웹 개발자를 위한 자바 기초

CONTENTS

목차

Chapter 01

강의 소개

웹 개발자를 위한 자바 기초

강의 소개

- ✓ 우리가 사용하는 소프트웨어는 매우 다양함
  - ✓ 모바일 애플리케이션
    - ✓ ex) 카카오톡, 인스타그램
  - ✓ 웹 애플리케이션
    - ✓ ex) 페이스북, 알리바바
  - ✓ 데스크탑 프로그램
    - ✓ ex) 슬랙, 디스코드
- ✓ 각각 만드는 방식, 언어, 기술이 다르다.

강의 소개

✓ 우리는 **웹 애플리케이션**을 개발하는 방법을 배움

- ✓ 웹 애플리케이션 = 인터넷으로 접속하는 프로그램
  - ✓ ex) 블로그, 온라인 쇼핑몰, 커뮤니티 사이트
  - ✓ PC 에서도 되고, 모바일에서도 되는 서비스

✓ 웹 개발자

- ✓ 사용자와 직접 소통할 화면을 구현
- ✓ 안보이는 곳에서 돌아가는 서비스 로직 구현
- ✓ 서비스 로직과 화면을 연결

강의 소개

✓ Web Application 에서 필요한 요소는 매우 많다

- ✓ 로그인 → 보안
- ✓ 주문 처리 → 비즈니즈 로직
- ✓ 화면 표시 → UI
- ✓ 데이터 저장 → DB (데이터베이스)
- ✓ 파일 업로드, 결제, 알림 등등...

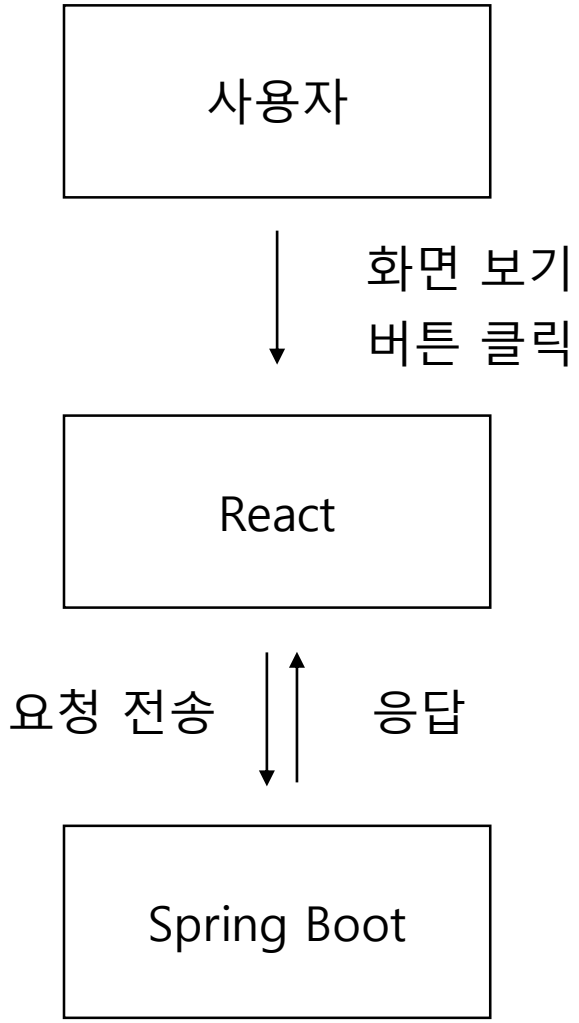
✓ 직접 다 만들기는 너무 어려움

- ✓ 그래서 남들이 만들어 놓은 코드를 가져와 쓴다.
  - ✓ **프레임워크, 라이브러리**

강의 소개

✓웹에서 가장 많이 사용되는 프레임워크/라이브러리

- ✓ **React (프론트엔드)**
  - ✓ 화면 개발
  - ✓ JavaScript 학습 필요
- ✓ **Spring Boot (백엔드)**
  - ✓ 화면에 보이지 않는 기능, 로직, 서버
  - ✓ Java 학습 필요



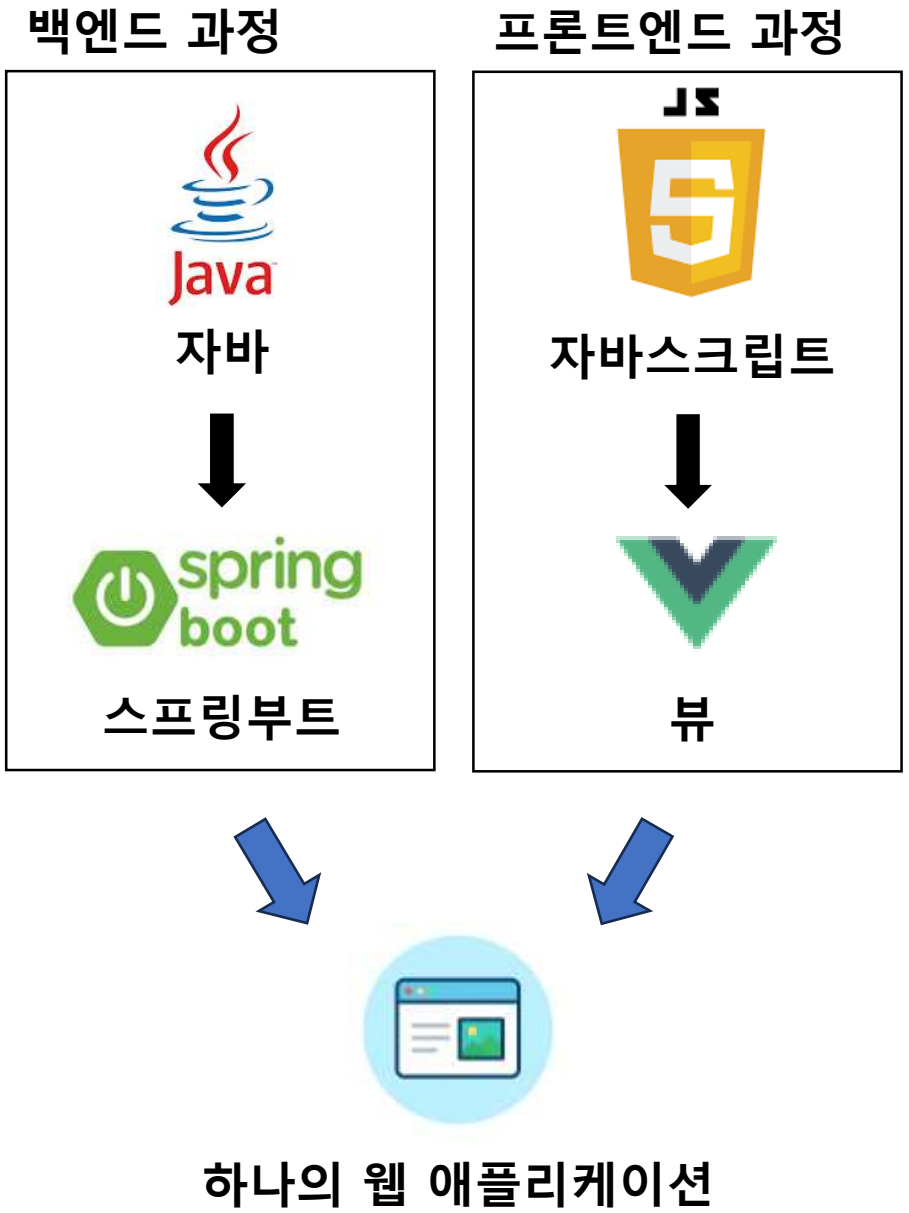
강의 소개

✓정리해보자면

- ✓ 웹 서비스를 만들려면 수많은 기능이 필요
- ✓ 모든 기능을 직접 구현하기엔 너무 어렵다
    - ✓ 그래서 "프레임워크" 나 "라이브러리" 처럼 남들이 만들어둔 도구를 활용
- ✓ 우리가 선택한 도구
    - ✓ Vue : 프론트엔드
    - ✓ Spring Boot : 백엔드

강의 로드맵 소개

- ✓ 전체 강의 로드맵
  - ✓ 백엔드 과정 수료
    - ✓ 자바 언어 학습
    - ✓ 스프링부트 학습
  - ✓ 프론트엔드 과정 수료
    - ✓ 자바스크립트 언어 학습
    - ✓ 학습



Chapter 02

개발 환경 세팅

웹 개발자를 위한 자바 기초

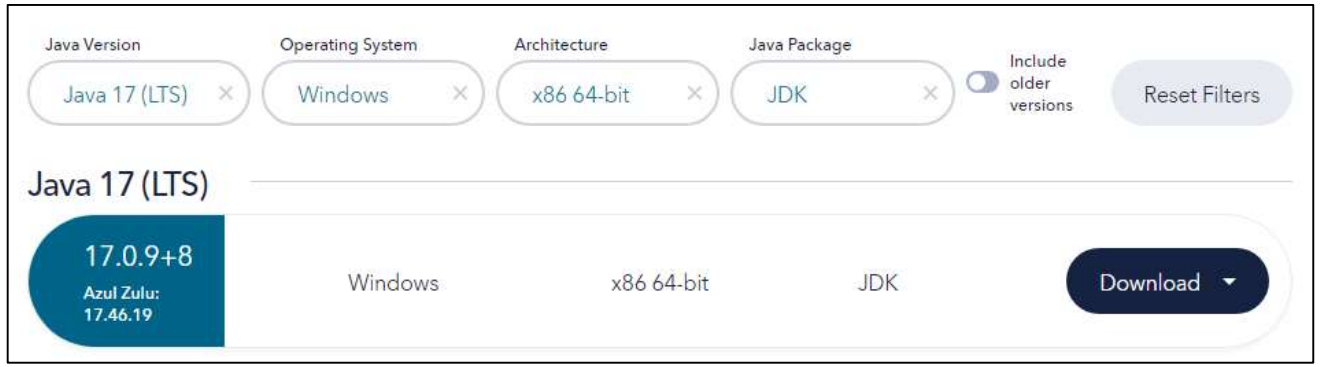
Java 설치

Zulu 17 버전 설치

- Spring Boot 3 최소사양
- Zulu 공식사이트 방문
- https://www.azul.com/downloads/?package=jdk#zulu

- Windows
  - Java Version - Java 17
  - Operating System - macOS 또는 Windows
  - Architecture - x86 64-bit
  - Java Package - JDK
  - Download - msi
- Mac

  Homebrew 설치
  - brew tap mdogan/zulu
  - brew install zulu-jdk17



IntelliJ Community 설치

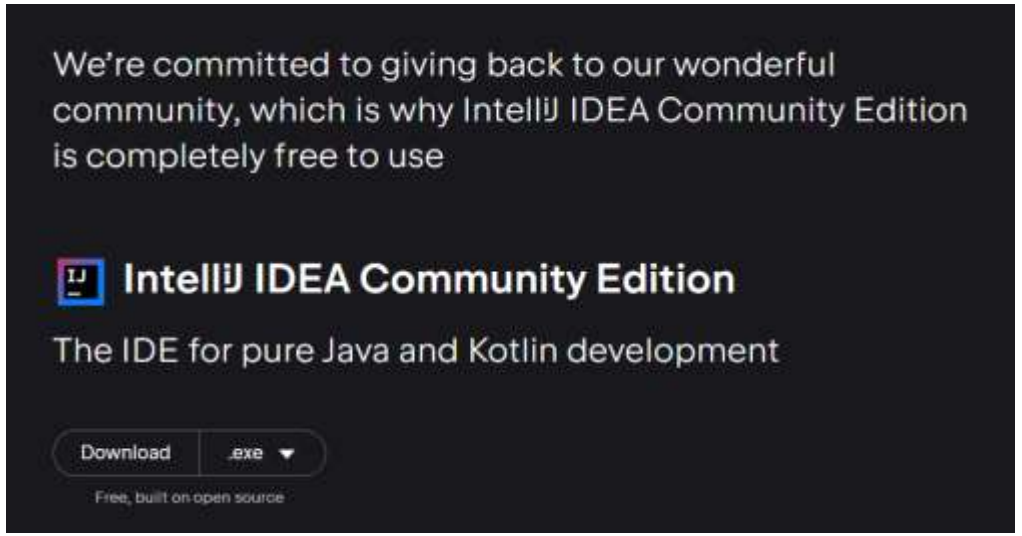
Ultimate 버전 사용자일 경우, 해당 과정 불필요

Windows

- 공식 홈페이지 방문해 인스톨러 다운로드
- https://www.jetbrains.com/idea/download/?section=windows
- 스크롤 내려서 Community Edition 찾아야 함

Mac

- Homebrew 설치
- brew install --cask intellij-idea-ce



Intellij 설치

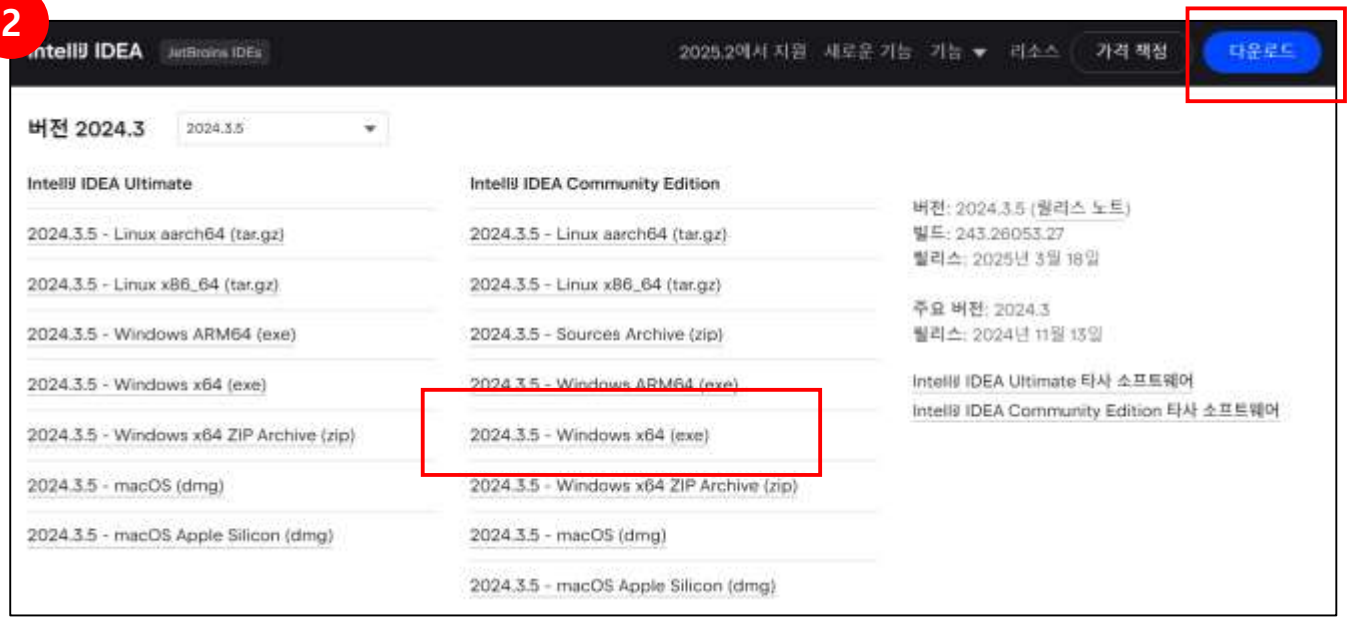
✓강의에서는 사용률이 가장 높은 2024.3 버전을 사용합니다.

1

IntelliJ IDEA Ultimate

Branch Number	IntelliJ Platform Version	Distribution
243	2024.3	31.2%
242	2024.2	11.6%
241	2024.1	15.7%
233	2023.3	8%
232	2023.2	6.5%
231	2023.1	4.4%
223	2022.3	2.9%
222	2022.2	3.6%
221	2022.1	2.8%

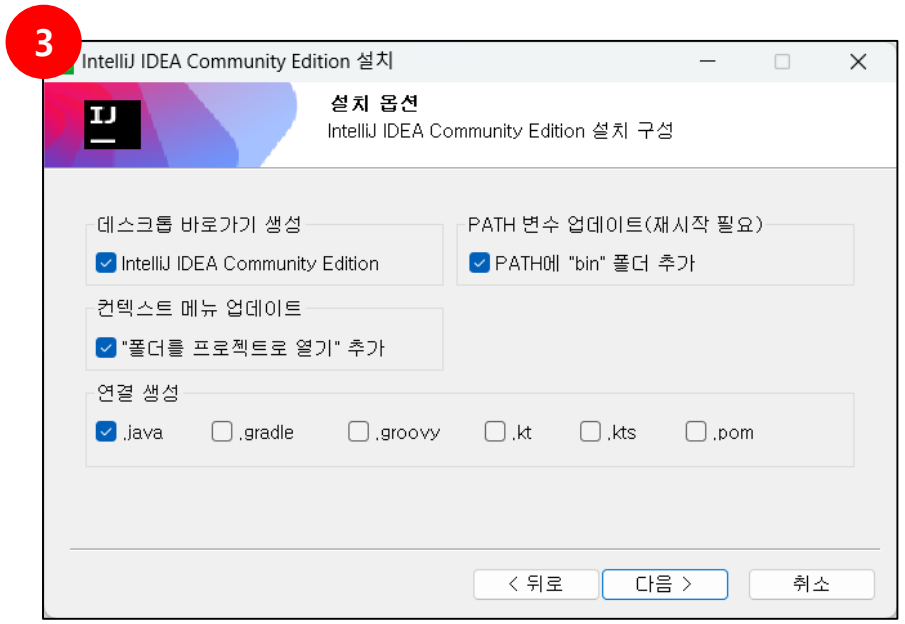
2



공식 홈페이지에서 2024.3 버전 설치

Intellij 설치

✓설치 파일을 실행하고, 아래 설정과 같이 설치를 진행합니다.



