);
```



강제 타입 변환 (Casting)

- 큰 그릇을 작은 그릇 안에 넣을 수 없듯, 큰 허용 범위 타입은 작은 허용 범위 타입으로 자동 타입 변환될 수 없다.
 - 하지만 큰 그릇을 작은 그릇 단위로 쪼개어 한 조각만 작은 그릇에 넣는 것은 가능하다.
-
- 이렇게 큰 허용 범위 타입을 작은 허용 범위 타입으로 쪼개어 저장하는 것을 강제 타입 변환이라고 한다.
 - 강제 타입 변환은 캐스팅 연산자인 괄호를 사용하는데, 괄호 안에 작은 허용 범위 타입을 기재한다.

```
int intValue = 10;  
byte byteValue = (byte) intValue;  
System.out.println(byteValue);
```

00000000

00000000

00000000

00001010

00001010

```
intValue = 103029770;  
byteValue = (byte) intValue;  
System.out.println(byteValue);
```

00000110

00100100

00011100

00001010

00001010



강제 타입 변환 (Casting) [실습]

- 강제 타입 변환의 목적은 원래 값을 유지하면서 타입만 바꾸는 것이다.
- 그렇기 때문에 작은 하용 범위 타입에 저장될 수 있는 값을 가지고 강제 타입 변환을 해야 한다.

```
int var1 = 10;
byte var2 = (byte) var1;
System.out.println(var2);
```

```
long var3 = 300;
int var4 = (int) var3;
System.out.println(var4);
```

```
int var5 = 65;
char var6 = (char) var5;
System.out.println(var6);
```

```
double var7 = 3.14;
int var8 = (int) var7;
System.out.println(var8);
```



연산식에서 자동 타입 변환

- 자바는 실행 성능을 향상시키기 위해 컴파일 단계에서 연산을 수행한다.

```
byte result = 10 + 20;
```

- 자바 컴파일러는 컴파일 단계에서 $10+20$ 을 미리 계산해서 30을 만들고, result 변수에 30을 저장하도록 바이트코드를 생성한다.
따라서 실행 시 덧셈 연산이 없으므로 실행 성능이 좋아진다.

- 하지만 정수가 아니라 변수가 피연산자로 사용되면, 실행 시 연산을 수행해야 한다.
정수 타입 변수가 산술 연산식에서 피연산자로 사용되면
int 타입보다 작은 byte, short 타입 변수는 int 타입으로 자동 타입 변환되어 연산을 수행한다.

```
byte x = 10;  
byte y = 20;  
byte result = x + y;  
int result = x + y;
```

- 즉, 연산의 결과값을 byte 변수에 저장할 수 없고, int 변수에 저장해야 한다.
- 특별한 이유가 없다면 정수 연산에서 변수가 사용될 경우에는
int 타입으로 변수를 선언하는 것이 타입 변환이 발생하지 않기 때문에 실행 성능에도움이 된다.
- 정수 연산식에서 모든 변수가 int 타입으로 변환되는 것은 아니다.
int 타입보다 허용 범위가 더 큰 long 타입이 피연산자로 사용되면 다른 피연산자는 long 타입으로 변환되어 연산을 수행한다.



연산식에서 자동 타입 변환

- 피연산자가 동일한 실수 타입이라면, 해당 타입으로 연산된다.

```
float fResult = 1.2f + 3.4f;
```

- 하지만 피연산자 중 하나라도 double 타입이면 다른 피연산자도 double 타입으로 변환되어 연산되고, 연산 결과 또한 double 타입이 된다.

```
double dResult = 1.2f + 3.4;
```

- int 타입과 double 타입을 연산하는 경우에도 int 타입 피연산자가 double 타입으로 자동 변환되고 연산을 수행한다.
- 만약 int 타입으로 연산을 해야 한다면 double 타입을 int 타입으로 강제 변환하고 덧셈 연산을 수행하면 된다.

```
int intValue = 10;  
double doubleValue = 5.5;  
double result = intValue + doubleValue;
```

```
int intValue = 10;  
double doubleValue = 5.5;  
int result = intValue + (int) doubleValue;
```



연산식에서 자동 타입 변환

- 아래 코드를 실행하면 0.5가 아니라 0.0이 출력된다.

```
int x = 1;  
int y = 2;  
double result = x / y;  
System.out.println(result);
```

- Java에서는 정수 연산의 결과는 항상 정수가 되기 때문에 x/y 의 결과는 0이 되고, 0을 double 타입 변수에 result를 저장하므로 0.0이 된다.
- 위 코드 결과가 0.5가 되기 위해서는 x/y 부분을 정수 연산이 아니라 실수 연산으로 변경해야 한다.
- 즉, x와 y 둘 중 하나 또는 모두를 double 타입으로 변환하는 것이다.



연산식에서 자동 타입 변환 [실습]

```
byte result1 = 10 + 20;  
System.out.println("result1: " + result1);
```

```
byte v1 = 10;  
byte v2 = 20;  
int result2 = v1 + v2;  
System.out.println("result2: " + result2);
```

```
byte v3 = 10;  
int v4 = 100;  
long v5 = 1000L;  
long result3 = v3 + v4 + v5;  
System.out.println("result3: " + result3);
```

```
char v6 = 'A';  
char v7 = 1;  
int result4 = v6 + v7;  
System.out.println("result4: " + result4);  
System.out.println("result4: " + (char) result4);
```

```
int v8 = 10;  
int result5 = v8 / 4;  
System.out.println("result5: " + result5);
```

```
int v9 = 10;  
double result6 = v9 / 4.0;  
System.out.println("result6: " + result6);
```

```
int v10 = 1;  
int v11 = 2;  
double result7 = (double) v10 / v11;  
System.out.println("result7: " + result7);
```



+ 연산자의 두 가지 기능

- Java에서 + 연산자는 피연산자가 모두 숫자일 경우에는 덧셈 연산을 수행하고, 하나가 문자열일 경우에는 나머지 피연산자로 문자열로 자동 변환되어 문자열 결합 연산을 수행한다.

```
int value = 3 + 7;  
String str = "3" + 7;
```

```
System.out.println(value);  
System.out.println(str);
```

- + 연산자는 앞에서부터 순차적으로 수행한다. 먼저 수행된 연산이 덧셈 연산이라면 덧셈 결과를 가지고 + 연산을 수행한다. 만약 먼저 수행된 연산이 문자열 결합 연산이라면 이후 + 연산은 모두 결합 연산이 된다.
- 순차적으로 연산을 수행하지 않고, 특정 부분을 우선 연산하고 싶다면 괄호를 이용하면 된다.
- 괄호는 항상 최우선으로 연산을 수행한다.

