# Java Programming II

# Lab1

514770-1
Fall 2025
9/10/2025

Kyoung Shin Park
Computer Engineering
Dankook University

# DRY (Don’t Repeat Yourself) Principle

- In the book “The Pragmatic Programmer”, DRY is defined as “Every piece of **knowledge** must have a single, unambiguous, authoritative representation within a system."
  - Knowledge – a precise functionality or an algorithm
- Violations of DRY
  - WET, "We enjoy typing," or “Waste everyone’s time”.
- How to Achieve DRY
  - To avoid violating the DRY principle, divide your system into pieces. Divide your code and logic into **smaller reusable units** and use that code by calling it where you want.
- DRY Benefits
  - Less code is good: It saves time and effort, is easy to maintain, and also reduces the chances of bugs.

# KISS (Keep It Simple Stupid) Principle

- "Keep It Simple Stupid", "Keep It Short and Simple"
- The KISS principle is descriptive to **keep the code simple and clear**, making it **easy to understand**.
- Violations of KISS
  - "Why they have written these unnecessary lines and conditions when we could do the same thing in just 2-3 lines?"
- How to Achieve KISS
  - To avoid violating the KISS principle, try to **write simple code**. Whenever you find lengthy code, divide that into multiple methods — **refactor**.
- KISS Benefits
  - If the code is written simply, then there will not be any difficulty in understanding that code, and also will be easy to modify.

# YAGNI (You Aren't Gonna Need It) Principle

- YAGNI says "don't implement something until it is necessary." YAGNI tells us to cut off any unnecessary part while KISS advises to make the rest as simple as possible.
- Violations of YAGNI
  - "over engineering" - a feature for every possible case, functions with a lot of input parameters, multiple if-else branches, rich and detailed interfaces, all those could be a smell of over engineering.
- How to Achieve YAGNI
  - Always **implement things when you actually need them**, never when you just foresee that you need them.
- YAGNI Benefits
  - Software developers don't have enough information to make the call on extra features, the time spent could be used elsewhere more productively. Extra features mean extra development time, testing time, documentation time, code review time.

# SOLID Principle

- **Single Responsibility Principle**
  - "A class should have one, and only one, reason to change."
- **Open/Closed Principle**
  - "Software entities (e.g. classes, modules, functions, etc) should be open for extension, but closed for modification."
- **Liskov Substitution Principle**
  - "Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program."
- **Interface Segregation Principle**
  - "Clients should not be forced to depend upon interfaces that they do not use." Reduce fat interfaces into multiple smaller and more specific client specific interfaces.
- **Dependency Inversion Principle**
  - One should depend on abstractions (interfaces and abstract classes) instead of concrete implementations (classes).

# Lab1

- 제공하는 코드를 SOLID 원칙에 따라 개선하라. 실행결과:
  - **== 초기 ==**
  - **a: apple(1) avocado(1)**
  - **b: banana(2)**
  - **c: carrot(3)**
  - **== banana 1회 제거 후 ==**
  - **a: apple(1) avocado(1)**
  - **b: banana(1)**
  - **c: carrot(3)**
  - **== banana 2회 제거 후 ==**
  - **a: apple(1) avocado(1)**
  - **c: carrot(3)**
  - **== date 추가 후 ==**
  - **a: apple(1) avocado(1)**
  - **c: carrot(3)**
  - **d: date(1)**

# Submit to e-learning

- **Google 검색 또는 생성형 AI (Chat GPT)에서 제공하는 코드를 그대로 사용하지 말고, 참조한 사이트 또는 AI 내용을 넣어주고, 본인이 생각해서 수정한 부분을 명시한다.**
- Lab1 과제에 yourcode(e.g. 클래스, 함수, 등등)를 추가 (yourcode 없을시 10점에서 -1점 감점)
- **Java25-2-HW1-YourID-YourName.zip** 과제(보고서에는 코드 설명을 반드시 구체적으로 포함)를 e러닝에 제출 **(due by 9/16)**.