- 바로가기 생성
  - 바탕화면에 IntelliJ 아이콘 추가
- PATH 등록
  - 터미널에서 idea 명령어로 실행 가능
- 컨텍스트 메뉴
  - 폴더 우클릭 시 "IntelliJ로 열기" 메뉴 추가
- 확장자 연결
  - 해당 확장자 파일을 더블 클릭할 때 intellij 로 자동 열림

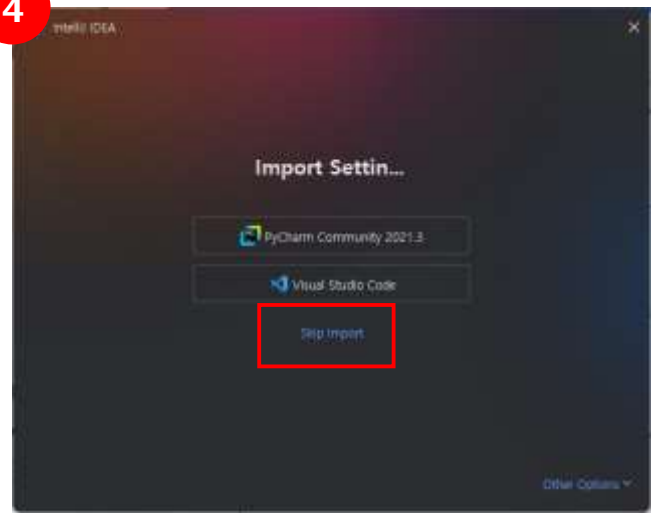
Intellij 설치

✓세팅 후 프로젝트를 생성합니다.

✓다른 에디터 세팅 불러오기

- ✓ Pycharm : 파이썬 개발용 IDE
- ✓ Visual Studio Code : 웹 개발용 IDE

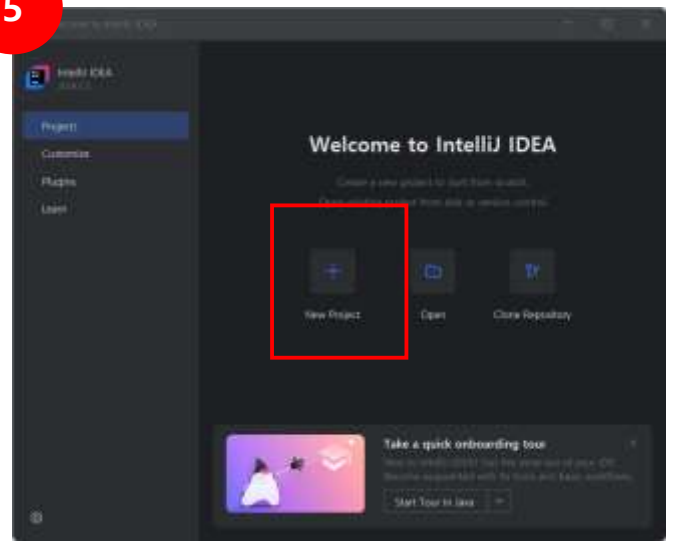
4



실행 화면

Skip import 로 초기 상태로 시작

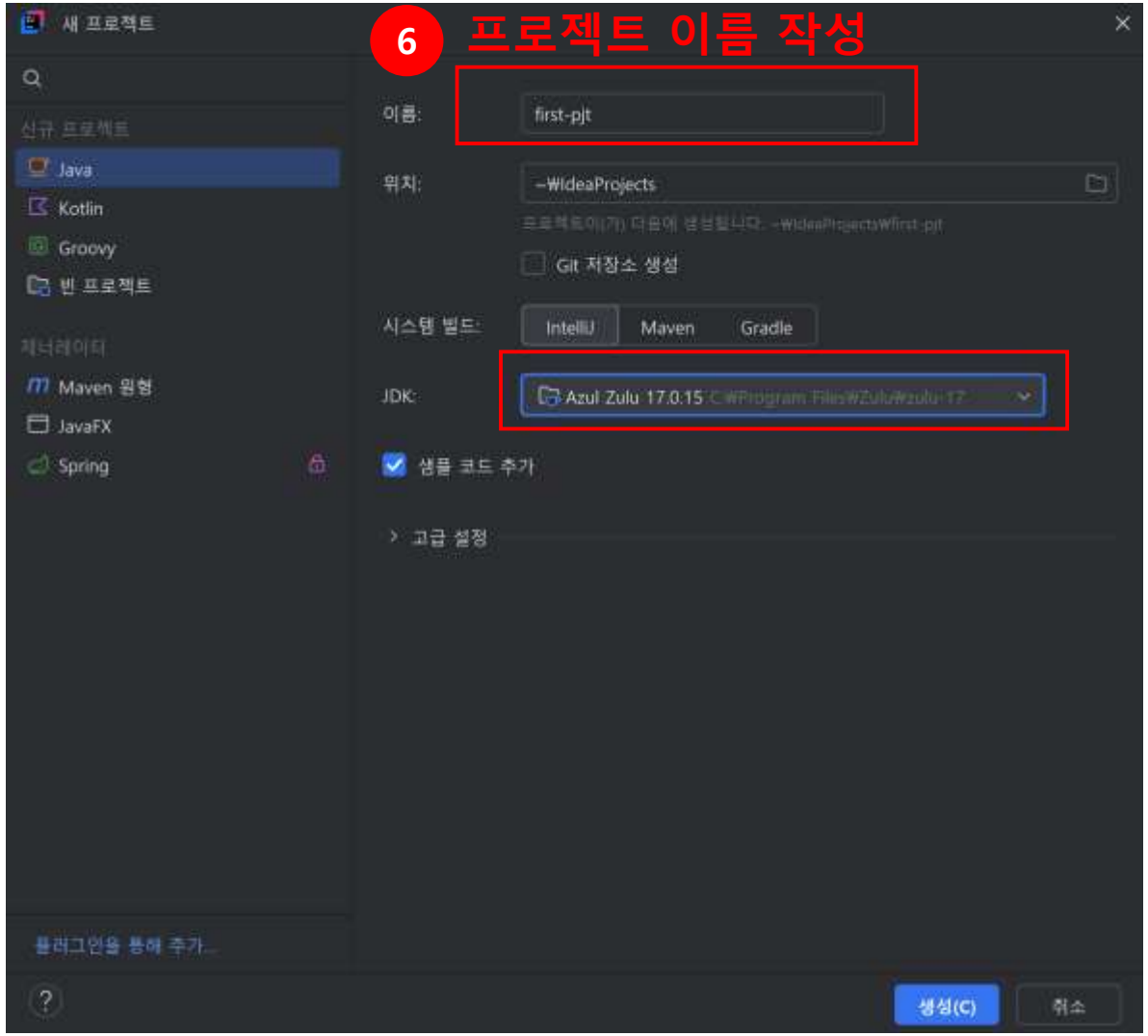
5



New Project 클릭

Intellij 설치

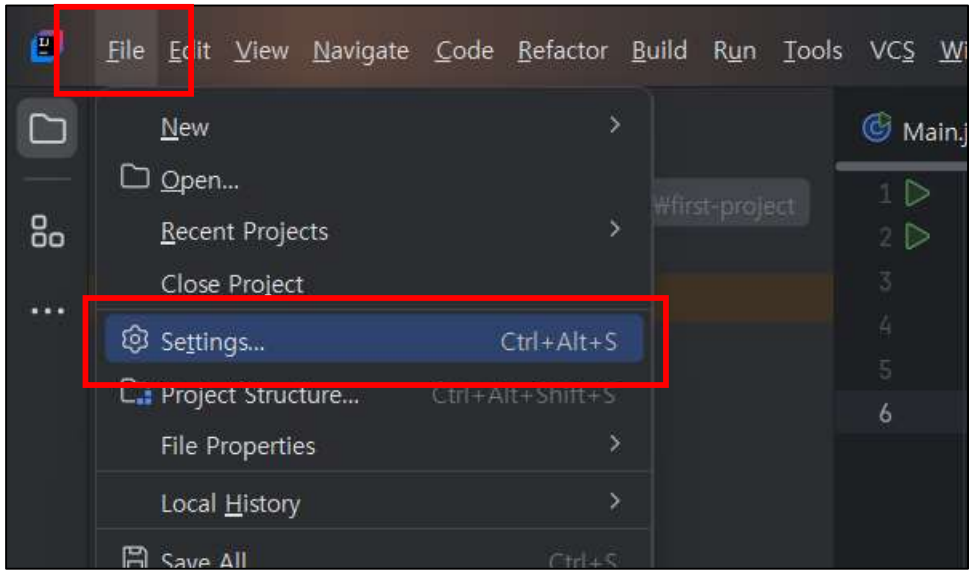
✓세팅 후 프로젝트를 생성합니다.



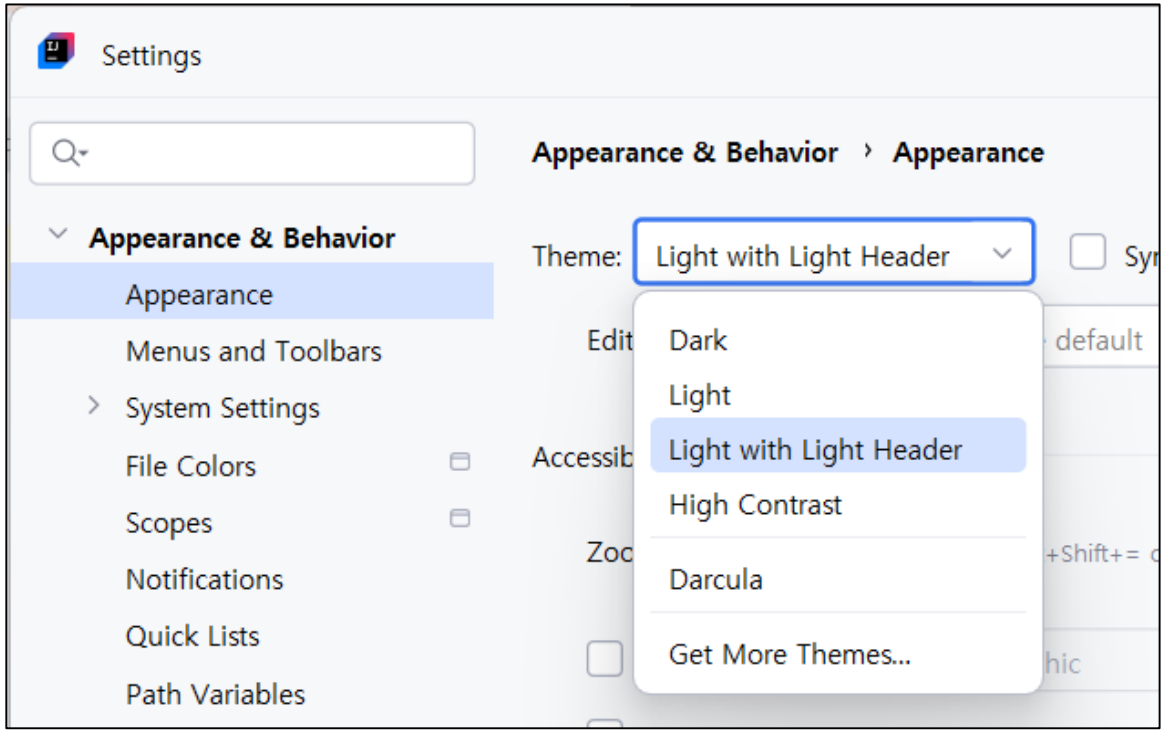
Intellij 설치

✓ 강의에서는 흰 색 테마로 진행합니다.

✓ 원하시는 분은 아래 설정에서 변경하시면 됩니다.



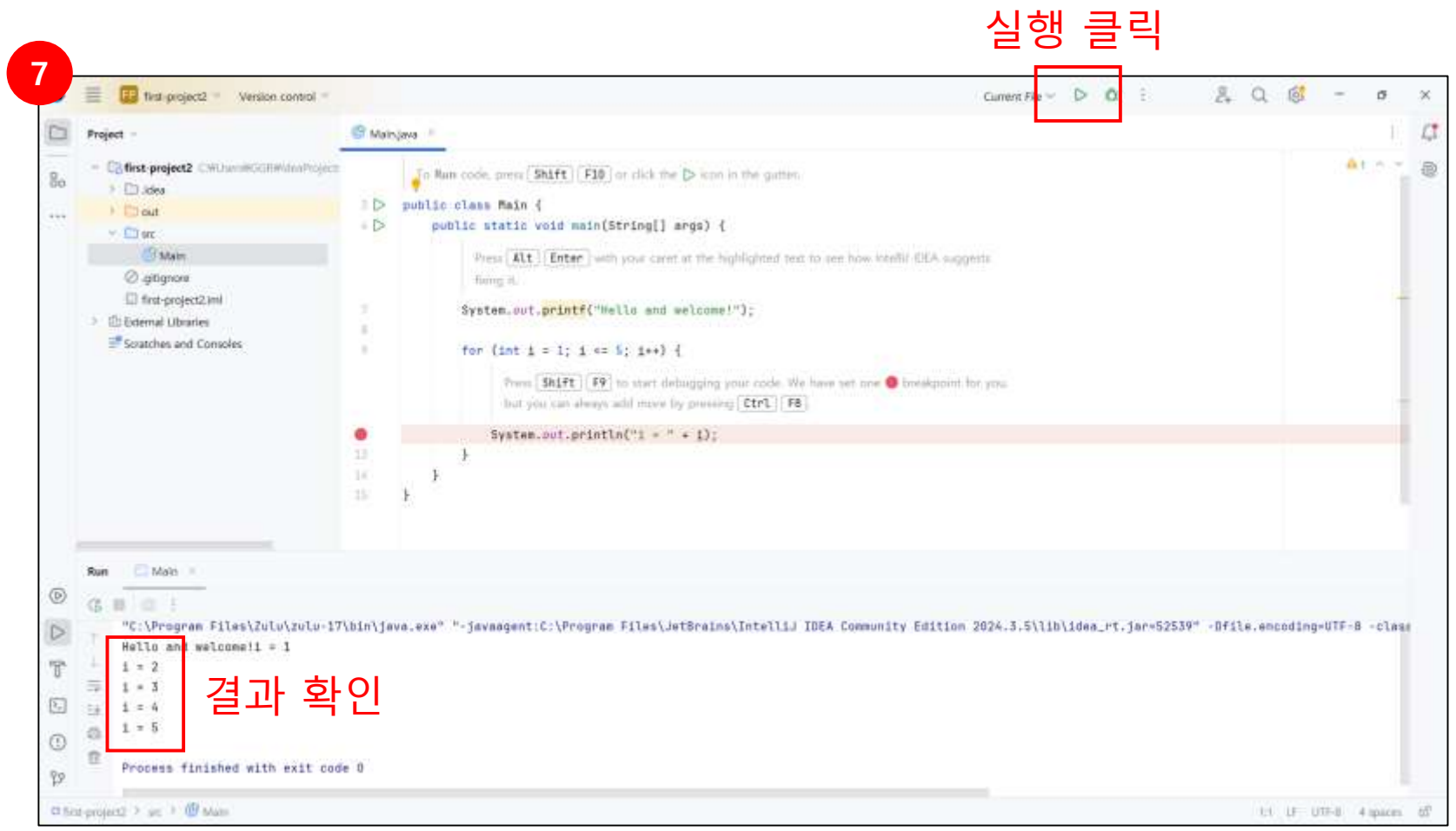
File -> Settings



Theme 를 Light 로 변경

Intellij 설치

✓ 기본 코드 실행으로 설치를 확인합니다.