```
String result = "1" + (2 + 3);  
System.out.println(result);
```



+ 연산자의 두 가지 기능 [실습]

```
int result1 = 10 + 2 + 8;  
String result2 = 10 + 2 + "8";  
String result3 = 10 + "2" + 8;  
String result4 = "10" + 2 + 8;  
String result5 = "10" + (2 + 8);
```



문자열을 기본 타입으로 변환 [실습]

- 프로그래밍을 하다 보면 문자열을 숫자 타입으로 변환해야 하는 경우가 많다.
 - Java에서 문자열을 기본 타입으로 변환하는 방법은 아래와 같다.
-
- String => byte : **byte** byteValue = Byte.parseByte("10");
 - String => short : **short** shortValue = Short.parseShort("200");
 - String => int : **int** intValue = Integer.parseInt("300000");
 - String => long : **long** longValue = Long.parseLong("40000000000");
 - String => float : **float** floatValue = Float.parseFloat("12.345");
 - String => double : **double** doubleValue = Double.parseDouble("12.345");
 - String => boolean : **boolean** booleanValue = Boolean.parseBoolean("true");



문자열을 기본 타입으로 변환 [실습]

- 반대로 기본 타입의 값을 문자열로 변경해야 하는 경우도 있는데, 이 때는 간단히 String.valueOf() 메소드를 이용하면 된다.

```
int value1 = Integer.parseInt("10");
double value2 = Double.parseDouble("3.14");
boolean value3 = Boolean.parseBoolean("true");
```

```
String str1 = String.valueOf(10);
String str2 = String.valueOf(3.14);
String str3 = String.valueOf(true);
```



변수 사용 범위

- main() 메소드 블록에는 다른 중괄호 블록들이 작성될 수 있다.
- 조건문에 해당하는 if, 반복문에 해당하는 for, while 등이 중괄호 블록을 가질 수 있는데, 중괄호 내에서 선언된 변수는 해당 중괄호 블록 내에서만 사용이 가능하고 밖에서는 사용할 수 없다.

```
public static void main(String[] args) {  
    int v1 = 15;  
    if (v1 > 10) {  
        int v2 = v1 - 10;  
    }  
    int v3 = v1 + v2 + 5; // 변수 v2 사용 불가 [컴파일 에러]  
}
```

- 메소드 블록 전체에서 사용하고자 한다면, 메소드 블록 첫머리에 선언하는 것이 좋고, 특정 블록 내부에서만 사용된다면 해당 블록 내에 선언하는 것이 좋다.



콘솔 입력과 출력

- 데이터를 출력하는 방법 중 가장 간단한 방법은 앞서 활용했던 System.out.print 함수를 이용하는 것이다.
 - 프로그램에 데이터를 입력하는 다양한 방법들 중 쉬운 방법은 Scanner를 사용하는 것이다.
-
- Scanner가 존재하는 위치 표시
`import java.util.Scanner;`
 - Scanner 타입 변수를 선언
`Scanner sc = new Scanner(System.in);`
 - Enter를 입력하기 전까지 읽은 문자열을 변수에 저장
`String inputData = sc.nextLine();`



콘솔 입력과 출력 [실습]

```
Scanner sc = new Scanner(System.in);
System.out.print("x 값 입력 : ");

String strX = sc.nextLine();
int x = Integer.parseInt(strX);

System.out.print("y 값 입력 : ");
String strY = sc.nextLine();
int y = Integer.parseInt(strY);

int result = x + y;
System.out.println("x + y = " + result);
```

```
Scanner sc = new Scanner(System.in);
System.out.print("x 값 입력 : ");
int x = sc.nextInt();

System.out.print("y 값 입력 : ");
int y = sc.nextInt();

int result = x + y;
System.out.println("x + y = " + result);
```



연산자

- 연산자(Operator)란 자료의 가공을 위해 정해진 방식에 따라 계산하고 결과를 얻기 위한 행위를 의미하는 기호

1. 증감 연산자
2. 산술 연산자
3. 비트 이동 연산자(시프트 연산자)
4. 비교 연산자
5. 논리 연산자
6. 삼항 조건 연산자
7. 대입 연산자



부호 연산자 / 증감 연산자

- 부호 연산자는 변수의 부호를 유지(+)하거나 변경(-)할 때 사용된다.

```
byte b = 100;  
byte result = -b;
```

- 위 코드는 컴파일 에러가 발생한다.
왜냐하면 부호 연산도 연산이므로 연산의 결과가 int가 된다. 따라서 int 타입 변수에 대입해야 한다.

```
byte b = 100;  
int result = -b;
```

- 증감 연산자는 변수의 값을 1 증가(++)시키거나 1 감소(--)시키는 연산자이다.
- 변수 단독으로 증감 연산자가 쓰이는 것이 아니라
증감 연산자와 여러 개의 연산자가 사용될 경우에는 증감 연산자 위치에 따라 결과가 달라진다.

```
int x = 1, y = 1;  
int result1 = ++x + 10;  
int result2 = y++ + 10;
```

```
System.out.println("result1: " + result1 + "result2: " + result2);  
System.out.println("x: " + x + "y:" + y);
```



부호 연산자 / 증감 연산자 [실습]

```
int x = 10, y = 10, z;  
x++;  
++x;  
System.out.println("x :" + x);  
  
System.out.println("-----");  
y--;  
--y;  
System.out.println("y :" + y);  
  
System.out.println("-----");  
z = x++;  
System.out.println("x :" + x);  
System.out.println("z :" + z);  
  
System.out.println("-----");  
z = ++x;  
System.out.println("x :" + x);  
System.out.println("z :" + z);  
  
System.out.println("-----");  
z = ++x + y++;  
System.out.println("x :" + x);  
System.out.println("y :" + y);  
System.out.println("z :" + z);
```



산술 연산자

- 산술 연산자는 더하기(+), 빼기(-), 곱하기(*), 나누기(/), 나머지(%)로 구성되어 있다.
- 피연산자가 정수 타입(byte, short, char, int)이면 연산의 결과는 int 타입이다.
- 피연산자가 정수 타입이고, 그 중 하나가 long 타입이면 결과는 long 타입이다.
- 피연산자 중 하나가 실수 타입이면, 연산의 결과는 실수 타입이다.

```
byte v1 = 10;
byte v2 = 4;
long v3 = 10L;

int result1 = v1 + v2;
System.out.println("result1: " + result1);
long result2 = v1 + v2 - v3;
System.out.println("result2: " + result2);
double result3 = (double) v1 / v2;
System.out.println("result3: " + result3);
int result4 = v1 % v2;
System.out.println("result4: " + result4);
```



오버플로우(overflow)와 언더플로우(underflow)

- 오버플로우 : 타입이 허용하는 최대값을 벗어나는 것
- 언더플로우 : 타입이 허용하는 최소값을 벗어나는 것
- 정수 타입 연산에서 허용 범위를 벗어나면, 해당 정수 타입의 최소값 또는 최대값으로 되돌아간다.

```
byte byteValue = 127;
System.out.println(++byteValue);
System.out.println(--byteValue);
short shortValue = 32767;
System.out.println(++shortValue);
System.out.println(--shortValue);
int intValue = 2_147_483_647;
System.out.println(++intValue);
System.out.println(--intValue);
long longValue = 9_223_372_036_854_775_807L;
System.out.println(++longValue);
System.out.println(--longValue);
```



오버플로우(overflow)와 언더플로우(underflow) [실습]

- 만약 연산 과정에서 오버플로우 또는 언더플로우가 발생될 가능성이 있다면, 보다 높은 범위의 타입 연산을 하도록 해야 한다.

```
byte var1 = 125;
for (int i = 0; i < 5; i++) {
    var1++;
    System.out.println("var1: " + var1);
}
```

TIP

for 반복문으로 해당 블록을 5번 실행합니다.

```
System.out.println("-----");
```

```
byte var2 = 125;
for (int i = 0; i < 5; i++) {
    var2--;
    System.out.println("var2: " + var2);
}
```



[참고] 정확한 계산은 정수 연산으로 할 것

- 산술 연산을 정확하게 하기 위해서는 실수 타입을 사용하지 않는 것이 좋다.
- 사과 1개를 10조각으로 보고, 그 중 7조각을 뺀 3조각을 result 변수에 저장

```
Scanner sc = new Scanner(System.in);
```

```
int apple = 1;  
double pieceUnit = 0.1;
```

```
System.out.print("몇 조각을 드셨나요?: ");
```

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

```
double result = apple - number * pieceUnit;
```

```
System.out.println("사과 1개에서 남은 양: " + result);
```

- 출력된 결과를 보면 result 변수의 값은 정확히 0.30이 되지 않는다.



[참고] 정확한 계산은 정수 연산으로 할 것

```
Scanner sc = new Scanner(System.in);

int apple = 1;
int totalPiece = apple * 10;

System.out.print("몇 조각을 드셨나요?: ");
int number = sc.nextInt();

double result = (totalPiece - number) / 10.0;
System.out.println("사과 1개에서 남은 양: " + result);
```



나누기 연산 후 NaN과 Infinity 처리

- 나눗셈 또는 나머지 연산에서 좌측 피연산자가 정수이고 우측 피연산자가 0일 경우, 무한대의 값을 정수로 표현할 수 없기 때문에 예외가 발생한다.

```
int exception = 5 / 0;  
System.out.println(exception);
```

- 하지만 좌측 피연산자가 실수이거나 우측 피연산자가 0.0 또는 0.0f 이면 예외가 발생하지 않고, Infinity(무한대) 또는 NaN(Not a number)이 된다.

```
double infinity = 5 / 0.0;  
double notNumber = 5 % 0.0;  
System.out.println(infinity);  
System.out.println(notNumber);
```



나누기 연산 후 NaN과 Infinity 처리

- Infinity 또는 NaN 상태에서 계속해서 연산을 수행하면, 어떤 연산을 하더라도 결과는 Infinity 또는 NaN이 되므로 데이터가 엉망이 될 수 있다.
- 그렇기 때문에 Infinity 또는 NaN인지 먼저 확인하고, 다음 연산을 수행하는 것이 좋다. 이를 확인하기 위해서는 Double.isInfinite() DoubleisNaN()을 사용한다.

```
int number = 5;

Scanner sc = new Scanner(System.in);
System.out.print("나눌 값을 입력하세요 : ");
double value = sc.nextDouble();

boolean resultA = Double.isInfinite(number / value);
boolean resultB = DoubleisNaN(number % value);

System.out.println(resultA);
System.out.println(resultB);
```



비교 연산자

- 비교 연산자는 동등 또는 크기를 평가해서 boolean 타입인 true/false를 산출한다.
 - boolean을 산출하는 연산의 경우에는 코드의 흐름을 제어하는 제어문(조건문, 반복문)에서 주로 사용된다.
-
- 동등 비교 연산자: ==, !=
 - 크기 비교 연산자: >, >=, <, <=
-
- 피연산자 간 데이터 타입이 다를 경우에는 비교 연산을 수행하기 전에 자동으로 타입을 일치시켜 비교한다.
- ```
System.out.println('A' == 65);
System.out.println(3 == 3.0);
```
- 하지만 한 가지 예외가 있는데, 부동 소수점 방식을 사용하는 실수 타입은 정밀도 차이로 인해 정확한 비교가 되지 않는다.

```
System.out.println(0.1 == 0.1f);
System.out.println((float) 0.1 == 0.1f);
```



# 비교 연산자

- 문자열을 비교할 때는 동등 비교 연산자 대신 equals(), !equals()를 사용한다.

```
String str1 = "Hello";

Scanner sc = new Scanner(System.in);
System.out.print("비교대상 문자열 입력 : ");
String str2 = sc.nextLine();

System.out.println(str1 == str2);
System.out.println(str1.equals(str2));
System.out.println(!str1.equals(str2));
```



# 논리 연산자

- 논리 연산자는 논리곱(&, &&), 논리합(|, ||), 배타적 논리합(^), 논리 부정(!) 연산을 수행한다.
- 논리 연산자도 boolean을 산출하기 때문에 제어문(조건문, 반복문)에서 주로 사용된다.

| 구분               | 연산식   |         | 결과    | 설명    |
|------------------|-------|---------|-------|-------|
| AND<br>(논리곱)     | true  | &       | true  | TRUE  |
|                  | true  | 또는<br>& | false | FALSE |
|                  | false | &       | true  | FALSE |
|                  | false |         | false | FALSE |
| OR<br>(논리합)      | true  |         | true  | TRUE  |
|                  | true  | 또는<br>  | false | TRUE  |
|                  | false |         | true  | TRUE  |
|                  | false |         | false | FALSE |
| XOR<br>(배타적 논리합) | true  |         | true  | FALSE |
|                  | true  | ^       | false | TRUE  |
|                  | false |         | true  | TRUE  |
|                  | false |         | false | FALSE |
| NOT<br>(논리 부정)   |       | !       | true  | FALSE |
|                  |       |         | false | TRUE  |

## TIP

-&보다 &&이 효과적  
-|보다 ||이 효과적



# 논리 연산자 [실습]

```
Scanner sc = new Scanner(System.in);
String str = sc.nextLine();
char c = str.charAt(0);

if((65 <= c) && (c <= 90)) {
 System.out.println("대문자이군요.");
}

if((97 <= c) && (c <= 122)) {
 System.out.println("소문자이군요.");
}

if((48 <= c) && (c <= 57)) {
 System.out.println("숫자이군요.");
}
```

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();

if((i%2 == 0) || (i%3 == 0)) {
 System.out.println("2 또는 3의 배수이군요.");
}
```



# 비트 논리 연산자

- 비트 논리 연산자는 bit 단위(0과 1)로 논리 연산을 수행한다.
- 0과 1이 피연산자가 되므로 2진수로 저장되는 정수 타입만 피연산자가 될 수 있고, 부동 소수점 방식으로 저장되는 실수 타입은 피연산자가 될 수 없다.

| 구분               | 연산식 |   | 결과     | 설명     |
|------------------|-----|---|--------|--------|
| AND<br>(논리곱)     | 1   | & | 1      | 1      |
|                  | 1   |   | 0      | 0      |
|                  | 0   |   | 1      | 0      |
|                  | 0   |   | 0      | 0      |
| OR<br>(논리합)      | 1   |   | 1      | 1      |
|                  | 1   |   | 0      | 1      |
|                  | 0   |   | 1      | 1      |
|                  | 0   |   | 0      | 0      |
| XOR<br>(비타적 논리합) | 1   | ^ | 1      | 0      |
|                  | 1   |   | 0      | 1      |
|                  | 0   |   | 1      | 1      |
|                  | 0   |   | 0      | 0      |
| NOT<br>(논리 부정)   |     | ~ | 1<br>0 | 0<br>1 |



# 비트 이동 연산자

- 비트 연산자에는 논리 연산자 외에도 이동 연산자가 있다.
- 비트 이동 연산자는 비트를 좌측 또는 우측으로 밀어서 이동시키는 연산을 한다.