실습 환경

- ✓ **Windows 기반으로 수업이 진행됩니다**
- ✓ Java JDK
  - ✓ 버전 - **Java 17**
  - ✓ JDK (Java Development Kit) : 자바 개발 도구
  - ✓ Spring Boot 3.x 버전을 지원하는 최소버전 (Java 11 지원x)
- ✓ IntelliJ IDEA - **Community Edition (2025년 이후 버전)**
  - ✓ Java 진영에서 가장 많이 사용되는 IDE

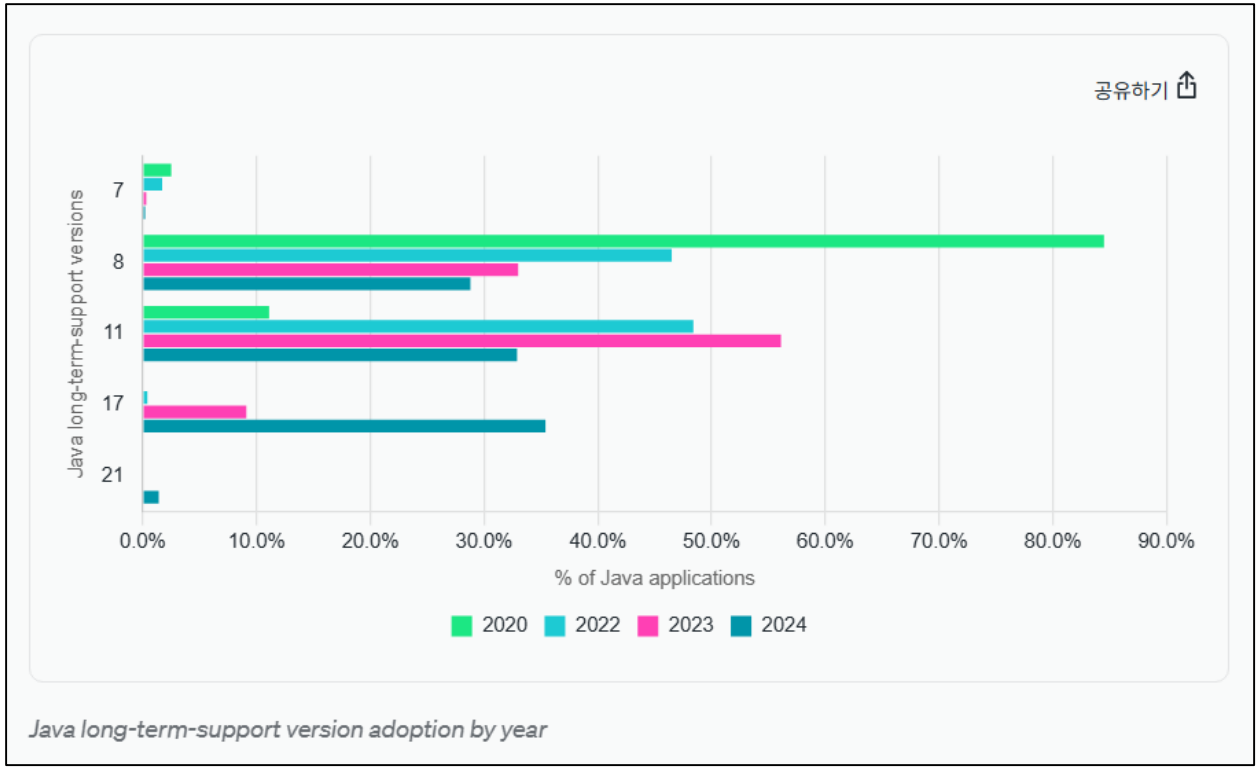
실습 환경

✓ 왜 Java 17 버전인가요 ?

- ✓ 가장 많이 사용되는 버전
- ✓ 최신 버전: 24

✓ 많이 사용된 예전 버전

- ✓ 안정적이다
- ✓ 많은 시스템이 해당 버전으로 개발됨



Java long-term-support version adoption by year

출처 : new relic

실습 환경

- ✓ Intellij 가 eclipse 보다 좋은 점
  - ✓ 자동완성 / 코드제안 / 리팩토링 기능 성능 좋음
  - ✓ HTML / CSS / JS / Spring Bean / Config 자동완성 기능 지원
  - ✓ 디자인 / 성능
  - ✓ JetBrain 개발 -> 빠른 버그 update & 패치

설치 순서

1. Intellij 를 설치한다
2. Intellij 에서 프로젝트를 생성한다
    - 이 때, Java 도 함께 설치하면서 프로젝트를 생성

Chapter 03

변수와 자료형 기초

웹 개발자를 위한 자바 기초

기본 소스코드의 해석

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

- public class Main
  - 자바는 **모든 코드가 클래스 라는 곳 안에 있어야** 실행 가능
  - public 어디서든 접근 허용하겠다는 의미
  - 파일 이름이 Main.java 라면 클래스 이름도 반드시 Main 이어야 함 (자바 규칙)

기본 소스코드의 해석

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

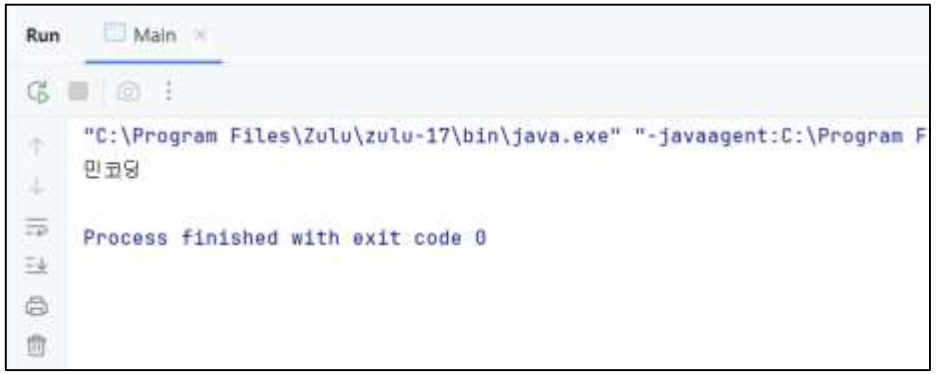
- public static void main(String[] args)
  - 자바 프로그램의 시작점
  - 즉, main 코드부터 실행을 시작함

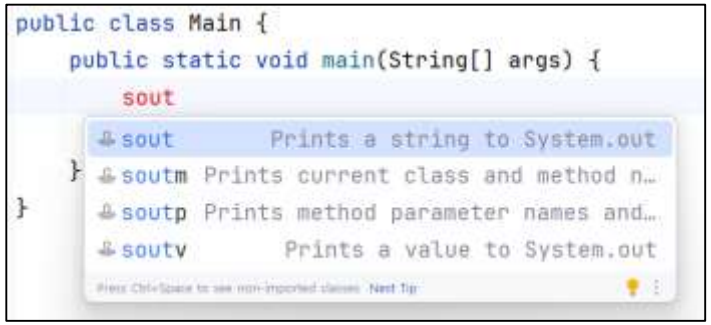
기본 소스코드의 해석

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

- System.out.println("민코딩");
  - 콘솔 출력 명령어
  - "민코딩" 이라는 문자열을 화면에 출력합니다.

- 실행 결과





[참고] sout 입력 시 자동완성 가능

왜 변수를 가장 먼저 배우는가 ?

- ✓ 프로그램
  - ✓ 순서도
  - ✓ 순서도를 작성할때, 데이터가 중복포함된 경우(-128~127 제외)
    - ✓ 숫자, 문자, 결과 출력 등 과정에 데이터가 관여

- ✓ 데이터를 저장(자동으로 진행)하고, 불러오는 방법이 필요
  - ✓ 이름, 나이, 결과값 등을 어딘가에 저장됨, 불러오려면?
  - ✓ 변수 = 이 저장 공간에 이름을 붙인 것

변수란 ?

✓ 데이터(한값)를 저장하기 위한 메모리 공간에 붙인 이름

✓ 숫자(정수)나 문자, 소수점 등을 저장할 수 있다.

✓ 변수(메모리)의 종류 예시

숫자(정수)가 들어가는
int 타입 박스

=인트형 변수

글자가 들어가는
char 타입 변수

=캐릭터형 변수

소수점 숫자가 들어가는
double 타입 박스

=더블형 변수

변수 작명 규칙

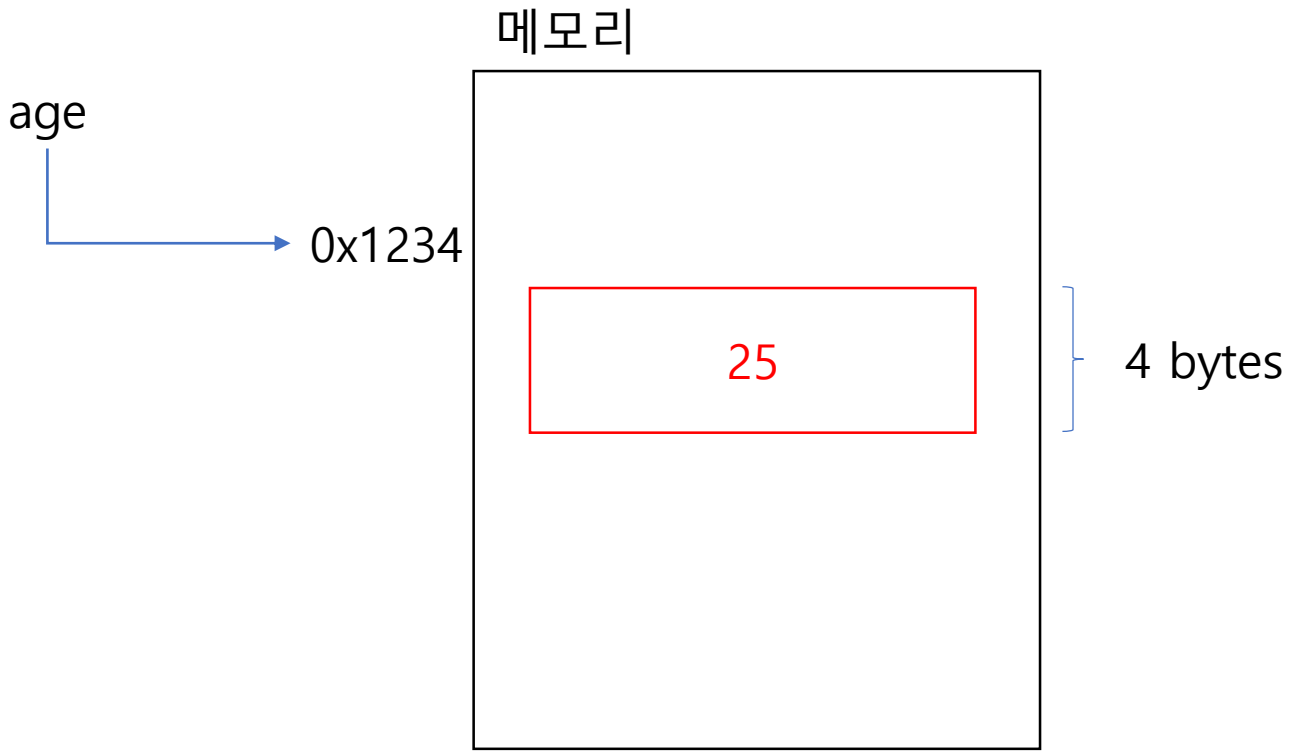
- ✓ 영문자(Aa ~ Zz), 숫자(0~9), _, $ 사용 가능
- ✓ 여러 단어 조합 시 카멜케이스(CamelCase) 사용 권장
  - ✓ 예시) username, totalScore, isLoggedIn
- ✓ 주의 사항
  - ✓ 숫자로 시작 불가 (예 : 1stNumber X)
  - ✓ 공백 사용 불가 (예 : user name X)
  - ✓ 예약어 사용 금지 (예 : int, class, public 등 X)
  - ✓ 대소문자 구분함 (예 : count != Count)

변수는 컴퓨터에 어떻게 저장될까 ?

✓ 변수는 값을 저장하거나 나중에 다시 사용하기 위해 필요

✓ 입력값, 계산 결과, 출력 등 모든 흐름에 관여

✓ 메모리 구조 예시 int age = 25;



웹 개발에서 사용되는 자료형

✓ 자료형

자료형	설명	크기	예시 값
String	문자열을 저장 (사용자 입력, 응답 메시지 등)	문자 수 x 2 bytes (가변 길이)	“Hello”, “123”
int	정수값 저장 (나이, 수량 등)	4 bytes	42, 0, -5
boolean	참/거짓 상태 (로그인 여부 등)	1 byte (JVM)	true, false
long	큰 정수값 저장 (주민번호, 회원 ID 등)	8 bytes	123456789012L
double	소수점이 있는 숫자 (가격, 점수 등)	8 bytes	3.14, 99.99
null	아무 값도 없는 상태	-	없음(null)

배열

✓ 회사 사원들을 관리하기 위해 모두 저장하려고 한다

```
String name1 = "김대리";
String name2 = "이과장";
String name3 = "박부장";
String name4 = "금차장";
```

✓ 팀원이 10명 이상이라면...?

- ✓ 변수를 계속 만들어주어야 한다.
- ✓ 반복적인 데이터를 다룰 때 다른 저장 방법이 필요하다.

배열

✓ 배열이란 ?

- ✓ **같은 자료형**의 값들을 하나의 변수 이름으로 묶어서 저장
- ✓ 각 변수는 **인덱스(index)로 구분**한다.

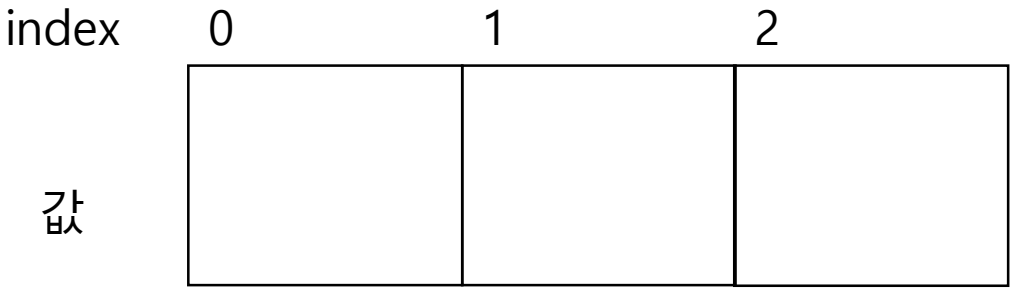
✓ 인덱스

- ✓ 배열 안에 저장된 값 하나하나를 구분하기 위한 번호
- ✓ 배열의 **인덱스는 0부터 시작**

배열 선언과 초기화

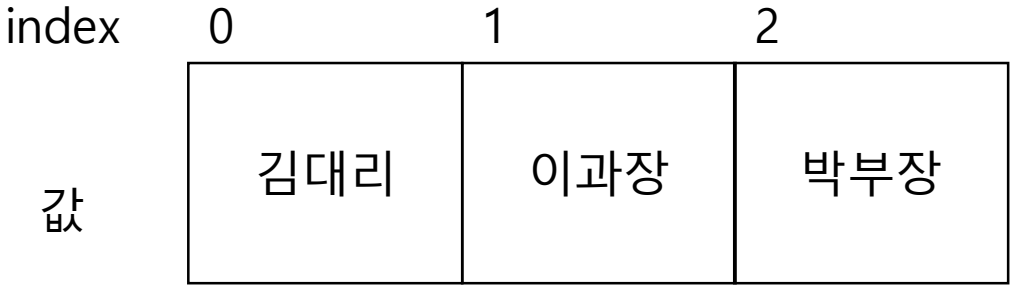
✓방법1. 선언 + 할당

```
String[] names = new String[3];
```



✓방법2. 선언 + 초기값 대입

```
String[] names = {"김대리", "이과장", "박부장"};
```



배열 출력하기

✓인덱스를 활용한 접근

```
String[] names = {"김대리", "이과장", "박부장"};

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

✓[주의] 범위를 벗어나면 오류가 난다.

```
System.out.println(names[3]);
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : Index 3 out of bounds for length 3
    at Main.main(Main.java:15)
```

변수와 자료형 Quiz

✓ 어떤 웹 서비스에서 사용자의 기본 정보를 저장하려고 합니다.

✓ 아래 정보를 저장합니다.

- ✓ 적절한 자료형과 변수명을 생각합니다.
- ✓ 변수로 선언한 후, 예시 값을 대입하세요.

저장할 정보	설명	예시 값
사용자 이름	문자로 저장되는 이름	"김자바"
나이	정수로 저장되는 나이	29
로그인 여부	로그인한 상태인지 아닌지	true
회원 ID	아주 큰 정수 값으로 저장됨	202406131234L
평점	소수점 포함 점수	4.8
휴대폰 번호	아직 입력하지 않음 → **빈 값(null)**	null
관심 있는 태그 목록	사용자가 선택한 태그 3개	"코딩", "자바", "웹개발"

변수와 자료형 Quiz 정답 코드

✓정답 코드

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

        String name = "김자바";
        int age = 29;
        boolean isLoggedIn = true;
        long memberId = 202406131234L;
        double rating = 4.8;
        String phoneNumber = null;

        String[] tags = new String[3];
        tags[0] = "코딩";
        tags[1] = "자바";
        tags[2] = "웹개발";

    }
}
```

형변환이란 ?