| 구분            | 연산식 |     |   | 설명                                                                                |
|---------------|-----|-----|---|-----------------------------------------------------------------------------------|
| 이동<br>(shift) | a   | <<  | b | 정수 a의 각 비트를 b만큼 왼쪽으로 이동<br>오른쪽 빈자리는 0으로 채움<br>$a \times 2^b$ 과 동일한 결과가 됨          |
|               | a   | >>  | b | 정수 a의 각 비트를 b만큼 오른쪽으로 이동<br>왼쪽 빈자리는 최상위 부호 비트와 같은 값으로 채움<br>$a / 2^b$ 과 동일한 결과가 됨 |
|               | b   | >>> | b | 정수 a의 각 비트를 b만큼 오른쪽으로 이동<br>왼쪽 빈자리는 0으로 채움                                        |



# 대입 연산자

- 대입 연산자는 우측 피연산자의 값을 좌측 피연산자인 변수에 대입한다.
- 단순히 값을 대입하는 단순 대입 연산자와 정해진 연산을 수행한 후 결과를 대입하는 복합 대입 연산자로 나뉜다.

| 구분    | 연산식 |      |      |
|-------|-----|------|------|
| 단순 대입 | 변수  | =    | 피연산자 |
| 복합 대입 | 변수  | +=   |      |
|       |     | -=   |      |
|       |     | *=   |      |
|       |     | /=   |      |
|       |     | %=   |      |
|       |     | &=   | 피연산자 |
|       |     | =    |      |
|       |     | ^=   |      |
|       |     | <<=  |      |
|       |     | >>=  |      |
|       |     | >>>= |      |

```
int result = 0;

result += 10;
System.out.println(result);

result -= 5;
System.out.println("result: " + result);

result *= 3;
System.out.println("result: " + result);

result /= 5;
System.out.println("result: " + result);

result %= 3;
System.out.println("result: " + result);
```



# 삼항(조건) 연산자

- 삼항연산자는 물음표와 콜론으로 구성되어, 총 3개의 피연산자를 갖는다.
- [조건식? 값 또는 연산식 : 값 또는 연산식]
  - 첫 번째 피연산자는 boolean 변수 또는 조건식이 온다.
  - 두 번째 피연산자는 첫 번째 피연산자가 true일 경우에 선택된다.
  - 두 번째 피연산자는 첫 번째 피연산자가 false일 경우에 선택된다.

```
int score = 85;
char grade = (score > 90) ? 'A' : 'B';
```

```
System.out.println(score + "점은 " + grade + "등급입니다.");
```



## 삼항(조건) 연산자 [실습]

```
Scanner sc = new Scanner(System.in);
int score = sc.nextInt();

char grade = (score > 90) ? 'A'
 : (score > 80) ? 'B'
 : (score > 70) ? 'C'
 : (score > 60) ? 'D'
 : 'F';

System.out.println(score + "점은 " + grade + "등급입니다.");
```



# 연산의 방향과 우선순위

- 일반 산술 연산식에서 덧셈과 뺄셈 연산보다 곱셈과 나눗셈 연산이 우선 처리되듯, 모든 연산자에는 우선 순위가 정해져 있다.
- 우선 순위를 숙지하여도 여러 가지 연산자들이 섞여 있다면 혼란스러울 수 있다.
- 이때는 먼저 처리해야 할 연산을 괄호로 묶어서 수행 순서를 명확히 하는 것이 좋다.

- 증감(++, --) 부호(+, -) 비트(~), 논리(!)
- 산술(\*, /, %)
- 산술(+, -)
- 비트 이동(<<, >>, >>>)
- 비교(<, >, <=, >=, instanceof)
- 비교(==, !=)
- 논리(&)
- 논리(^)
- 논리(|)
- 논리(&&)
- 논리(||)
- 삼항 조건(? :)
- 대입(=, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=)



# 연산자 [실습]

- 선언된 변수를 통한 연산으로 주석에 나타난 결과와 동일하게 출력되도록 빈 칸을 완성하시오.
  - 힌트] ASCII 값을 활용하자. 'A'→65 / 'B'→66 / '1'→49 / '2'→50

```
public static void main(String[] args) {
 String s1 = "1";
 String s2 = "2";
 boolean bnx = false;
 char c1 = 'A';
 char c2 = 'B';
 char c3 = '1';
 char c4 = '2';
 int inx = 2;

 System.out.println_____(); // 12
 System.out.println_____(); // true
 System.out.println_____(); // 131
 System.out.println_____(); // 51
 System.out.println_____(); // 99
}
```



## 연산자 [실습]

- 아래와 같이 변수가 선언되어 있을 때, 100의 자리만 남기고 나머지 자리수는 0으로 바꾸는 프로그램을 완성하시오.

```
public static void main(String[] args) {
 int num = 456;
 int result = _____;
 System.out.println(result); // 400
}
```



# 연산자 [실습]

- 문자형 변수 ch가 영문자(대문자 또는 소문자)일 때만, 변수 b의 값이 true가 되도록 프로그램을 완성하시오.

```
public static void main(String[] args) {
 char ch = 'z';
 boolean b = _____;
 System.out.println(b); // true
}
```



# 연산자 [실습]

- 화씨를 섭씨로 변경하는 프로그램을 작성하시오.
  - C(Celsius) : 섭씨, F(Fahrenheit) : 화씨
  - 공식 :  $C = \frac{5}{9} * (F - 32)$

```
public static void main(String[] args) {
 int fahrenheit = 100;
 float celcius = _____;
 System.out.println("Fahrenheit: " + fahrenheit); // Fahrenheit: 100
 System.out.println("Celcius: " + celcius); // Celcius: 37.77778
}
```



# 연산자 [실습]

- 변수 선언이 아래와 같을 때, 실행 결과가 주석과 같이 출력될 수 있도록 프로그램을 완성하시오.

```
public static void main(String[] args) {
 byte a = 10;
 byte b = 20;
 byte c = _____;
 char ch = 'A';
 ch = _____;
 float f = _____;
 float f2 = 0.1f;
 double d = 0.1;
 boolean result = ____ d == f2;
 System.out.println("c="+c); // 30
 System.out.println("ch="+ch); // C
 System.out.println("f="+f); // 1.5
 System.out.println("result="+result); // true
}
```



# 연산자 [실습]

- Scanner 클래스를 이용하여 키보드로 두 개의 정수값을 입력받아 최대값을 구하는 코드를 작성하시오.
- 단, 삼항조건연산자를 사용할 것

```
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 System.out.println("1. 정수를 입력하세요.");
 _____;
 System.out.println("2. 정수를 입력하세요.");
 _____;

 System.out.println(______); // 최대값 : ____
}
```



# 연산자 [실습]

- Scanner 클래스를 이용하여 키보드로 세 개의 정수값을 입력받아 최대값을 구하는 코드를 작성하시오.
- 단, 삼항조건연산자를 사용할 것

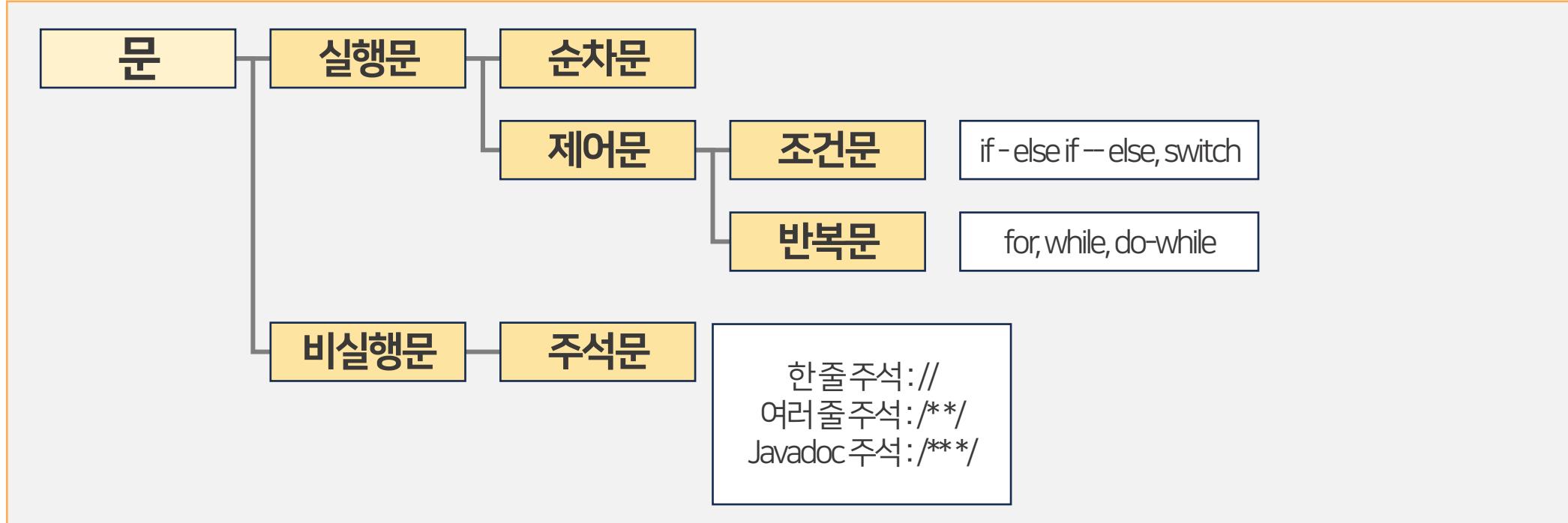
```
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 System.out.println("1. 정수를 입력하세요.");
 _____;
 System.out.println("2. 정수를 입력하세요.");
 _____;
 System.out.println("3. 정수를 입력하세요.");
 _____;

 System.out.println(______); // 최대값 : ____
}
```



# Statement

- 문(statement)은 Java 프로그래밍에서 소스 코드에 입력시키는 문장을 의미한다.



## TIP

Javadoc 주석은 HTML 문서화로 주석이 처리되므로 API와 같은 도움말 페이지를 만들 때 사용된다.



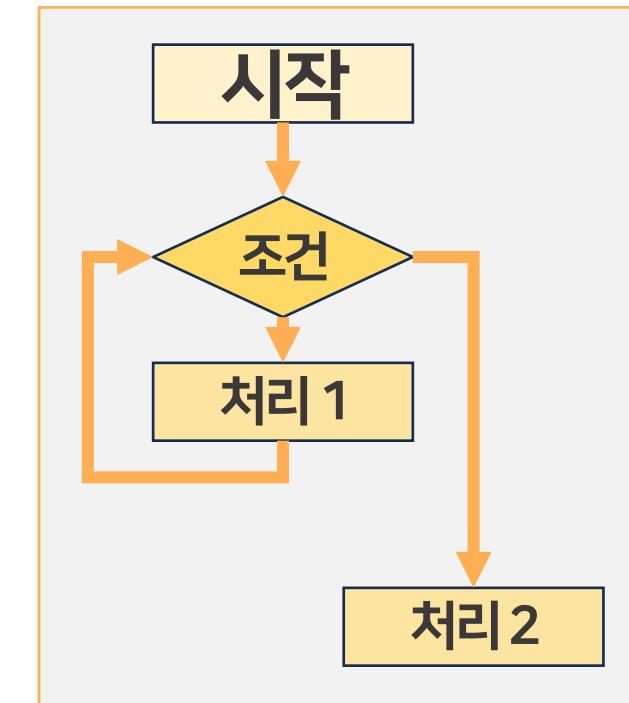
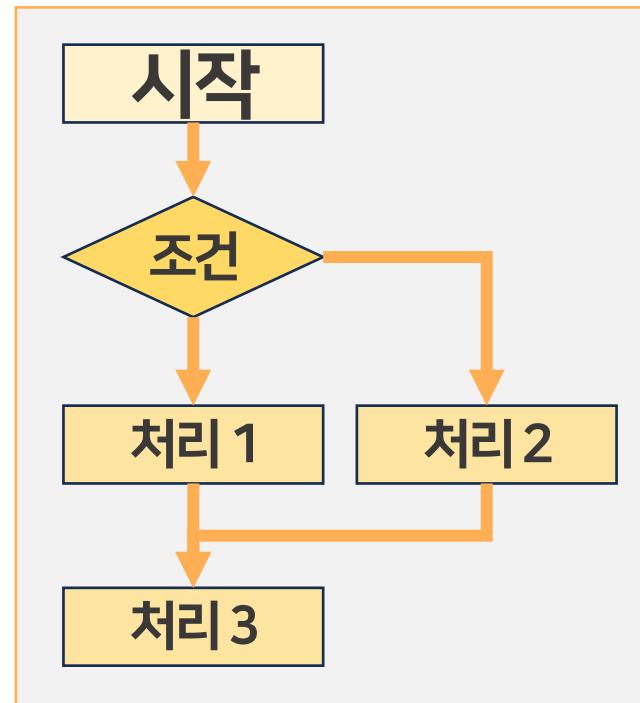
# Statement

- 순차문 : 메서드 내의 문장 중에서 순차적으로 실행되는 문장으로 반드시 세미콜론(;)으로 끝난다.
- 제어문 : 프로그램의 흐름에 영향을 주고, 때에 따라 제어가 가능하다. 모든 제어문은 중첩이 가능하다.
- 주석문 : 실제 프로그램에 영향을 주지 않으며 단지 소스코드의 기능이나 동작을 설명하기 위해 사용되는 문장이다.



# 제어문

- 자바 프로그램은 main() 메소드의 시작부터 끝까지 위에서부터 아래로 실행하는 흐름을 가지고 있다.
- 이러한 실행 흐름을 개발자가 원하는 방향으로 바꿀 수 있도록 해주는 것이 [제어문]이다.





# 제어문

- 제어문은 조건식과 중괄호 블록으로 구성되는데, 조건식의 연산 결과에 따라 블록 내부의 실행 여부가 결정된다.
- 제어문의 종류
  - 조건문(if 문, switch 문)
  - 반복문(for 문, while 문, do-while 문)
  - break문 : 반복문을 빠져나갈 때 또는 switch문을 빠져나갈 때 주로 쓰인다.
  - continue문 : 반복문에서 현재 진행되는 회차를 포기하고 다음 회차로 이동한다.
- 조건문은 제어문 블록을 마친 후, 정상 흐름으로 돌아온다.
- 반복문은 제어문 블록을 마친 후, 다시 제어문으로 되돌아가 반복 실행 (looping) 된다.
- 제어문 블록 내부에 또 다른 제어문 사용이 가능하다.