✓ 서로 다른 자료형 간에 값을 변환하는 것

✓ 주로 입력 처리, 계산, 출력 시 자주 사용됨

✓ 예시

✓ 사용자가 주문 수량을 입력하면 단가를 곱하여 최종 금액을 출력하기

string → int * double → string

```
String orderCountStr = "3";      // 사용자가 입력한 수량 (문자열)
int unitPrice = 5000;            // 단가

int orderCount = Integer.parseInt(orderCountStr);  // 문자열 → 정수 형변환
int totalPrice = orderCount * unitPrice;           // 최종 금액 계산

System.out.println("총 금액: " + totalPrice + "원");
```

형변환 종류

✓자동 형변환 (묵시적)

- ✓ 작은 타입 -> 큰 타입으로 변환
- ✓ 예) int -> long, int -> double

✓강제 형변환 (명시적)

- ✓ 큰 타입 -> 작은 타입으로 변환
- ✓ 예) double -> int, long -> int

✓문자열 형변환

- ✓ string -> int 등의 기본형으로 변환

```java
// string -> int
String ageStr = "25";
int result = Integer.parseInt(ageStr);
System.out.println(result);  // 정수형 25

// int -> string
int age = 25;
String result2 = String.valueOf(age);
System.out.println(result2); // 문자열 25

// 명시적 형변환 (double -> int)
double pi = 3.14;
int num = (int) pi;
System.out.println(num);    // 결과: 3

// 자동 형변환 (int -> double)
int score = 90;
double result3 = score;
System.out.println(result3);  // 결과: 90.0
```

⚠□주의

강제 형변환은 데이터 손실이 발생할 수 있음

예: double -> int 시 소수점 버려짐

상수와 final

- ✓ 한 번 정해지면 값이 변하지 않는 변수 (값을 변경할 수 없음)
- ✓ 주로 변하지 않는 설정값이나 에러 메시지, 기본값 등에 사용
- ✓ TIP
  - ✓ 읽기 쉬운 이름으로 정의하기
  - ✓ **전체 대문자 + 밑줄로 사용**
- ✓ 예시
  - ✓ `final int MAX_USERS = 100;`
  - ✓ `final String ERROR_MESSAGE = "잘못된 요청입니다.";`

상수와 final

✓ final 키워드

- ✓ 변수나 메서드, 클래스에 **변경 불가 속성**을 부여하는 키워드

✓ 주의사항

- ✓ final 변수는 선언과 동시에 초기화하거나, 1회만 초기화 가능
- ✓ 2번 이상 초기화 시 버그 발생
- ✓ 예시

```
final int X;
X = 10;       // O – 1번은 가능
```

변수와 자료형 기초 Quiz (1/2)

✓간단한 영화 예매 시스템을 개발하고 있다.

- ✓ 필요한 정보들을 변수, 배열, 형변환, 상수를 활용하여 구현하시오.

✓요구사항

- ✓ 예매 가능한 영화는 3편이며, 각각의 제목을 문자열 배열에 저장한다.
- ✓ 영화 가격은 모두 동일하며, 상수로 선언하여 1장당 12000원으로 저장한다.
- ✓ 사용자가 선택한 영화는 "어벤져스", "인사이드 아웃", "라라랜드" 순으로 각각 1장, 2장, 3장을 예매했다고 가정한다.
- ✓ 입력값은 문자열(String)로 주어지며, 이를 정수로 변환하여 계산한다.
- ✓ 총 결제 금액을 출력한다.

변수와 자료형 기초 Quiz (2/2)

✓예제 출력

```
영화: 어벤져스, 예매 수: 1, 금액: 12000원
영화: 인사이드 아웃, 예매 수: 2, 금액: 24000원
영화: 라라랜드, 예매 수: 3, 금액: 36000원

총 결제 금액: 72000원
```

변수와 자료형 기초 Quiz 정답 코드

✓정답 코드

```java
public class Solution {
    public static void main(String[] args) {
        // 상수 선언: 영화 한 장 가격은 고정된 값이므로 final 사용
        final int PRICE_PER_TICKET = 12000;
        // 배열: 영화 제목들을 배열에 저장
        String[] movieTitles = { "어벤져스", "인사이드 아웃", "라라랜드" };
        // 배열 + 문자열 자료형: 예매 수량을 문자열로 저장
        String[] ticketCounts = { "1", "2", "3" };

        // 형변환: 문자열을 정수로 변환 (String → int)
        int count1 = Integer.parseInt(ticketCounts[0]);
        int count2 = Integer.parseInt(ticketCounts[1]);
        int count3 = Integer.parseInt(ticketCounts[2]);

        // 계산: 예매 수량 × 가격 → 영화별 금액 계산
        int total1 = count1 * PRICE_PER_TICKET;
        int total2 = count2 * PRICE_PER_TICKET;
        int total3 = count3 * PRICE_PER_TICKET;
        int grandTotal = total1 + total2 + total3;  // 전체 금액 계산: 영화별 금액 합산

        // 출력: 각 영화 정보 출력
        System.out.println("영화: " + movieTitles[0] + ", 예매 수: " + count1 + ", 금액: " + total1 + "원");
        System.out.println("영화: " + movieTitles[1] + ", 예매 수: " + count2 + ", 금액: " + total2 + "원");
        System.out.println("영화: " + movieTitles[2] + ", 예매 수: " + count3 + ", 금액: " + total3 + "원");
        System.out.println(); // 줄 바꿈
        System.out.println("총 결제 금액: " + grandTotal + "원");  // 총 결제 금액 출력
    }
}
```

Chapter 04

조건과 반복 – 흐름 제어

웹 개발자를 위한 자바 기초

조건문이란 ?

- ✓ 주어진 조건에 따라 코드 실행 흐름을 분기 시키는 문장
- ✓ "만약 ~~ 라면, ~~를 실행하라" 형태의 논리 구조
- ✓ 예시
  - ✓ 지각 할 것 같으면 택시를 탄다.
  - ✓ 회식 메뉴가 소고기라면 참석한다.

if 문 구조

✓조건문 형태

```
if (조건식) {
    // 조건이 참일 때 실행될 코드
} else {
    // 위의 모든 조건이 거짓일 때 실행
}
```

✓조건식은 true 또는 false 를 반환해야 함

if-else / else if

✓다양한 조건을 처리하는 방법

```
if (조건식1) {
    // 조건식1 이 참일 때 실행될 코드
} else if (조건식2) {
    // 조건식1 이 거짓이고, 조건식2 가 참일 때 실행될 코드
} else {
    // 위의 모든 조건이 거짓일 때 실행
}
```

다중 조건 처리 - 중첩 조건문

✓ if 문 안에 또 다른 if 문을 사용하는 구조

✓ 논리적 순서를 분리하거나, 1차 조건 만족 후 세부 조건 검토 시 사용

```
if (조건식1) {
    // 조건식1이 참일 때 실행될 코드
    if (조건식2) {
        // 조건식1 과 조건식2가 모두 참일 때
    }
}
```

논리 연산자

✓ 여러 조건을 한 번에 검사 할 수 있도록 도와주는 기호

1. && (AND 연산자)

✓ "그리고" 의 의미

✓ 모든 조건이 true 일 때만 전체가 true

```
if (조건식1 && 조건식2) {
    // 조건식1 과 조건식2가 모두 참일 때
} else {
    // 위의 모든 조건이 거짓일 때 실행
}
```

논리 연산자

2. || (OR 연산자)

✓ "또는" 의 의미

✓ 조건 중 하나라도 true 면 전체가 true

```
if (조건식1 || 조건식2) {
    // 조건식1 과 조건식2가 모두 참일 때
} else {
    // 위의 모든 조건이 거짓일 때 실행
}
```

논리 연산자

3. ! (NOT 연산자)

- ✓ "아니면" 의 의미
- ✓ 조건식을 반대로 바꾼다.
- ✓ true -> false / false -> true

```
if (!조건식1) {
    // 조건식1 이 거짓이라면 실행
} else {
    // 조건식1이 참이라면 실행
}
```

[도전]조건문

✓ 여행지를 선택하고, 나이와 회원 여부에 따라 추천 혜택을 받는다.

✓ 여행지 이름과 예약 수량, 회원 여부와 나이가 다음과 같이 저장되어 있다.

```
String[] destinations = { "제주도", "강릉", "부산" };
String[] bookingCounts = { "2", "0", "1" };
boolean isMember = true;
int age = 25;
```

✓ 요구사항

✓ 각 여행지의 예약 현황을 출력한다. (예약 수량은 정수로 변환하여 활용)

✓ 1인당 여행 비용은 100,000원으로 고정되어 있다.

✓ 전체 여행 예약 수가 3건 이상이고, 회원이며, 나이가 20세 이상이면

- ✓ "여행 추천 혜택 대상" 출력
- ✓ 아니라면 "일반 고객" 출력

출력 결과

```
[예약 현황]
제주도: 2건
강릉: 0건
부산: 1건

총 금액: 300000원
여행 추천 혜택 대상
```

조건문 Quiz 정답 코드

✓정답 코드

```java
final int PRICE = 100000;

String[] destinations = { "제주도", "강릉", "부산" };
String[] bookingCounts = { "2", "0", "1" };
boolean isMember = true;
int age = 25;

// 형변환
int count1 = Integer.parseInt(bookingCounts[0]);
int count2 = Integer.parseInt(bookingCounts[1]);
int count3 = Integer.parseInt(bookingCounts[2]);

int totalBookings = count1 + count2 + count3;
int totalPrice = totalBookings * PRICE;

// 출력
System.out.println("[예약 현황]");
System.out.println(destinations[0] + ": " + count1 + "건");
System.out.println(destinations[1] + ": " + count2 + "건");
System.out.println(destinations[2] + ": " + count3 + "건");
System.out.println("\n총 금액: " + totalPrice + "원");

// 조건문
if (totalBookings >= 3 && isMember && age >= 20) {
    System.out.println("여행 추천 혜택 대상");
} else {
    System.out.println("일반 고객");
}
```

반복문

✓코드를 여러 번 실행하고 싶을 때 사용하는 중요한 문법

✓조건을 만족하는 동안 특정 코드를 반복 실행하는 구조

반복문 종류	기본 구조	사용 지점
for	횟수 기반	반복 횟수가 정해진 경우
while	조건 기반	반복 횟수를 모를 때

for 문

✓반복 횟수가 정해진 경우 활용

- ✓ 시작부터 조건이 만족할 때 까지 특정 횟수를 반복

✓for문 기본 형태

```
for (시작점; 조건식; 매번 반복 후) {
    // 조건이 만족하는 동안 반복될 코드
}
```

✓기본 예제

- ✓ 횟수 반복 (“안녕하세요” 5번 출력)

```java
for (int i = 0; i < 5; i++) {
    System.out.println("안녕하세요");
}
```

for 문 – 반복 가능한 객체 반복

✓배열 반복에서는 두 가지 방법을 쓸 수 있다.

1. 기본 for 문

- ✓ 인덱스를 사용해서 직접 요소에 접근
- ✓ 반복 범위를 조절하거나 부분 반복할 수 있음

```
String[] names = {"김대리", "이과장", "박부장", "금차장"};
for (int i = 0; i < names.length; i++) {
    System.out.println(names[i] + "님 안녕하세요");
}
```

for 문 – 반복 가능한 객체 반복

2. for-each 문 (향상된 for문)

- ✓ 인덱스가 필요 없고, 배열의 값만 필요할 때 활용
- ✓ 간결하고 읽기 쉬움

```
String[] names = {"김대리", "이과장", "박부장", "금차장"};
for (String name : names) {
    System.out.println(name + "님 안녕하세요");
}
```

while 문

- ✓조건이 참인 동안 코드를 계속 반복하는 문법
- ✓반복 횟수가 미리 정해지지 않은 상황에 유용
- ✓기본 구조

```
while (조건식) {
    // 조건이 true 인 동안 실행될 코드
}
```

while 문

✓기본 예제

✓ 1 ~ 5 숫자 세기

```
int i = 1;
while (i <= 5) {
    System.out.println(i + "번째 반복");
    i++;
}
```

while 문

✓기본 예제2

- ✓ 비밀번호 재입력
- ✓ 특정 변수 값에 따라

  반복 여부가 정해지는 상황

```java
import java.util.Scanner;

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

        Scanner sc = new Scanner(System.in);
        String input = "";

        while (!input.equals("1234")) {
            System.out.print("비밀번호를 입력하세요: ");
            input = sc.nextLine();
        }
        System.out.println("접속 성공!");
    }
}
```

[참고] Scanner: 키보드 입력을 받을 수 있게 해주는 도구

조건과 반복 Quiz

✓ 문제

- ✓ 여러 개의 여행지가 있고, 각각은 계절 정보를 포함합니다.
- ✓ 사용자에게 계절을 입력 받아, 해당 계절에 맞는 여행지를 추천해주는 프로그램을 작성하세요.
- ✓ 여행지 목록과 계절 정보는 인덱스로 연결되어 있음
    - ✓ places[0] = 제주도, seasons[0] = 여름 -> 제주도는 여름에 추천

```
const places = ["제주도", "강릉", "부산", "설악산", "여수"];
const seasons = ["여름", "겨울", "여름", "가을", "봄"];
```

- ✓ 일치하는 여행지가 없다면 “해당 계절의 추천 여행지가 없습니다.” 출력

조건과 반복 Quiz

✓ 실행 예시

✓ 사용자 입력 대기

```
추천받고 싶은 계절을 입력하세요:
```

✓ 여름 입력 시

```
추천받고 싶은 계절을 입력하세요: 여름
[추천 여행지]
- 제주도
- 부산
```

✓ 없는 계절 입력 시

```
추천받고 싶은 계절을 입력하세요: 봄같은 가을
[추천 여행지]
해당 계절의 추천 여행지가 없습니다.
```

조건과 반복 Quiz 정답 코드

✓정답 코드

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String[] places = {"제주도", "강릉", "부산", "설악산", "여수"};
        String[] seasons = {"여름", "겨울", "여름", "가을", "봄"};

        Scanner sc = new Scanner(System.in);
        System.out.print("추천받고 싶은 계절을 입력하세요: ");
        String inputSeason = sc.nextLine();

        boolean found = false;

        System.out.println("[추천 여행지]");
        for (int i = 0; i < places.length; i++) {
            if (seasons[i].equals(inputSeason)) {
                System.out.println("- " + places[i]);
                found = true;
            }
        }

        if (!found) {
            System.out.println("해당 계절의 추천 여행지가 없습니다.");
        }
    }
}
```

Chapter 05

함수와 파라미터

웹 개발자를 위한 자바 기초

중복되는 로직의 문제점

✓ 같은 코드가 여러 번 중복되는 경우

✓ 예제

✓ 각 여행지의 가격이 정해져 있고, 할인 쿠폰을 적용할 수 있다.

✓ 할인 쿠폰 = (2만원 * 10% 추가) 할인

```
int jejuPrice = 200000;
System.out.println("제주도 가격: " + (jejuPrice - 20000 * 1.1) + "원");

int gangneungPrice = 150000;
System.out.println("강릉 가격: " + (gangneungPrice - 20000 * 1.1) + "원");

int busanPrice = 180000;
System.out.println("부산 가격: " + (busanPrice - 20000 * 1.1) + "원");
```

중복되는 로직의 문제점

✓ 다음과 같은 상황에서 코드 유지보수를 어떻게 해야 할까 ?

- ✓ 할인 금액이 변경되는 경우
- ✓ 추가 할인 비율이 변경되는 경우
- ✓ 금액을 관리해야 할 도시가 더 많을 경우

```
int jejuPrice = 200000;
System.out.println("제주도 가격: " + (jejuPrice - 20000 * 1.1) + "원");

int gangneungPrice = 150000;
System.out.println("강릉 가격: " + (gangneungPrice - 20000 * 1.1) + "원");

int busanPrice = 180000;
System.out.println("부산 가격: " + (busanPrice - 20000 * 1.1) + "원");
```

같은 로직이 반복되는 경우

리스크 ⬆

시간 낭비 ⬆

함수란

✓로직이 동일하고, 입력 값만 다를 때 코드를 따로 묶어주자

✓예제

```
public static int getDiscountedPrice(int basePrice, int discountAmount, double discountRate) {  3 usages
    return (int)(basePrice - discountAmount * discountRate);
}                                                                                          함수

public static void main(String[] args) {
    int discountAmount = 20000;
    double discountRate = 1.1;

    int jejuPrice = getDiscountedPrice( basePrice: 200000, discountAmount, discountRate);
    int gangneungPrice = getDiscountedPrice( basePrice: 150000, discountAmount, discountRate);
    int busanPrcie = getDiscountedPrice( basePrice: 180000, discountAmount, discountRate);

    System.out.println("제주도 가격: " + jejuPrice + "원");
    System.out.println("강릉 가격: " + gangneungPrice + "원");
    System.out.println("부산 가격: " + busanPrcie + "원");
}
```

함수 기초

✓함수 기본 구조

```
public static 반환값형태 함수이름(전달받을 값) {
    // 실행할 코드
}
```

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

- ✓ public static : 어느 곳 에서든 접근 가능하다 (객체지향에서 상세히 설명)
- ✓ void : 반환 값 없음
- ✓ main : 함수 이름 (자바 코드가 실행될 기본 함수이름)

기본 함수 만들어보기

✓기본 출력하는 기능만 따로 묶어보기

```
public static void sayHello() {  1 usage
    System.out.println("환영합니다!");
}

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

함수 정의 : 함수가 어떤 일을 할 지 적어두는 것

함수 호출 : 정의한 함수를 실행

파라미터 (Parameter)

✓ 외부에서 값을 전달 받아야 하는 경우

- ✓ 함수 내에서 이름을 함께 출력하고 싶은 경우
- ✓ 호출할 때마다 다른 값을 전달받아서 활용

파라미터

```
public static void sayHello(String name) {  2 usages
    System.out.println("환영합니다! " + name + "님!");
}

public static void main(String[] args) {
    sayHello( name: "자바");    // "환영합니다! 자바님!" 출력
    sayHello( name: "스프링");  // "환영합니다! 스프링님!" 출력
}
```

전달인자 (Argument)

이름을 전달받아, 함수 내부에서 활용

파라미터 : 정의 시 받기로 한 값

함수 호출 시 이름을 전달

인자 (Argument) : 호출 시 전달한 실제 값

파라미터 예제

✓기본 예제

✓ 숫자 2개를 전달받아 더한 값을 출력해보기

✓예제 정답 코드

```
public static void getSum(int a, int b) {
    System.out.println(a + b);
}

public static void main(String[] args) {
    getSum( a: 3,  b: 9);
}
```

반환값 (Return Value)

✓함수 내에서 실행한 결과를 외부에서 사용할 수 있다.