# if 문

- 조건식의 결과에 따라 블록 실행 여부가 결정된다.
- 조건식에는 true, false 값을 산출할 수 있는 연산식이나 boolean 변수가 올 수 있다.
- 조건식이 true이면 블록을 실행하고, false이면 블록을 실행하지 않는다.
- 중괄호 블록 내 실행문이 한 문장이라면 중괄호 생략이 가능하다. 다만, 가독성을 떨어뜨려 버그 발생의 원인이 될 수 있다.

```
Scanner sc = new Scanner(System.in);
System.out.print("점수 입력 : ");
int score = sc.nextInt();

if (score >= 60) {
 System.out.println("축하합니다.");
 System.out.println("합격입니다.");
}

if (score < 60) System.out.println("불합격입니다.");
```



# if - else문

- if 문은 else 블록과 함께 사용되어 조건식의 결과에 따라 실행 블록을 선택할 수 있다.
- if 문의 조건식이 true이면 if 문 블록이 실행되고, false이면 else 블록이 실행된다.

```
Scanner sc = new Scanner(System.in);
System.out.print("점수 입력 : ");
int score = sc.nextInt();

if (score >= 60) {
 System.out.println("축하합니다.");
 System.out.println("합격입니다.");
} else {
 System.out.println("불합격입니다.");
}
```



# if - else if 문

- 조건문이 여러 개인 if 문도 있다.
- else if 문은 상위 조건식이 false일 경우 평가되고, else if 가 true면 해당 블록이 실행된다.
- else if 의 수는 제한이 없으며, 여러 개의 조건식 중 true 가 되는 블록만 실행하고 전체 if 문을 벗어나게 된다.
- 마지막에는 else 블록을 추가하여, 모든 조건식이 false 일 경우 실행되도록 할 수 있다.

```
Scanner sc = new Scanner(System.in);
System.out.print("점수 입력 : ");
int score = sc.nextInt();

if (score >= 90) System.out.println("90점 이상으로 A등급 합격입니다.");
else if (score >= 80) System.out.println("80점 이상으로 B등급 합격입니다.");
else if (score >= 70) System.out.println("70점 이상으로 C등급 합격입니다.");
else if (score >= 60) System.out.println("60점 이상으로 D등급 합격입니다.");
else System.out.println("F등급, 불합격입니다.");
```



# if 문 [실습]

```
int score = (int) (Math.random() * 30) + 71; // 70 ~ 100까지의 숫자 중 랜덤숫자 반환
System.out.println("점수 : " + score);
```

```
String grade = "F";
```

// 중첩 if를 이용하여 문제를 풀 것.

// 바깥 if : 90점 이상은 A, 80점 이상은 B, 70점 이상은 C 등급을 부여한다.

// 중첩 if : 95점, 85점, 75점을 기준으로 등급에 +를 붙인다.

```
System.out.println("학점 : " + grade);
```



# switch 문

- if 문은 조건식의 결과가 true, false 두 가지 밖에 없기 때문에 경우의 수가 많아질수록 else if를 반복적으로 추가해야 하므로 코드가 복잡해진다.
- 그러나 switch 문은 변수의 값에 따라 실행문이 결정되기 때문에 같은 기능을 하는 if 문보다 코드가 간결해진다.
- switch 문은 괄호 안의 변수값에 따라 해당 case로 가서 실행문을 실행시킨다.
- case 끝마다 있는 break는 다음 case를 실행하지 않고, switch 문을 빠져나가기 위해 필요하다. 만약 break가 없다면 다음 case가 연달아 실행된다.

```
int genderNum = 3;

switch (genderNum % 2) {
 case 0:
 System.out.println("남성입니다.");
 break;
 case 1:
 System.out.println("여성입니다.");
 break;
}
```



# switch 문

- switch 문의 괄호에는 정수 타입(byte, char, short, int)과 문자열 타입(String) 변수, enum 타입 변수를 사용할 수 있다.
- 변수 값과 동일한 값을 갖는 case가 없으면, default로 가서 실행문을 실행시킨다. default는 생략 가능하다.

```
char grade = (char) ((Math.random() * 6) + 'A');
System.out.println("등급: " + grade);
switch(grade) {
 case 'A':
 System.out.println("훌륭합니다!");
 break;
 case 'B':
 System.out.println("좋습니다!");
 break;
 case 'C':
 System.out.println("나쁘지 않습니다!");
 break;
 case 'D':
 System.out.println("조금 아쉽습니다!");
 break;
 case 'E':
 System.out.println("나쁩니다!");
 break;
 case 'F':
 System.out.println("실패했습니다!");
 break;
 default:
 System.out.println("잘못된 등급입니다!");
}
```



# switch 문 [실습]

```
Scanner sc = new Scanner(System.in);
System.out.print("등급을 입력하세요: ");
String grade = sc.nextLine();

switch (grade) {
 case "A":
 case "a":
 System.out.println("훌륭합니다!");
 break;
 case "B":
 case "b":
 System.out.println("좋습니다!");
 break;
 case "C":
 case "c":
 System.out.println("나쁘지 않습니다!");
 break;
 case "D":
 case "d":
 System.out.println("조금 아쉽습니다!");
 break;
 case "F":
 case "f":
 System.out.println("실패했습니다!");
 break;
 default:
 System.out.println("잘못된 등급입니다!");
}
```



# 개선된 switch 문 (switch expressions)

- Java 12 이후부터는 switch 문에서 표현식(Expression)을 사용할 수 있다.
- break 문을 없애는 대신에 화살표(->)와 중괄호를 사용해 가독성이 개선되었다.
- 여기에서도 실행문이 하나만 있을 경우에는 중괄호를 생략할 수 있다.

```
Scanner sc = new Scanner(System.in);
System.out.print("등급을 입력하세요: ");
String grade = sc.nextLine();

switch (grade) {
 case "A", "a" -> System.out.println("훌륭합니다!");
 case "B", "b" -> System.out.println("좋습니다!");
 case "C", "c" -> System.out.println("나쁘지 않습니다!");
 case "D", "d" -> System.out.println("조금 아쉽습니다!");
 case "F", "f" -> System.out.println("실패했습니다!");
 default -> System.out.println("잘못된 등급입니다!");
}
```



# 개선된 switch 문 (switch expressions)

- 개선된 switch 문을 사용하면 스위치된 값을 변수에 바로 대입할 수도 있다.
- 단일 값일 경우에는 화살표 오른쪽에 바로 값을 기술하면 되고, 중괄호를 사용할 경우에는 yield 키워드를 통해 값을 지정하면 된다.
- yield 키워드를 사용할 경우에는 default가 반드시 존재해야 한다.

```
Scanner sc = new Scanner(System.in);
System.out.print("등급을 입력하세요: ");
String grade = sc.nextLine();
```

```
int score = switch (grade) {
 case "A", "a" -> 100;
 case "B", "b" -> 90;
 case "C", "c" -> 80;
 case "D", "d" -> 70;
 case "F", "f" -> {
 int result = 100;
 result -= 40;
 yield result;
 }
 default -> 60;
};
```