✓숫자 2개를 전달받아 더한 값을 반환

정수형을 반환

```
public static int getSum(int a, int b) {  1 usage
    return a + b;  // a + b 값을 반환
}

public static void main(String[] args) {
    // 대입연산자를 활용하여 함수의 반환값을 변수에 저장
    int result = getSum(   3,    9);
    System.out.println(result);
}
```

정수형을 전달 받음

- 주의사항

반환값을

전달할 때와 전달받을 때

자료형이 동일해야 한다

✓ return : 함수가 결과값을 돌려줄 때 사용하는 키워드

✓ 반환형 : 어떤 자료형을 return 할 지 지정 (int, String 등)

파라미터 Quiz

✓ 가격 계산기 만들어보기

- ✓ 이름이 getDiscountedPrice 인 함수를 정의한다.
- ✓ 함수 구성
  - ✓ 아래 3가지 데이터를 전달 받는다.
    - ✓ basePrice : 기본 가격(int)
    - ✓ discountAmount: 할인 가격(int)
    - ✓ discountRate: 추가 할인 비율(double)
  - ✓ 기본 가격 – (할인 가격 * 추가 할인 비율) 을 반환한다.
- ✓ 아래 조건에 맞게 3번 호출한다.
  - ✓ 할인 가격: 20,000, 할인 비율: 1.1
  - ✓ 기본 가격: 제주도(200000), 강릉(150000), 부산(180000)

- 출력 결과

```
제주도 가격: 178000원
강릉 가격: 128000원
부산 가격: 158000원
```

파라미터 Quiz 정답 코드

✓정답 코드

```java
public static int getDiscountedPrice(int basePrice, int discountAmount, double discountRate) {  3 usages
    return (int)(basePrice - discountAmount * discountRate);
}

public static void main(String[] args) {
    int discountAmount = 20000;
    double discountRate = 1.1;

    int jejuPrice = getDiscountedPrice( basePrice: 200000, discountAmount, discountRate);
    int gangneungPrice = getDiscountedPrice( basePrice: 150000, discountAmount, discountRate);
    int busanPrcie = getDiscountedPrice( basePrice: 180000, discountAmount, discountRate);

    System.out.println("제주도 가격: " + jejuPrice + "원");
    System.out.println("강릉 가격: " + gangneungPrice + "원");
    System.out.println("부산 가격: " + busanPrcie + "원");
}
```

Chapter 06

데이터 구조 기본

웹 개발자를 위한 자바 기초

현재까지 배운 저장 방식

✓ 현재까지 배운 저장 방식의 한계

- ✓ 변수 : 하나의 데이터만 저장 가능
- ✓ 배열 : 여러 개를 저장하지만, 기능이 제한적

✓ 현실 세계엔 다양한 저장 방식의 요구가 존재

- ✓ 순서가 중요한 경우
- ✓ 검색이 빠른 게 중요한 경우
- ✓ 중복을 막는 게 중요한 경우
- ✓ 등등

데이터 구조 (자료구조) 란 ?

✓ 데이터 구조 (Data Structure)

- ✓ 데이터를 효율적으로 저장하고 관리하기 위한 구조

✓ 웹 개발에서 자주 활용되는 자료구조

- ✓ ArrayList
- ✓ HashMap
- ✓ HashSet, TreeMap 등등

ArrayList

✓ ArrayList 란 ?

- ✓ 배열처럼 여러 개의 데이터를 저장하지만, 배열보다 훨씬 더 유연하게 동작하는 자료구조
- ✓ java.util 패키지 안에 들어있다.

✓ 기본 코드

```
// 배열 : 고정된 크기를 지정
String[] names = new String[3];
names[0] = "김선배";
names[1] = "박후배";
names[2] = "금선생";

// ArrayList : 비어있는 상태로 시작
ArrayList<String> namesList = new ArrayList<>();
namesList.add("김선배");
namesList.add("박후배");
namesList.add("금선생");
```

배열 vs ArrayList

✓ 배열과 ArrayList 의 차이

항목	배열 (String[])	ArrayList (ArrayList<String>)
크기	고정	자동으로 늘어남
순서	있음	있음
인덱스로 접근	가능	가능
추가/삭제	불편함. 수동으로 처리해줘야 함	편리함. 기능들을 제공해 줌
크기 변경	불가능	가능 (자동으로 확장됨)

ArrayList 연습

✓ ArrayList 사용해보기

- ✓ 미리 만들어진 기능(methods) 들을 활용

✓ 회의 참석자 명단 관리 프로그램

- ✓ 참석자 이름을 리스트에 추가하고 출력

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        // ArrayList 선언
        ArrayList<String> attendees = new ArrayList<>();

        // 참석자 추가
        attendees.add("김대리");
        attendees.add("이과장");
        attendees.add("박부장");

        // 한 명 더 추가
        attendees.add("최사원");

        // 출력
        System.out.println("총 참석자 수: " + attendees.size());
        System.out.println("참석자 명단:");
        for (String name : attendees) {
            System.out.println("- " + name);
        }
    }
}
```

ArrayList 주요 메서드

✓ ArrayList 를 잘 활용하기 위한 필수 기능들

메서드 이름	설명	예시
add(E e)	요소 추가	list.add("홍길동");
add(int index, E e)	특정 위치에 요소 삽입	list.add(1, "유관순");
get(int index)	해당 인덱스의 요소 가져오기	list.get(0);
set(int index, E e)	해당 인덱스 값 수정	list.set(0, "이순신");
remove(int index)	인덱스로 요소 삭제	list.remove(1);
remove(Object o)	값으로 요소 삭제	list.remove("홍길동");
size()	전체 요소 개수 반환	list.size();
contains(Object o)	해당 값이 있는지 확인	list.contains("유관순");
indexOf(Object o)	값이 있는 인덱스 반환 (없으면 -1)	list.indexOf("홍길동");
clear()	전체 요소 삭제	list.clear();
isEmpty()	리스트가 비어 있는지 확인	list.isEmpty();

HashMap

✓ ArrayList 의 한계점

- ✓ ArrayList 는 단순한 데이터 목록
- ✓ 100개의 데이터 중 특정 정보를 하나를 조회하고 싶다
  - ✓ 하나 씩 확인하면서 검색
  - ✓ 별도의 index 나 별도의 데이터로 따로 저장해 놓아야 함

✓ HashMap 이란 ?

- ✓ Key(키) 와 Value(값) 쌍으로 데이터를 저장하는 자료구조
- ✓ 사전처럼 (이름->전화번호), (아이디->사용자 정보) 처럼 쌍으로 연결하여 데이터를 저장

HashMap

✓기본 코드

```java
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        // HashMap: 키와 값을 쌍으로 저장
        HashMap<String, String> phoneBook = new HashMap<>();

        // put(key, value): 값 추가
        phoneBook.put("김선배", "010-1234-5678");
        phoneBook.put("박후배", "010-2345-6789");
        phoneBook.put("금선생", "010-3456-7890");

        // get(key): 값 조회
        System.out.println("김선배 번호: " + phoneBook.get("김선배"));  // 010-1234-5678
        System.out.println("박후배 번호: " + phoneBook.get("박후배"));  // 010-1234-5678
    }
}
```

HashMap 주요 메서드

✓ HashMap 을 잘 활용하기 위한 필수 기능들

메서드 이름	설명	예시
put(K key, V value)	값 추가	map.put("이름", "전화번호")
get(K key)	키로 값 조회	map.get("이름")
remove(K key)	삭제	map.remove("이름")
containsKey(K key)	존재 여부 확인	map.containsKey("이름")
size()	전체 개수	map.size()
clear()	전체 삭제	map.clear()
isEmpty()	비어있는지	map.isEmpty()

그 외 자주 사용되는 자료구조

✓정리

자료구조	특징	주요 사용 상황
HashSet	중복을 허용하지 않는 집합	태그, 중복 제거, 체크리스트
LinkedList	삽입/삭제에 특화된 리스트	큐, 스택 구현 또는 요소 자주 추가/삭제 시
PriorityQueue	우선순위 기반 큐 (작은 값 우선 처리)	작업 스케줄링, 힙 구조 활용
TreeMap	자동 정렬되는 Map	정렬이 필요한 키-값 쌍 관리
TreeSet	자동 정렬되는 집합 (오름차순 기본)	정렬된 유일한 값 목록
Stack	후입선출 (LIFO) 구조	뒤로 가기, 괄호 검사, 계산기
Queue	선입선출 (FIFO) 구조	대기열, 작업 순서 처리

Chapter 07

객체 지향 프로그래밍의 이해

웹 개발자를 위한 자바 기초

객체 지향이란 ?

✓ 객체 지향 프로그래밍(OOP, Object-Oriented Programming)

- ✓ 현실 세계의 사물(객체)을 코드로 표현하여 프로그램을 만들고 관리하는 방식
- ✓ 즉, 현실에 존재하는 것들을 **객체**로 보고, **객체들이 정보와 기능을 갖도록 하는 방식**

✓ 객체 지향의 구성 요소

구성요소	설명	예시 (자동차)
클래스	객체를 만들기 위한 설계도	자동차 설계도
객체	정보와 기능을 담은 데이터	내 차 (회색, 연비, 브레이크 기능 등)
속성(변수)	객체가 가진 정보	색상, 연비
메서드(함수)	객체가 할 수 있는 행동	엑셀 기능, 브레이크 기능

클래스와 객체 예시

✓카페 주문 시스템

✓ ㅁ 💼 "김선배는 아아(아이스 아메리카노) 하나랑 박후배는 라떼, 금선생은 바닐라라떼~"

✓ 이걸 코드로 짜본다면 아래와 같다.

```
String name1 = "김선배";
String drink1 = "아이스 아메리카노";

String name2 = "박후배";
String drink2 = "라떼";

String name3 = "금선생";
String drink3 = "바닐라라떼";

System.out.println(name1 + "의 주문: " + drink1);
System.out.println(name2 + "의 주문: " + drink2);
System.out.println(name3 + "의 주문: " + drink3);
```

불편한 점

•사람이 많아지면 name1, name2, drink5 ... 관리 폭발

•실수하기 쉬움 (순서 헷갈림)

•음료와 사람 정보가 흩어져 있어 유지보수 어렵다!

클래스와 객체 예시

✓카페 주문 시스템 - 객체지향

✓ 주문 하나를 객체로 표현

```
class Order {  6 usages
    String name;  4 usages
    String drink;  4 usages

    void printOrder() {  3 usages
        System.out.println(name + "의 주문: " + drink);
    }
}
```

✓ class 로 (데이터 + 기능) 설계

✓ 각 객체가 자기 데이터를 관리 (인스턴스)

✓ 함수 재사용

사용 예시

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

        Order order1 = new Order();
        order1.name = "김선배";
        order1.drink = "아이스 아메리카노";

        Order order2 = new Order();
        order2.name = "박후배";
        order2.drink = "라떼";

        Order order3 = new Order();
        order3.name = "금선생";
        order3.drink = "바닐라라떼";

        order1.printOrder();
        order2.printOrder();
        order3.printOrder();

    }
}
```

왜 객체 지향이 필요할까 ?

✓ 변수와 관련 함수가 흩여져 있어서 관리가 어렵다.

- ✓ 관련된 데이터와 함수를 하나의 객체로 묶는다. (가독성 / 유지보수⬆)

✓ 같은 종류의 데이터를 계속 새로운 변수로 생성 해야 한다.

- ✓ 클래스 설계도로 객체를 복사하여 사용 (재사용성⬆)

✓ 기능을 재활용 하려면 코드를 복사해야 한다.

- ✓ 공통 기능은 클래스로 묶고 상속/구성으로 공유

✓ 프로그램 구조가 현실과 동떨어져 있다.

- ✓ 현실 세계의 개념을 코드로 구조화 가능

클래스를 통해 객체를 생성하는 방법

✓생성자

✓ 자바에서 객체를 처음 만들 때만 호출되는 특별한 메서드

✓ 클래스를 실체화해서 사용할 수 있게 해주는 **초기화 담당**

```
Order order1 = new Order( name: "김선배",  drink: "아이스 아메리카노");
```

✓생성자 기본 구조

```
class 클래스이름 {
    클래스이름 (매개변수) {
        // 필드 초기화
    }
}
```

```
class Order {  2 usages
    String name;  3 usages
    String drink;  3 usages

    Order(String name, String drink) {  1 usage
        this.name = name;
        this.drink = drink;
    }

    void printOrder() {  1 usage
        System.out.println(name + "의 주문: " + drink);
    }
}
```

객체 지향 개념 정리

✓ 객체 지향은 현실처럼 데이터와 기능을 묶어서

코드의 구조화, 유지보수, 확장성을 가능하게 해주는 방식

✓ 객체 지향의 구성요소

구성요소	설명	예시 (카페 주문)
클래스	객체를 만들기 위한 설계도	이름, 메뉴, 주문 기능
객체	정보와 기능을 담은 데이터	김선배, 아이스 아메리카노
속성(변수)	객체가 가진 정보	이름, 메뉴
메서드(함수)	객체가 할 수 있는 행동	주문 기능

자바는 객체 지향 언어이다

- ✓모든 코드(변수, 함수)는 클래스 안에 있어야 함
  - ✓코드의 최소 단위를 개체 중심으로 구현함
- ✓모든 데이터가 “클래스 기반 객체”로 구성됨
  - ✓정수, 문자열도 사실은 클래스 기반 (Integer, String)
  - ✓new 키워드로 만든 객체 뿐만 아니라, 대부분이 클래스 기반
- ✓절차가 아닌 객체 위주로 생각
  - ✓객체 간의 상호작용으로 프로그램이 구성됨
- ✓객체 지향의 여러가지 특징을 기본으로 설계됨
  - ✓캡슐화, 상속, 다형성, 추상화

[참고] 자바 코드가 실행될 수 있는 이유

✓파일명이 Solution.java 라면

- ✓실행했을 때, Solution class 가 Solution.class 로 생성됨
- ✓해당 class 의 main 함수를 실행

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

- ✓static : 객체를 생성하지 않고도 사용할 수 있는 영역

✓static 이 없이 main 을 실행하려면, 먼저 객체부터 생성해야 함

- ✓반드시 main 은 static 이어야 한다.

객체 지향 프로그래밍을 위해 알아야 할 개념들

✓ 필수 개념

1. 상속과 다형성
2. 정보 은닉(캡슐화)
3. 인터페이스와 추상 클래스

✓ 이전 카페 주문 시스템을 기반으로 학습

```
class Order {  2 usages
    String name;  3 usages
    String drink;  3 usages

    Order(String name, String drink) {  1 usage
        this.name = name;
        this.drink = drink;
    }

    void printOrder() {  1 usage
        System.out.println(name + "의 주문: " + drink);
    }
}
```

상속의 필요성

✓ 이전까지는 음료 주문만 받았으나, 간식 주문도 추가해달라는 요청이 왔다.

✓ 가장 쉬운 방법

- ✓ "음료주문" 과 "간식주문" 을 별개로 관리

✓ 어떠한 문제점이 있을까 ?

- ✓ 같은 역할의 코드가 반복됨 (name, printOrder() 등)
- ✓ 나중에 기능이 바뀌면 ...?

```
class DrinkOrder {  2 usages
    String name;  2 usages
    String drink;  2 usages

    DrinkOrder(String name, String drink) {  1 usage
        this.name = name;
        this.drink = drink;
    }

    void printOrder() {  1 usage
        System.out.println(name + "의 주문: " + drink);
    }
}

class SnackOrder {  2 usages
    String name;  2 usages
    String snack;  2 usages

    SnackOrder(String name, String snack) {  1 usage
        this.name = name;
        this.snack = snack;
    }

    void printOrder() {  1 usage
        System.out.println(name + "의 주문: " + snack);
    }
}
```

공통된 역할은 따로 관리

✓음료주문과 간식주문은 "주문" 이라는 공통 역할이 존재

✓ 공통 역할 : name, printOrder()

```
class Order {  2 usages  2 inheritors
    String name;  4 usages

    Order(String name) {  2 usages
        this.name = name;
    }

    void printOrder() {  2 usages  2 overrides
        System.out.println(name + "의 주문");
    }
}
```

공통 기능의 재사용

✓상속을 통해 공통 기능 재사용

- ✓ extends : 상속을 선언할 때 사용하는 키워드
- ✓ super : 부모 클래스를 참조
  - ✓ super() : 부모 클래스의 생성자

```
class Order {  2 usages  2 inheritors
    String name;  4 usages

    Order(String name) {  2 usages
        this.name = name;
    }

    void printOrder() {  2 usages  2 overrides
        System.out.println(name + "의 주문");
    }
}
```

```
class DrinkOrder extends Order {  2 usages
    String drink;  1 usage

    DrinkOrder(String name, String drink) {
        super(name);  // 부모 생성자 호출
        this.drink = drink;
    }
}
```

```
class SnackOrder extends Order {  2 usages
    String snack;  1 usage

    SnackOrder(String name, String snack) {
        super(name);  // 부모 생성자 호출
        this.snack = snack;
    }
}
```

공통 기능의 재사용

✓ 상속하면 자식 클래스는 부모의 변수와 메서드까지 쓸 수 있다.

- ✓ DrinkOrder 는 Order 의 자식 클래스
- ✓ 부모의 name, printOrder() 를 그대로 상속받아 사용 가능
- ✓ 자식만의 필드(drink)는 직접 추가

```
class Order {  2 usages  2 inheritors
    String name;  4 usages

    Order(String name) {  2 usages
        this.name = name;
    }

    void printOrder() {  2 usages  2 overrides
        System.out.println(name + "의 주문");
    }
}
```

```
class DrinkOrder extends Order {  2 usages
    String drink;  1 usage

    DrinkOrder(String name, String drink) {
        super(name);  // 부모 생성자 호출
        this.drink = drink;
    }
}
```

두 클래스의 기능이 합쳐진 객체 생성

```
DrinkOrder drinkOrder1 = new DrinkOrder( name: "김선배",  drink: "아이스 아메리카노");

drinkOrder1.printOrder();                // 부모의 메서드 사용
System.out.println(drinkOrder1.name);    // 부모의 필드 접근
System.out.println(drinkOrder1.drink);   // 자식의 필드 접근
```

출력 결과

```
김선배의 주문
김선배
아이스 아메리카노
```

자식 클래스의 역할

✓공통 기능이 아닌 자식 클래스만의 변수와 기능은 따로 추가

✓ @override : 같은 이름의 메서드인데, 자식만의 기능을 따로 정의할 때 활용

```
class DrinkOrder extends Order {  2 usages
    String drink;  3 usages

    DrinkOrder(String name, String drink) {  1 usage
        super(name);  // 부모 생성자 호출
        this.drink = drink;
    }

    @Override  1 usage
    void printOrder() {
        System.out.println(name + "의 마실 것 주문 : " + drink);
    }
}
```

```
DrinkOrder drinkOrder1 = new DrinkOrder( name: "김선배",  drink: "아이스 아메리카노");

drinkOrder1.printOrder();                // 자식클래스에서 재정의 된 메서드 사용
System.out.println(drinkOrder1.name);    // 부모의 필드 접근
System.out.println(drinkOrder1.drink);   // 자식의 필드 접근
```

출력 결과

```
김선배의 마실 것 주문 : 아이스 아메리카노
김선배
아이스 아메리카노
```

상속 코드 완성

✓간식 주문 클래스 (SnackOrder) 도 완성합니다.

```java
class DrinkOrder extends Order {  2 usages
    String drink;  3 usages

    DrinkOrder(String name, String drink) {  1 usage
        super(name);  // 부모 생성자 호출
        this.drink = drink;
    }

    @Override  1 usage
    void printOrder() {
        System.out.println(name + "의 마실 것 주문 : " + drink);
    }
}
```

```java
class SnackOrder extends Order {  2 usages
    String snack;  2 usages

    SnackOrder(String name, String snack) {  1 usage
        super(name);  // 부모 생성자 호출
        this.snack = snack;
    }

    @Override  2 usages
    void printOrder() {
        System.out.println(name + "의 간식 주문 : " + snack);
    }
}
```

main 코드

```java
DrinkOrder drinkOrder1 = new DrinkOrder( name: "김선배",  drink: "아이스 아메리카노");
drinkOrder1.printOrder();                    // 자식클래스에서 재정의 된 메서드 사용

SnackOrder snackOrder1 = new SnackOrder( name: "김선배",  snack: "딸기 케이크");
snackOrder1.printOrder();
```

출력 결과

```
김선배의 마실 것 주문 : 아이스 아메리카노
김선배의 간식 주문 : 딸기 케이크
```

상속 정리

✓ 상속(Inheritance) 이란 ?

- ✓ 부모 클래스의 필드와 메서드를 자식 클래스가 물려받는 기능
- ✓ 코드 재사용성 향상 -> 중복 제거, 유지보수 용이

✓ 상속의 특징

- ✓ extends 키워드로 상속 선언
- ✓ 부모 객체는 super 를 활용하여 호출 가능
- ✓ 부모의 필드와 메서드를 자식에서 그대로 사용하거나 재정의(Override) 가능

✓ 상속이 필요한 이유

- ✓ 공통 기능을 부모에 두고 차이점만 구현

여러 개의 객체 관리

✓ 여러 명에게서 음료와 간식 주문을 받은 상황

- ✓ 김선배 : 아이스 아메리카노, 딸기 케이크
- ✓ 박후배 : 라떼
- ✓ 금선생 : 바닐라 라떼
- ✓ 최부장 : 초코쿠키

✓ 여러 개의 주문을 우선 객체로 생성

```
DrinkOrder drinkOrder1 = new DrinkOrder( name: "김선배",  drink: "아이스 아메리카노");
DrinkOrder drinkOrder2 = new DrinkOrder( name: "박후배",  drink: "라떼");
DrinkOrder drinkOrder3 = new DrinkOrder( name: "금선생",  drink: "바닐라 라떼");

SnackOrder snackOrder1 = new SnackOrder( name: "김선배",  snack: "딸기 케이크");
SnackOrder snackOrder2 = new SnackOrder( name: "최부장",  snack: "초코쿠키");
```

여러 개의 객체 관리

✓앞으로 주문이 더 추가될 수 있으니, ArrayList 로 관리

```
ArrayList<DrinkOrder> drinkOrders = new ArrayList<>();
ArrayList<SnackOrder> snackOrders = new ArrayList<>();
```

✓동일한 "주문" 이라는 개념이지만 따로 관리하는 형태

- ✓ 전체 메뉴 출력 시 **두 번 작업**해야 함
- ✓ 전체 주문 개수, 검색 등을 하려면 **두 번 작업**해야 함
- ✓ 식사 주문 등 더 많은 종류의 주문이 생긴다면, **유지보수가 어려워짐**

다형성이란 무엇인가?

✓ 다형성이란 ?

- ✓ 하나의 부모 타입으로 여러 형태의 자식 타입을 다룰 수 있는 성질

✓ 예시

```
Order o1 = new DrinkOrder( name: "김선배", drink: "아메리카노");
Order o2 = new SnackOrder( name: "김선배", snack: "딸기케이크");

o1.printOrder();    // DrinkOrder의 printOrder() 실행
o2.printOrder();    // SnackOrder의 printOrder() 실행
```

- ✓ o1, o2 는 Order 타입으로 생성
- ✓ 실제 데이터는 자식 클래스 기준 데이터
  - ✓ 따라서, 자식 클래스의 오버라이딩된 메서드가 실행됨

다형성과 업캐스팅

✓ 업캐스팅(Upcasting)

✓ 자식 객체를 부모 타입으로 형변환

```
Order o1 = new DrinkOrder( name: "김선배",  drink: "아메리카노");
Order o2 = new SnackOrder( name: "김선배",  snack: "딸기케이크");
```

✓ 업캐스팅의 특징 및 주의사항

특징 및 주의사항	설명
자동으로 변환됨	명시적인 형변환 필요 없음 (Order o1 = new DrinkOrder();)
실제 데이터는 자식 객체가 가진 데이터	오버라이딩된 메서드는 자식 방식으로 실행됨 (다형성) ex) o1.printOrder() // DrinkOrder의 printOrder() 실행
부모의 멤버만 접근 가능	오직 부모 클래스에 선언된 필드나 메서드만 사용할 수 있음 ex) System.out.println(o1.drink); // 오류! drink 는 자식의 고유 변수

다형성과 업캐스팅

✓ 업캐스팅 적용 코드

- ✓ Order 타입 리스트로 모두 관리 가능
- ✓ 실제 동작은 자식 방식

✓ 장점

- ✓ 전체 주문에 대한 작업을 할 수 있다.
  - ✓ 메뉴 출력, 메뉴 수 계산 등
- ✓ 나중에 다른 주문 방식이 생겨도 그대로 담을 수 있다.

```
// 업캐스팅을 활용하여 선언
Order o1 = new DrinkOrder( name: "김선배",  drink: "아이스 아메리카노");
Order o2 = new DrinkOrder( name: "박후배",  drink: "라떼");
Order o3 = new DrinkOrder( name: "금선생",  drink: "바닐라 라떼");
Order o4 = new SnackOrder( name: "김선배",  snack: "딸기 케이크");
Order o5 = new SnackOrder( name: "최부장",  snack: "초코쿠키");

// 전체 주문을 관리할 리스트
ArrayList<Order> orderList = new ArrayList<>();

// 주문 추가
orderList.add(o1);
orderList.add(o2);
orderList.add(o3);
orderList.add(o4);
orderList.add(o5);

// 각 주문 출력
for (Order order : orderList) {
    order.printOrder(); // 각각의 자식 printOrder()가 호출됨
}
```

다형성 정리

- ✓ 다형성 (Polymorphism) 이란?
  - ✓ 하나의 부모 타입으로 여러 자식 객체를 다룰 수 있는 특성
  - ✓ 선언은 부모 타입으로, 실제 동작은 자식 방식대로
- ✓ 업캐스팅 (Upcasting)
  - ✓ 자식 객체를 부모 타입으로 참조하는 것
  - ✓ 자동으로 형변환됨 (명시적 형변환 불필요)
  - ✓ 부모에 정의된 멤버만 사용 가능

실무에서의 상속과 다형성

✓실무에서는 상속은 조심스럽게 사용해야 함

이유	설명
강한 결합	부모 클래스가 바뀌면, 자식 클래스들이 모두 영향을 받음
확장에 불리함	완벽한 포함 관계 (is-a) 관계가 아니면 억지스러운 상속이 발생
단일 상속 제한	Java 는 하나만 상속할 수 있어 복잡한 계층 설계가 어려움

✓대안 : 인터페이스(interface)

- ✓ 다형성은 유지하면서 유연한 구조 설계 가능

정보 은닉이 필요한 이유

✓카페 주문 시스템 예제

✓ 김선배의 메뉴만 다루기 위해 아래 코드로 수정

```
DrinkOrder order1 = new DrinkOrder( name: "김선배",  drink: "아이스 아메리카노");
order1.printOrder();
```

정보 은닉이 필요한 이유

✓카페 주문 시스템 예제

- ✓ 누군가 갑자기 “김선배 메뉴를 이상한 음료로 변경” 이라고 말함
  - ✓ 김선배는 이를 모르는 상태
- ✓ 주문을 받고, 김선배 메뉴를 이상한 음료로 변경함
- ✓ 이 후 상황
  - ✓ 아메리카노를 마실 생각에 신난 김선배는

    갑자기 이상한 음료를 받음

정보 은닉이 필요한 이유

✓김선배 메뉴 변경

```
// 메뉴 변경
order1.drink = "이상한 음료";
order1.printOrder();  // 이상한 음료로 변경
```

✓문제점

- ✓ 비정상적인 값으로 수정해도 막을 방법이 없음
- ✓ 아무 클래스에서나 직접 접근 가능
- ✓ 여러 곳에서 수정할 경우 유지보수가 매우 어려워짐

정보 은닉 (캡슐화)

✓객체의 데이터를 외부로부터 보호하는 것이 핵심

✓두 가지 접근 방식

1. 외부에서 직접 보거나 수정하지 못하게 감춤
2. 꼭 필요한 경우에만 제한적으로 접근 허용
   - ✓수정만 허용 / 읽기만 허용 등
   - ✓초기화 및 수정에서는 유효성 검사를 넣음

외부란 무엇인가 ?

✓ 회사에서 문서를 어디까지 공유할 수 있는가 ?

Level 1. 나 혼자만 보는 문서

- ✓ 내 연봉표

Level 2. 팀 내부에서 공유하는 문서

- ✓ 팀 별 주간 업무 보고서

Level 3. 부서 간 협력 시 공유되는 문서

- ✓ 협업 기획서

Level 4. 외부에 공개되는 문서

- ✓ 제품 소개서

외부란 무엇인가 ?

✓ 회사에서 문서를 어디까지 공유할 수 있는가 ?

Level 1. 나 혼자만 보는 문서

✓ 내 연봉표 → ✓ 같은 클래스내에서만 가능

Level 2. 팀 내부에서 공유하는 문서

✓ 팀 별 주간 업무 보고서 → ✓ 같은 패키지까지 가능

Level 3. 부서 간 협력 시 공유되는 문서

✓ 협업 기획서 → ✓ 같은 패키지 +
다른 패키지의 자식 클래스 가능

Level 4. 외부에 공개되는 문서

✓ 제품 소개서 → ✓ 어디서든 접근 가능

자바

패키지란 ?

✓ 자바에서 파일(클래스)을 폴더처럼 정리하는 방법

- ✓ 규모가 큰 프로젝트에서는 기능별로 묶어놓는 것이 좋다

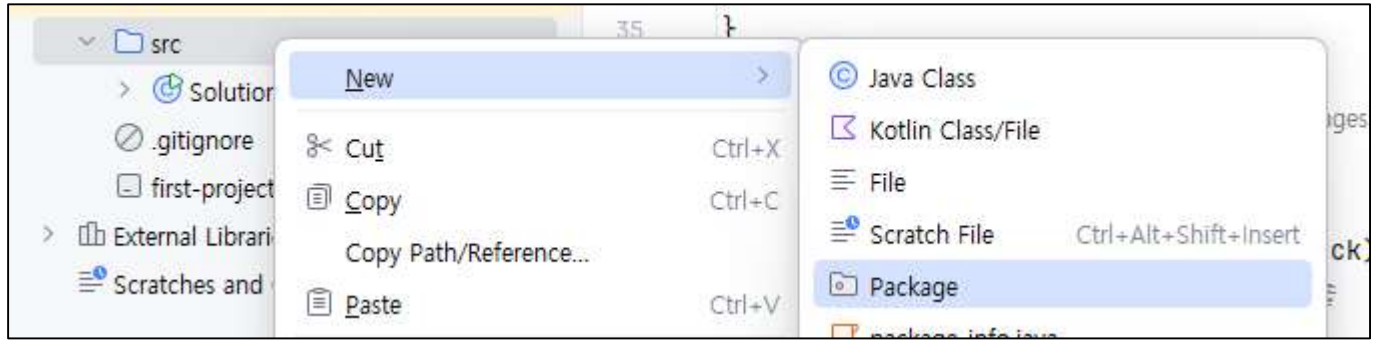
✓ 예시

- ✓ 자바의 패키지 = 회사의 부서
    - ✓ dev 패키지 = 개발팀, sales 패키지 = 영업팀
- ✓ 각 부서에는 고유한 업무 문서(클래스)가 있다.
    - ✓ 인사팀 : Report (버그 리포트 등)
    - ✓ 영업팀 : Report (매출 보고서 등)
    - ✓ 패키지가 다르면 클래스 이름이 동일해도 충돌 없다

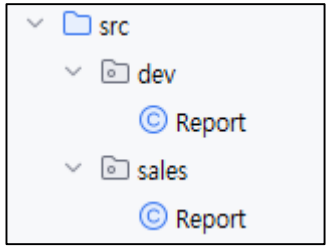
패키지란 ?

✓ 개발팀(dev) 패키지와 영업팀(sales) 패키지를 만든다

✓ src 폴더 우클릭 -> New -> Package



✓ 각 패키지에 Report class를 생성한다.




```
package dev;

public class Report {
}
```

```
package sales;

public class Report {
}
```

패키지를 활용하는 방법

- ✓ 개발팀의 Report 에 테스트용 코드가 있다고 가정
- ✓ 메인 코드에서 dev 패키지의 Report class 를 불러와 사용
  - ✓ **import** 를 활용하여 패키지를 불러옴

```java
package dev;

public class Report {  3 usages
    public void testCode() {  no usages
        System.out.println("테스트 동작");
    }
}
```

dev 패키지의 Report.java

```java
import dev.Report; // 패키지 경로를 import

public class Solution {
    public static void main(String[] args) {
        Report devReport = new Report();
        devReport.testCode();
    }
}
```

src/Solution.java

자바의 접근 계층

✓ 회사에서 문서를 어디까지 공유할 수 있는가 ?