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



# for 문

- 프로그램을 작성하다 보면 똑같은 실행문을 반복적으로 실행해야 하는 경우가 많이 발생한다.

```
System.out.println("안녕하세요");
```

- 반복되는 코드를 계속해서 작성해야 한다면, 코드의 양이 엄청 늘어날 것이다.  
이런 경우, for 문을 사용하면 코드를 획기적으로 줄일 수 있다.

```
for (int i = 0; i < 10; i++) {
 System.out.println("안녕하세요");
}
```



# for 문

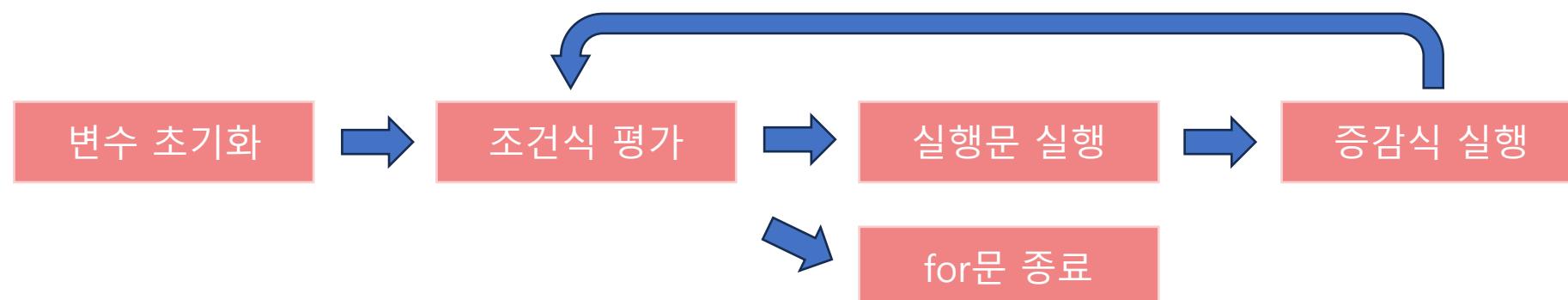
- for 문의 괄호 내부에서 선언된 변수를 이용하면 반복문의 활용 범위는 더욱 커진다.

```
int sum = 0;
sum += 1;
sum += 2;
sum += 3;
sum += 4;
sum += 5;
System.out.println("1~5까지의 합: " +sum);
```

```
int sum = 0;
for (int i = 1; i <= 5; i++) {
 sum += i;
}
System.out.println("1~5까지의 합: " +sum);
```

## TIP

```
for (변수초기화; 조건식; 증감식) {
 실행문
}
```





# for 문

- for 문 괄호 내부에서 선언된 변수는 for 문 안에서만 사용되는 지역(local) 변수이다. 그리고 변수는 하나일 수도 있고, 둘 이상이 될 수도 있다. 이런 경우에는 쉼표로 구분해서 작성한다.

```
for (int num = 1, triangularNum = 1; num <= 5; triangularNum += ++num) {
 System.out.println("숫자: " + num + ", 삼각수 : " + triangularNum);
}
```

- 부동 소수점 타입은 연산 과정에서 정확한 숫자를 표현하지 못하기 때문에 초기화식에서는 부동 소수점을 쓰는 float 타입을 사용하지 않는 것이 좋다.

```
for(float x=0.1f; x<=1.0f; x+=0.1f) {
 System.out.println(x);
}
```



# 중첩 for 문

- for 문은 또 다른 for 문을 내포할 수 있는데, 이것을 중첩된 for 문이라 한다.
- 이 경우 바깥 for 문이 한번 실행될 때마다, 중첩된 for 문은 지정 횟수만큼 반복하고 다시 바깥 for 문으로 돌아간다.

```
int rows = 5;

for (int i = 1; i <= rows; i++) {
 for (int j = 1; j <= i; j++) {
 System.out.print("* ");
 }
 System.out.println();
}
```



# 중첩 for 문 [실습]

- 우측 내용처럼 중첩된 for 문을 사용하여 구구단을 출력해보자.
- 바깥쪽 for 루프는 구구단의 각 단을 나타내고, 안쪽 for 루프는 각 단의 곱셈 결과를 출력하면 된다.
- 바깥쪽 루프가 한번 반복할 때마다 안쪽 루프가 1부터 9까지 반복하여 해당 단의 구구단을 출력하게 된다.

2단

$$2 \times 1 = 2$$

$$2 \times 2 = 4$$

$$2 \times 3 = 6$$

$$2 \times 4 = 8$$

$$2 \times 5 = 10$$

$$2 \times 6 = 12$$

$$2 \times 7 = 14$$

$$2 \times 8 = 16$$

$$2 \times 9 = 18$$

3단

$$3 \times 1 = 3$$

$$3 \times 2 = 6$$

$$3 \times 3 = 9$$

(...중략...)

9단

$$9 \times 1 = 9$$

$$9 \times 2 = 18$$

$$9 \times 3 = 27$$

$$9 \times 4 = 36$$

$$9 \times 5 = 45$$

$$9 \times 6 = 54$$

$$9 \times 7 = 63$$

$$9 \times 8 = 72$$

$$9 \times 9 = 81$$

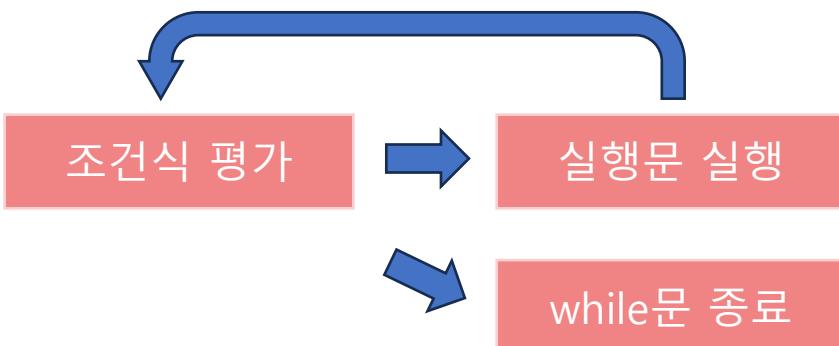


# while 문

- for 문을 정해진 횟수만큼 반복한다면, while 문은 조건식이 true일 경우에 계속해서 반복하고 false가 되면 반복을 멈춘다.