이 범위를 접근 제어자를 통해 지정

Level 1. 나 혼자만 보는 문서

✓ 내 연봉표 → ✓ 같은 클래스내에서만 가능

Level 2. 팀 내부에서 공유하는 문서

✓ 팀 별 주간 업무 보고서 → ✓ 같은 패키지까지 가능

Level 3. 부서 간 협력 시 공유되는 문서

✓ 협업 기획서 → ✓ 다른 패키지 + 자식 클래스 가능

Level 4. 외부에 공개되는 문서

✓ 제품 소개서 → ✓ 어디서든 접근 가능

자바

정보 은닉 (캡슐화)

1. 외부에서 직접 보거나 수정하지 못하게 감춤

✓ 접근 제어자

✓ 클래스 내부 멤버(변수, 메서드 등)에 대한 **접근 범위를 제어**하는 키워드

✓ 정보를 숨길 지, 공개할 지를 결정하는 보안 등급

제어자	접근 가능 범위	예시
private	같은 클래스 내부에서만	개발자 혼자만 보는 디버깅 노트
(default)	같은 패키지(package) 내부에서만	개발팀 내부 보고서
protected	같은 패키지 + 다른 패키지의 **자식 클래스**	보고서 양식
public	**어디서든 누구나** 사용 가능	홈페이지 업로드용 공식 릴리즈 노트

접근 제어자 - private

✓ 개발팀의 에러 로그, 테스트 메모 같은 개발자 개인만 알아야 할 내용

✓ 영업팀이 보면 오히려 혼란만 줌

✓ **private : 같은 클래스 내에서만 조회 가능**

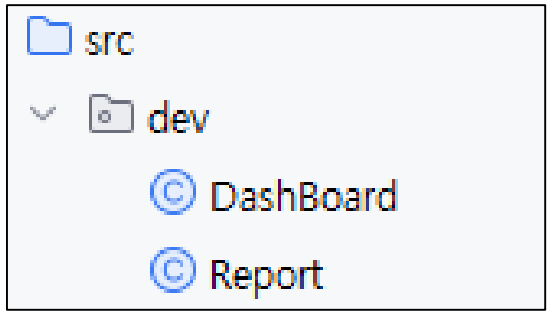
```
package dev;

public class Report {  3 usages
    private String memo = "5번째 Line 버그 ?";

    private void showMemo() {  no usages
        System.out.println(memo);
    }
}
```

접근 제어자 - default

✓업무 진행 보고서는 같은 팀 내에서는 모두 확인 가능

✓개발팀 내 업무 현황 게시판 (DashBoard class) 이 있다고 가정

✓ dev 패키지에 DashBoard class 생성



접근 제어자 - default

✓ Report.java 에서 활용 가능

- ✓ default 클래스 + default 필드로 생성 (접근 제어자를 적지 않음)
- ✓ **default : 같은 패키지 내에서만 활용 가능**

```java
package dev;

class DashBoard {  2 usages
    String taskStatus = "진행 중: 로그인 기능 개선 중";
}
```

DashBoard.java

```java
import dev.DashBoard;
```

main 에서는 불가능

(main 과 dev 는 다른 패키지)

```java
package dev;

public class Report {  3 usages
    private String memo = "5번째 Line 버그 ?";  1 usage

    private void showMemo() {  no usages
        System.out.println(memo);
    }

    private void showStatus() {  no usages
        DashBoard db = new DashBoard();
        System.out.println("개발팀 현황: " + db.taskStatus);
    }
}
```

Report.java

접근 제어자 - protected

✓ 개발팀의 보고서 양식을 영업팀이 가져가서 쓰고 싶다고 한다.

- ✓ 개발팀의 보고서 클래스 (BaseReport.java) 생성
- ✓ 영업팀에서 상속해서 활용
- ✓ **protected : 같은 패키지 + 다른 패키지의 자식 클래스에서 접근 가능하도록 허용**

```java
package dev;

public class BaseReport {  no usages
    protected String template = "기본 보고서 양식 (제목 / 내용 / 날짜)";

    protected void printTemplate() {  no usages
        System.out.println("템플릿: " + template);
    }
}
```

dev 패키지의 BaseReport.java

```java
package sales;

import dev.BaseReport;

public class Report extends BaseReport {  no usages
    public void customize() {  no usages
        // protected 필드, 메서드 접근 가능
        System.out.println("영업팀 보고서 생성 중...");
        printTemplate();  // OK
        template = "제목 / 내용 / 매출";  // OK
    }
}
```

sales 패키지 Report.java

접근 제어자 - protected

✓ protected 요약

접근 위치	protected 필드 접근 가능 여부	이유
같은 패키지 클래스	✓	protected는 default 포함
다른 패키지 일반 클래스	✗	상속 관계 아님
다른 패키지 자식 클래스	✓	protected는 자식 클래스에 열려 있음

접근 제어자 - protected

- ✓ 최근엔 protected 를 잘 쓰지 않는다
- ✓ 이유1. 캡슐화 약화
  - ✓ 상속이기 때문에 자식 클래스에게 너무 많은 내부 정보를 넘겨줘야 함
  - ✓ 상속 구조가 복잡해지면, 어디서 접근 했는 지 추적이 어려워짐
- ✓ 이유2. 구현에 강하게 의존
  - ✓ 부모 코드를 바꾸기 어렵고, 유지보수가 어려움
- ✓ 이유3. 다른 방식이 있음
  - ✓ private + 추후에 배울 제한적 접근 허용을 통해 우회 가능

접근 제어자 - public

✓ 영업팀이 회사 홈페이지에 소개 자료를 작성함

✓ 누구나 볼 수 있는 정보

✓ **public : 외부 클래스, 다른 패키지, 외부 프로젝트 등 모두 접근 가능**

```
public class Report extends BaseReport {  no usages
    public void customize() {  no usages
        // protected 필드, 메서드 접근 가능
        System.out.println("영업팀 보고서 생성 중...");
        printTemplate();  // OK
        template = "제목 / 내용 / 매출";  // OK
    }

    public void introduction() {  no usages
        System.out.println("자바를 활용한 프로그램입니다.");
    }
}
```

sales 패키지의 Report.java

이름이 동일함
import 제거 + 전체이름 사용

```
public class Solution {
    public static void main(String[] args) {
        dev.Report devReport = new dev.Report();
        sales.Report salesReport = new sales.Report();

        salesReport.introduction();
    }
}
```

public : 다른 패키지도 접근 가능

Solution.java

접근 제어자 정리

✓캡슐화의 핵심 : 필요한 만큼만 열어라

제어자	접근 범위	현실 비유	특징	사용 팁
private	같은 클래스 내부	**개발자 혼자만 보는 디버그 코드**	외부 클래스 접근 완전 차단	진짜 내부 정보 보호용. getter/setter로 간접 접근 허용
(default)	같은 패키지	**개발팀 내부 회의록** (부서 안 공유 문서)	패키지 안에서는 접근 가능, 외부 패키지는 ✕	실무에선 같은 기능 단위로 묶은 클래스끼리 쓸 때 적합
protected	같은 패키지 + 다른 패키지의 자식 클래스	**보고서 양식**	상속관계에서만 외부 패키지 접근 가능	최근에는 잘 안 씀 대신 getter/setter + 인터페이스 활용
public	모든 클래스 어디서나	**회사 홈페이지의 소개 페이지**	완전 개방형, 누구나 접근 가능	클래스, 인터페이스, API 메서드에만 신중하게 사용

정보 은닉 (캡슐화) – 제한적 접근 허용

2. 꼭 필요한 경우에만 제한적으로 접근 허용

✓ 직원 정보를 관리하는 Employee 클래스 작성

✓ 직원 연봉을 관리

```
class Employee {  no usages
    public int salary = 5000;
}
```

Solution.java

✓만약 개발자가 실수로 -1을 입력한다면 ?

✓ 급여가 마이너스가 된다. (치명적인 버그가 발생)

✓ 문제점 : 아무 코드에서나 값 수정이 가능 + 잘못된 값도 입력 가능

```
Employee e1 = new Employee();
e1.salary = -1; // 실수로 -1을 입력
```

정보 은닉 (캡슐화) – 제한적 접근 허용

✓ 객체 내부 정보를 무분별하게 접근하지 않도록 통제 수단이 필요함

- ✓ 조회 : **getter** / 수정 : **setter** 라고 부름
- ✓ 즉, 데이터 접근 전 한 단계의 과정을 더 거쳐서 접근하도록 구성

✓ getter 와 setter 의 특징

- ✓ 데이터는 private 으로 구성함 (직접 접근 불가능하도록)
- ✓ 꼭 필요한 경우에만 getter/setter 를 만든다
  - ✓ 반드시 있을 필요는 없다
- ✓ setter 에는 잘못된 값이 들어가지 않도록 **검증 로직을 반드시 포함**한다.

정보 은닉 (캡슐화) – 제한적 접근 허용

✓ 예시

✓ salary 에 대한 getter / setter 구성

```
class Employee {  2 usages
    // 필드는 private 으로 감추기
    private int salary = 5000;  2 usages

    // getter: 조회용
    public int getSalary() {  1 usage
        return salary;
    }

    // setter: 수정 시 유효성 검사 포함
    public void setSalary(int salary) {  2 usages
        if (salary >= 0) {
            this.salary = salary;
        } else {
            throw new IllegalArgumentException("급여는 0 이상이어야 합니다.");
        }
    }
}
```

```
Employee e1 = new Employee();
// e1.salary = -1; // 직접 수정 불가능

e1.setSalary(6000); // 수정 가능
System.out.println(e1.getSalary());  // 출력 시에는 getSalary() 호출

e1.setSalary(-1); // 에러 발생
```

캡슐화 정리

1. 캡슐화란 ?

✓ 객체 내부 데이터를 외부로부터 숨기고, 필요할 때만 통제된 방식으로 접근

2. 접근 제어자와 정보 보호

✓ private : 클래스 내부만

✓ (default) : 같은 패키지에서만

✓ protected : 같은 패키지 + 다른 패키지의 자식 클래스

✓ public : 어디에서나

3. 통제된 통로 구성 (getter / setter)

✓ getter : 읽기 전용 인터페이스

✓ setter : 수정 시 유효성 검사 포함 가능

캡슐화 Quiz (1/2)

- ✓ 아래 조건을 만족하는 Employee 클래스를 작성하시오
  - ✓ 필드 선언
    - ✓ name(String): 직원 이름 - 직접 접근 가능
    - ✓ department (String): 부서명 – 같은 패키지 내에서만 접근 가능
    - ✓ salary (int): 급여 – 외부에서 직접 접근 불가능 / 기본값: 5000
  - ✓ 메서드
    - ✓ name : 모든 곳에서 조회만 가능 / 수정 불가능
    - ✓ department : 모든 곳에서 조회 가능 / 수정 불가능
    - ✓ salary : 조회 불가능 / 0~10000 이하로만 수정 가능

캡슐화 Quiz (2/2)

✓ main 실행 코드

✓ 실행 결과는 작성된 주석과 동일하면 된다.

```
public class Solution {
    public static void main(String[] args) {
        Employee e = new Employee();
        e.name = "홍길동";          // public이므로 접근 가능
        System.out.println(e.name);

        System.out.println(e.getSalary()); // 5000 출력
        e.setSalary(8000);                 // 급여 수정
        System.out.println(e.getSalary()); // 8000 출력

        e.setSalary(15000); // 예외 발생

        // e.department = "영업팀";     // default 접근 → 같은 패키지면 가능
        System.out.println(e.getDepartment()); // getter는 public
        // e.setDepartment("개발팀");   // setter 없음 → 수정 불가

    }
}
```

캡슐화 Quiz 정답 코드

✓정답 코드

```
class Employee {  2 usages
    public String name;          // 누구나 접근 가능  2 usages
    String department;           // default 접근제어자 (같은 패키지만)  1 usage
    private int salary = 5000;   // 외부에서 직접 접근 불가  2 usages

    // 부서 getter (외부에서 읽기만 가능)
    public String getDepartment() {  1 usage
        return department;
    }

    // 급여 getter
    public int getSalary() {  2 usages
        return salary;
    }

    // 급여 setter
    public void setSalary(int salary) {  2 usages
        if (salary >= 0 && salary <= 10000) {
            this.salary = salary;
        } else {
            throw new IllegalArgumentException("급여는 0 이상 10000 이하만 가능합니다.");
        }
    }
}
```

Chapter 08

유연한 설계: 인터페이스의 활용

웹 개발자를 위한 자바 기초

인터페이스의 등장 이유

✓ 회사에 2개의 부서가 있다고 가정

- ✓ 개발팀 (DevTeam): 시스템 개발 보고
- ✓ 영업팀 (SalesTeam): 실적 보고
- ✓ 편의를 위해 모두 Solution.java 에 구현

```
class DevTeam {  no usages

}

class SalesTeam {  no usages

}

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

    }
}
```

✓ 대표이사(CEO)가 각 팀의 보고를 받는 상황

- ✓ 각 부서별로 "보고(report)" 라는 기능을 구성해야 한다.
- ✓ 대표이사(CEO) 는 **보고만 잘 받으면 팀이 어떤 방식으로 일 했는 지 관심 없음**

인터페이스의 등장 이유

✓보고 메서드 구성

✓내장된 문제점

- ✓ CEO 가 팀마다 다른 메서드 이름을 외워야 함
  - ✓ devReport(), salesReport() 등
- ✓ 보고 양식이 제각각이라 통일이 안됨
- ✓ 팀이 늘어나면 CEO 클래스 코드도 계속 바뀜
  - ✓ MarketingTeam, SupportTeam 등

```
class DevTeam {  2 usages
    public void devReport() {  1 usage
        System.out.println("개발팀: 코드 배포 보고");
    }

}

class SalesTeam {  2 usages
    public void salesReport() {  1 usage
        System.out.println("영업팀: 월간 실적 보고");
    }
}

public class Solution {
    public static void main(String[] args) {
        DevTeam dev = new DevTeam();
        SalesTeam sales = new SalesTeam();

        // CEO 가 보고 받는다고 가정
        dev.devReport();        // 메서드 이름 외워야 함
        sales.salesReport();    // 다른 메서드 이름 = 통일성 없음
    }
}
```

인터페이스로 문제 해결하기

✓개선 구조

- ✓ 모든 팀은 "보고(report)" 라는 공통 메서드만 구현하면 된다.
- ✓ CEO 는 각 팀이 어떤 방식으로 일 했는 지 모르고 report() 만 호출
- ✓ 새로운 팀이 추가되어도 CEO 코드는 수정 불필요

✓인터페이스

- ✓ **공통된 메서드의 "틀(구조)" 만 따로 정의**한다
  - ✓ 어떤 기능을 구현해야 하는 지 약속
- ✓ 각 클래스에서 implements 키워드를 통해 메서드를 포함시킬 수 있다.
- ✓ 인터페이스를 구현하는 클래스는 **반드시 그 안의 메서드를 오버라이드(구현)** 해야 한다.

인터페이스 구현

✓ 예제 구현

- ✓ interface 키워드를 통해 인터페이스(틀)를 구성
- ✓ implements : 해당 클래스가 인터페이스의 기능을 실제로 구현하겠다는 약속
- ✓ @Override : 인터페이스에서 정의한 메서드를 정확히 구현했는 지 검사

```
interface  Reportable {  2 usages  2 implementations
    void report();  // 구조만 정의  2 usages  2 implementations
}


class DevTeam implements Reportable {  2 usages
    @Override  2 usages
    public void report() {
        System.out.println("개발팀: 코드 배포 보고");
    }
}

class SalesTeam implements Reportable {  2 usages
    @Override  2 usages
    public void report() {
        System.out.println("영업팀: 월간 실적 보고");
    }
}
```

```
public class Solution {
    public static void main(String[] args) {
        DevTeam dev = new DevTeam();
        SalesTeam sales = new SalesTeam();

        // CEO 가 보고 받는다고 가정
        dev.report();   // report() 라는 메서드 이름만 알면 된다.
        sales.report(); // report() 라는 메서드 이름만 알면 된다.
    }
}
```

implements vs extends

✓ implements 는 상속과 비슷해 보인다.

✓ implements 와 extends 의 차이점

구분	extends	implements
사용 대상	클래스 ← 클래스	클래스 ← 인터페이스
역할	상속 (부모 클래스를 이어받음)	구현 (기능 약속을 따름)
상속 수	단일 상속만 가능	**다중 구현 가능**
부모의 메서드	구현된 내용까지 상속	**이름(틀)만 선언됨 (내용 X)**
강제성	override는 선택 (필요 시)	**override 반드시 해야 함**
키워드 사용 예	class A extends B	class A implements I

상속과 인터페이스

✓ 현재까지의 내용

- ✓ 상속 : 각 클래스에서 공통으로 필요한 데이터나 기능을 물려받음
- ✓ 인터페이스 : 각 클래스에서 공통으로 필요한 기능의 틀을 제공

✓ 두 역할이 모두 필요할 때는 어떻게 해야 할까 ?

- ✓ 상속만으로는 기능의 틀을 강제할 수 없다
- ✓ 인터페이스만으로는 공통된 데이터/기능을 제공할 수 없다

추상 클래스

✓추상 클래스 (Abstract Class)

✓ 여러 클래스가 **공통으로 가지는 변수와 메서드를 미리 정의**

✓ 각 클래스에서 **구체적인 동작(행동)은 직접 구현하도록 강제**

```
abstract class Team {  2 usages  2 inheritors
    String name;  2 usages

    Team(String name) {  2 usages
        this.name = name;
    }

    // 공통 메서드
    public void intro() {  1 usage
        System.out.println(name + "팀입니다.");
    }

    // 추상 메서드: 자식 클래스에서 반드시 구현해야 함
    abstract void report();  2 usages  2 implementations
}
```

```
class MarketingTeam extends Team {  no usages
    MarketingTeam(String name) {  no usages
        super(name);
    }

    @Override  no usages
    public void report() {
        System.out.println("마케팅팀: 내용 보고");
    }
}
```

- extends Team: Team 클래스에서 공통 기능을 상속
- super(name): 부모 생성자 호출 – 필수
- @Override: 추상 메서드 구현(재정의)

인터페이스와 추상 클래스 Quiz (1/2)

✓ 아래 요구사항에 맞는 결재 시스템을 구축한다.

- ✓ 회사에는 여러 팀이 존재한다고 가정
- ✓ 모든 팀의 공통 속성
  - ✓ 이름(name)
  - ✓ intro() 메서드 : name + “팀 입니다” 출력
- ✓ 모든 팀은 아래 메서드를 반드시 구현해야 함
  - ✓ report() : 각자 다르게 구현
  - ✓ approve() : “name + 팀: 결재 승인 완료” 출력
- ✓ SupportTeam 클래스를 정의하고 위 조건에 맞도록 구현
  - ✓ 뒷 페이지에 SupportTEAM 의 상세 요구사항이 있습니다.

인터페이스와 추상 클래스 Quiz (2/2)

✓ SupportTeam 클래스는 아래 조건에 맞게 작성

- ✓ 추상 클래스와 인터페이스를 모두 구현할 것
- ✓ 생성자로 팀 이름을 전달받음
- ✓ report() 메서드 : “고객지원팀: 문의 처리 결과 보고” 출력
- ✓ approve() 메서드 : “고객지원팀: 결재 승인 완료” 출력

✓ 예시 실행 코드

```
SupportTeam support = new SupportTeam( name: "고객지원");

support.intro();    // 고객지원팀입니다.
support.report();   // 고객지원팀: 문의 처리 결과 보고
support.approve();  // 고객지원팀: 결재 승인 완료
```

실행 결과

```
고객지원팀입니다.
고객지원팀: 문의 처리 결과 보고
고객지원팀: 결재 승인 완료
```

인터페이스와 추상 클래스 Quiz 정답 코드

✓정답 코드

✓ report() : 팀의 핵심 동작이기 때문에 추상 클래스에 정의

✓ approve() : 선택적으로 갖는 기능이기 때문에 인터페이스로 분리

```
// 클래스: 공통 기능 제공
class Team {  1 usage  1 inheritor
    String name;  2 usages

    Team(String name) {  1 usage
        this.name = name;
    }

    public void intro() {  1 usage
        System.out.println(name + "팀입니다.");
    }
}

// 인터페이스: 보고, 결재 기능
interface Approveable {  1 usage  1 implementation
    void report();  1 usage  1 implementation
    void approve();  1 usage  1 implementation
}
```

```
// SupportTeam 클래스: 추상 클래스 상속 + 인터페이스 구현
class SupportTeam extends Team implements Approveable {  2 usages
    SupportTeam(String name) {  1 usage
        super(name);
    }

    @Override  1 usage
    public void report() {
        System.out.println("고객지원팀: 문의 처리 결과 보고");
    }

    @Override  1 usage
    public void approve() {
        System.out.println("고객지원팀: 결재 승인 완료");
    }
}
```

Chapter 09

예외 처리

웹 개발자를 위한 자바 기초

개발 시 발생하는 오류를 다루는 방법

✓ 실제 서비스에서는 다양한 사고들이 발생한다.

- ✓ 문제는 항상 발생한다. (사용자들의 행동은 항상 다채롭다)
- ✓ 에러가 발생해도, 서비스는 절대 멈춰서는 안된다.

✓ 예시: 고객이 홈페이지를 통해 주문을 한다.

- ✓ 고객이 **숫자를 입력해야 할 곳에 한글을 입력**하면?
- ✓ 고객이 **회원가입 시 이메일을 빼먹고 전송**하면?
- ✓ 혹은 서버에서 **DB 연결이 일시적으로 끊기면?**

✓ 이런 상황들을 바로 **예외(Exception)** 이라고 함

예외가 발생하는 코드

✓ 예외는 개발자가 통제할 수 없는 일시적 오류

- ✓ 하지만, 이걸 잡지 않으면 프로그램은 그대로 멈춘다.

✓ 예시 코드

- ✓ 에러보다 밑에 있는 코드들은 실행되지 못한다
- ✓ 코드가 멈춰버린다.

```
String userInput = "abc";
int number = Integer.parseInt(userInput); // 사용자 입력 오류

System.out.println("에러보다 밑에 있는 코드");
```

예외 처리 (Exception Handling)

✓ try-catch

✓ 자바 예외 처리 문법의 핵심

✓ 구조

```
try {
    // 예외가 발생할 수 있는 코드
} catch (예외타입 변수명) {
    // 예외 발생 시 실행할 코드
} finally {
    // 예외 발생 여부와 상관없이 실행할 코드
}
```

✓ 예제 코드

```
try {
    String userInput = "abc";
    int number = Integer.parseInt(userInput); // 사용자 입력 오류
} catch (Exception e) {
    System.out.println("입력값이 잘못되었습니다.");
} finally {
    System.out.println("에러보다 밑에 있는 코드");
}
```

✓ 출력 결과

```
입력값이 잘못되었습니다.
에러보다 밑에 있는 코드
```

예외의 종류

✓자바의 예외는 크게 두 가지로 나뉜다

분류	설명	대표 클래스
Checked Exception	**에러를 반드시 처리**해야 함 (try-catch 또는 throws)	IOException, SQLException, FileNotFoundException 등
Unchecked Exception	**실행 중에 발생** 에러 처리를 강제하지 않음	NullPointerException, ArrayIndex OutOfBoundsException, NumberFormatException 등

Checked Exception

✓ 일반 예외 (Checked Exception)

- ✓ 예외 처리를 필수적으로 해야함
  - ✓ try-catch 또는 throws 를 반드시 써야 함
- ✓ 대부분 **입출력, 네트워크, 파일, DB** 등에서 발생

```
import java.io.*;

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

        try {
            FileReader fr = new FileReader( fileName: "data.txt"); // 파일 없으면 예외
        } catch (FileNotFoundException e) {
            System.out.println("파일을 찾을 수 없습니다.");
        }

    }
}
```

Checked Exception

✓대표적인 일반 예외

예외 클래스	설명	발생 상황 예
IOException	입출력 작업 중 일반적인 오류	파일 읽기/쓰기, 스트림 처리
FileNotFoundException	파일이 존재하지 않을 때	없는 파일을 열 때
ClassNotFoundException	클래스를 찾을 수 없을 때	없는 클래스 조회 시
SQLException	DB 작업 중 오류	잘못된 SQL문 실행
ParseException	날짜나 숫자 파싱 실패	"2024-13-01" 같은 날짜 처리 시
InterruptedException	쓰레드 작업 중 인터럽트 발생	Thread.sleep() 도중 강제 중단

Unchecked Exception

✓실행 예외 (Unchecked Exception)

- ✓ 예외 처리를 강제하지 않음
- ✓ 즉, try-catch 나 throws 를 꼭 쓰지 않아도 컴파일은 된다

```
String name = null;
System.out.println(name.length()); // NullPointerException 발생
```

✓특징

항목	내용
검사 시점	컴파일러가 예외 처리를 강제하지 않음
주 발생 위치	**프로그래머 실수**나 **로직 오류**
주 예외 클래스	**RuntimeException**의 하위 클래스
예외 처리	try-catch는 **선택 사항** (안 해도 컴파일 됨)

Unchecked Exception

✓대표적인 실행 예외

예외	설명	발생 상황 예
NullPointerException	null 값을 참조하려고 할 때	String s = null; s.length();
ArrayIndexOutOfBoundsException	배열 인덱스 범위를 벗어날 때	int[] arr = {1,2}; arr[2];
ArithmeticException	0으로 나눌 때	int x = 10 / 0;
NumberFormatException	문자열을 숫자로 바꾸려다 실패할 때	Integer.parseInt("abc");
IllegalArgumentException	잘못된 인자가 전달될 때	Thread.sleep(-1000); 또는 setSalary(-1);

throws 와 예외 전가

✓ 그럼 모든 상황에 try-catch 를 넣으면 될까 ?

✓ 아래 상황을 보자

- ✓ 개발자 A, 팀장 B, 본부장 C가 있다.
- ✓ 개발자 A: "파일을 읽는 기능 개발" 을 맡음
- ✓ 팀장 B: A 의 기능을 이용해서 통계 계산
- ✓ 본부장 C: B가 만든 통계를 보고서에 붙임

- ✓ 각 기능에 예외가 발생할 가능성이 있다.
  - ✓ 각 기능에 모두 try-catch 를 넣는다.

각 기능에 모두 예외처리

✓코드

```java
// 개발자 A: 파일 읽는 기능
static void readFile() {  1 usage
    try {
        FileReader fr = new FileReader( fileName: "data.txt");
        System.out.println("파일 읽기 성공");
    } catch (FileNotFoundException e) {
        System.out.println("readFile 실패: 파일 없음");
    }
}

// 팀장 B: 통계 계산
static void calculate() {  1 usage
    try {
        readFile();
        System.out.println("통계 계산 완료");
    } catch (Exception e) {
        System.out.println("calculate 실패");
    }
}

// 본부장 C: 보고서 작성
static void report() {  1 usage
    try {
        calculate();
        System.out.println("보고서 작성 완료");
    } catch (Exception e) {
        System.out.println("report 실패");
    }
}
```

```java
public static void main(String[] args) {
    report(); // 예외는 이미 내부에서 다 처리돼버림
}
```

실행 결과

```
readFile 실패: 파일 없음
통계 계산 완료
보고서 작성 완료
```

✓ 문제점

- ✓ 예외가 위로 전달되지 않음
- ✓ main() 은 문제가 있었는 지 모름
- ✓ 예외 처리 로직이 중복됨 (관리 어려움)

throws

- ✓ 모든 함수에서 try-catch 를 쓰면
  - ✓ 예외가 main (위)까지 전달되지 않음
  - ✓ 위에선 문제 유무를 알 수 없음
- ✓ 해결 방법
  - ✓ 각 함수에서 예외를 처리하고 않고 **“위로 전달”**
  - ✓ 최종적으로 최상단 main() 에서 한 번에 처리

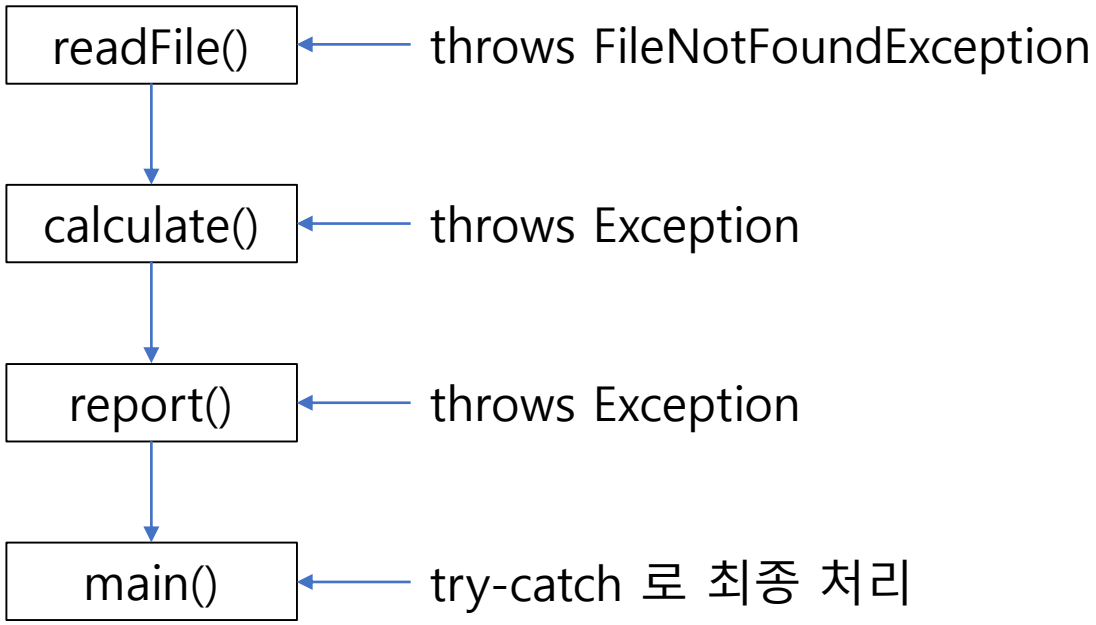
- ✓ 이 때 사용하는 키워드가 바로 **“throws”**

throws 와 예외 전가

✓ throws 란?

✓ 예외 처리를 자신이 하지 않고,

함수를 호출한 쪽에 전가하는 방식




```java
public class Solution {

    // 개발자 A: 파일 읽는 기능
    static void readFile() throws FileNotFoundException { 1 usage
        FileReader fr = new FileReader( fileName: "data.txt");
        System.out.println("파일 읽기 성공");
    }

    // 팀장 B: 통계 계산 기능
    static void calculate() throws Exception { 1 usage
        readFile();  // readFile이 던진 예외를 그대로 전가
        System.out.println("통계 계산 완료");
    }

    // 본부장 C: 보고서 작성 기능
    static void report() throws Exception { 1 usage
        calculate();  // calculate가 던진 예외를 그대로 전가
        System.out.println("보고서 작성 완료");
    }

    // main: 최종적으로 try-catch로 예외 처리
    public static void main(String[] args) {
        try {
            report();  // report가 던진 예외를 여기서 처리
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

언제 try-catch, 언제 throws 를 사용할까

✓ try-catch

- ✓ 예외를 현재 메서드 내에서 처리할 수 있을 때
  - ✓ ex) 사용자에게 에러 메시지만 출력하면 되는 경우
- ✓ 로그를 남기고 정상 흐름을 계속 이어가야 할 때
- ✓ 예외 복구 바로 가능한 경우
  - ✓ ex) 파일이 없으면 새로 만들고 진행

✓ throws

- ✓ 현재 메서드가 아닌, 상위 메서드가 책임질 때
- ✓ 계층적으로 예외를 모아서 한 곳에서 처리하려는 구조
- ✓ 핵심 로직에 집중하고, 예외 처리를 나중에 일괄적으로 하려는 경우

사용자 정의 예외 만들기

- ✓ 자바의 기본 예외로는 의미를 명확하게 표현하기 어려운 경우
- ✓ 프로젝트에 특화된 예외 상황을 명시적으로 표현
  - ✓ 협업 시 의도를 더 분명히 전달할 수 있음

- ✓ 예시 상황
  - ✓ 5명의 유저가 있는 상태
  - ✓ 없는 유저 조회 시 직접 정의한 UserNotFoundException 발생

사용자 정의 예외 만들기

✓ 예외 직접 정의하기

✓ Exception 객체를 상속받아 구현한다.

```java
// 1. 사용자 정의 예외 클래스
class UserNotFoundException extends Exception {  3 usages
    public UserNotFoundException(String message) {  1 usage
        super(message);
    }
}
```

사용자 정의 예외 클래스

```java
// 2. UserService: 유저 관리 기능
class UserService {  2 usages
    private List<String> users;  7 usages

    public UserService() {  1 usage
        users = new ArrayList<>();
        // 유저 5명 등록
        users.add("alice");
        users.add("bob");
        users.add("charlie");
        users.add("david");
        users.add("emma");
    }

    // 유저 조회
    public void findUser(String name) throws UserNotFoundException {  2 usages
        if (!users.contains(name)) {
            throw new UserNotFoundException(name + " 유저는 존재하지 않습니다.");
        }
        System.out.println(name + " 유저를 찾았습니다.");
    }
}
```

유저 관리 기능

사용자 정의 예외 만들기

✓활용하기

✓ "없는 유저 검색", "중복 유저" 등의 에러를 만들어서 사용할 수 있다.

```java
// 3. main(): 예외 처리
public class Solution {
    public static void main(String[] args) {
        UserService userService = new UserService();

        try {
            userService.findUser( name: "alice");      // 존재
            userService.findUser( name: "zoe");        // 존재하지 않음
        } catch (UserNotFoundException e) {
            System.out.println("에러 발생: " + e.getMessage());
        }

        System.out.println("프로그램 종료");
    }
}
```

main 코드

```
alice 유저를 찾았습니다.
에러 발생: zoe 유저는 존재하지 않습니다.
프로그램 종료
```

실행 결과

감사합니다.