TIP

```
while (조건식) {
 실행문
}
```



```
int i = 1;
while (i <= 10) {
 System.out.println("안녕하세요.");
 i++;
}
```



# while 문 [실습]

- while 문은 조건식을 평가한 후, 실행문이 반복 실행되기 때문에 실행문 내부에서 조건식을 false로 변경될 수 있도록 해야한다.
- 1부터 100까지의 합계를 구하는 while 문을 작성해보자.

```
int i = 1, sum = 0;

while (i <= 100) {
 sum += i++;
}

System.out.println("1부터 " + (i - 1) + "의 합계는 : " + sum);
```



# while 문

- while 문의 조건식에 true를 사용하면, 실행문이 무한 반복하게 된다.
- 이 경우에는 언젠가는 while 문을 빠져나가기 위한 코드가 필요하다.

```
Scanner sc = new Scanner(System.in);
boolean run = true;
int speed = 0;

while (run) {
 System.out.println("---".repeat(8));
 System.out.println("1. 증속 | 2. 감속 | 3. 중지");
 System.out.println("---".repeat(8));
 System.out.print("선택 : ");
 int num = sc.nextInt();

 if (num == 1) {
 speed++;
 System.out.println("현재 속도 = " + speed);
 } else if (num == 2) {
 speed--;
 System.out.println("현재 속도 = " + speed);
 } else if (num == 3) {
 run = false;
 }
}

System.out.println("프로그램 종료");
```



# while 문

- 무한히 반복하는 while 문을 빠져나가는 또 다른 방법은 break를 이용하는 것이다.

```
Scanner sc = new Scanner(System.in);
int speed = 0;

while (true) {
 System.out.println("---".repeat(8));
 System.out.println("1. 증속 | 2. 감속 | 3. 중지");
 System.out.println("---".repeat(8));
 System.out.print("선택 : ");
 int num = sc.nextInt();

 if (num == 1) {
 speed++;
 System.out.println("현재 속도 = " + speed);
 } else if (num == 2) {
 speed--;
 System.out.println("현재 속도 = " + speed);
 } else if (num == 3) {
 break;
 }
}

System.out.println("프로그램 종료");
```

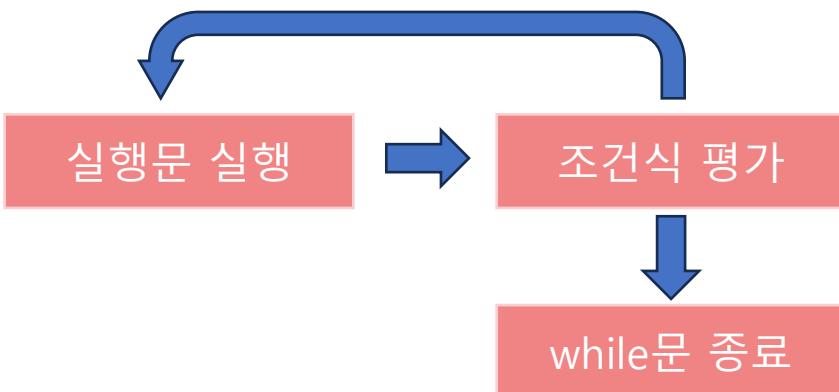


# do - while 문

- do-while 문은 조건식에 의해 반복 실행된다는 점에서 while 문과 동일하다.
- 하지만 시작할 때부터 조건식을 평가하는 while 문과 달리, do-while 문은 실행문을 먼저 실행시키고 조건식을 평가하여 반복 실행 여부를 결정한다.

TIP

```
do {
 실행문
} while (조건식);
```





# do - while 문 [실습]

```
System.out.println("메시지를 입력하세요.");
System.out.println("프로그램을 종료하려면 q를 입력하세요.");
Scanner sc = new Scanner(System.in);
String inputStr;

do {
 System.out.print("입력: ");
 inputStr = sc.nextLine();
 System.out.println(inputStr + " 을 입력하셨습니다.");
} while(!inputStr.equals("q"));

System.out.println("프로그램 종료");
```



# break 문

- break는 for 문, while 문, do-while 문의 실행을 중지하거나 조건문의 switch 문을 종료할 때 사용한다.
- break는 대부분 if 문과 같이 사용되어 조건식에 따라 반복문을 종료한다.

```
System.out.println("주사위가 6이 나오면 프로그램을 종료합니다.");

while (true) {
 System.out.println("주사위를 굴립니다.");
 int num = (int) (Math.random() * 6) + 1;
 System.out.println(num);
 if (num == 6) {
 System.out.println("6이 나왔습니다.");
 break;
 }
}

System.out.println("프로그램 종료");
```



# break 문

- 반복문이 중첩되어 있는 경우 break는 가장 가까운 반복문만을 종료하고, 바깥쪽 반복문까지 종료시키지 않는다.
- 만약 중첩된 반복문에서 바깥쪽까지 종료시키기 위해서는  
바깥쪽 반복문에 이름(Label)을 붙이고, [break 이름:]을 사용하면 된다.

```
for (char upper='A'; upper <= 'Z'; upper++) {
 for (char lower='a'; lower <= 'z'; lower++) {
 System.out.println(upper + "-" + lower);
 if(lower == 'c') {
 break;
 }
 }
}
```

```
Outer: for (char upper='A'; upper <= 'Z'; upper++) {
 for (char lower='a'; lower <= 'z'; lower++) {
 System.out.println(upper + "-" + lower);
 if(lower == 'c') {
 break Outer;
 }
 }
}
```



# continue 문

- continue 문은 반복문인 for 문, while 문, do-while 문에서만 사용된다.
- continue 문이 실행되면 for 문의 증감식 또는 while, do-while 문의 조건식으로 바로 이동하게 된다.
- break 문은 반복문을 종료하지만, continue 문은 해당 반복을 건너뛰고 다음 반복을 계속해서 수행한다는 점에서 다르다.

```
for(int i=1; i<=10; i++) {
 if(i%2 != 0) {
 continue;
 }
 System.out.print(i + " ");
}
```



## 제어문 [실습]

- 1부터 20까지의 정수 중에서 2의 배수가 아니고 3의 배수가 아닌 수의 총합을 구하시오.



# 제어문 [실습]

- 아래의 for문을 while문으로 변경하시오.

```
public static void main(String[] args) {
 for(int i = 0; i <= 10; i++) {
 for(int j = 0; j <= i; j++) {
 System.out.print("*");
 }
 System.out.println();
 }
}
```



# 제어문 [실습]

- 두 개의 주사위를 던졌을 때, 눈의 합이 6이 되는 모든 경우의 수를 출력하는 프로그램을 작성하시오.

```
/*
 * 출력 결과
 * 1+5=6
 * 2+4=6
 * 3+3=6
 * 4+2=6
 * 5+1=6
 */
```



# 제어문 [실습]

- 세 개의 주사위를 던졌을 때, 눈의 곱이 3의 배수인 값을 출력하는 프로그램을 작성하시오.

```
/*
 * 출력 결과
 * 1*1*3 = 3
 * 1*1*6 = 6
 * 1*2*3 = 6
 * 1*2*6 = 12
 *
 * ...
 */

```



# 제어문 [실습]

- 1부터 100까지 더하는 프로그램을 작성하시오.  
단, Scanner로 1자리 정수형 데이터를 입력 받아 입력 받은 수의 배수만 더해야 한다.
- 입력 예: 5 -> [ 5, 10, 15, 20, …, 100의 합 출력]



# 제어문 [실습]

- 1부터 100까지 반복하면서 3의 배수는 three, 5의 배수는 five, 7의 배수는 seven 값을 출력하는 프로그램을 작성하시오.

```
/*
 * 출력 결과
 * 1
 * 2
 * 3 three
 * 4
 * 5 five
 * 6 three
 * ...
 * 14 seven
 * 15 three five
 * 16
 * ...
 * 35 five seven
 * ...
 * 99 three
 * 100 five
 */
```



## 제어문 [실습]

- Scanner 클래스를 사용하여 키보드로 정수값을 입력 받은 후, 1부터 입력 받은 정수값까지의 총합을 구하시오.
- 단, 음수 및 0을 입력한 경우에는 다시 입력을 받을 수 있도록 한다.