Java Programming II
Lab1

514770-1
Fall 2024
9/11/2024

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

- Given a **LibraryItem and Library** program that reports a list of library items, **rewrite** this program based on SOLID principles.
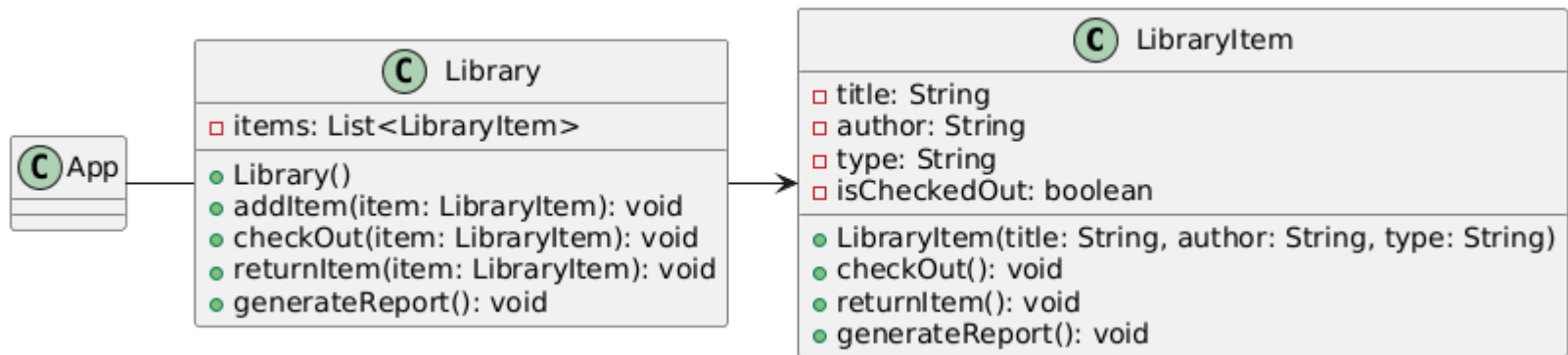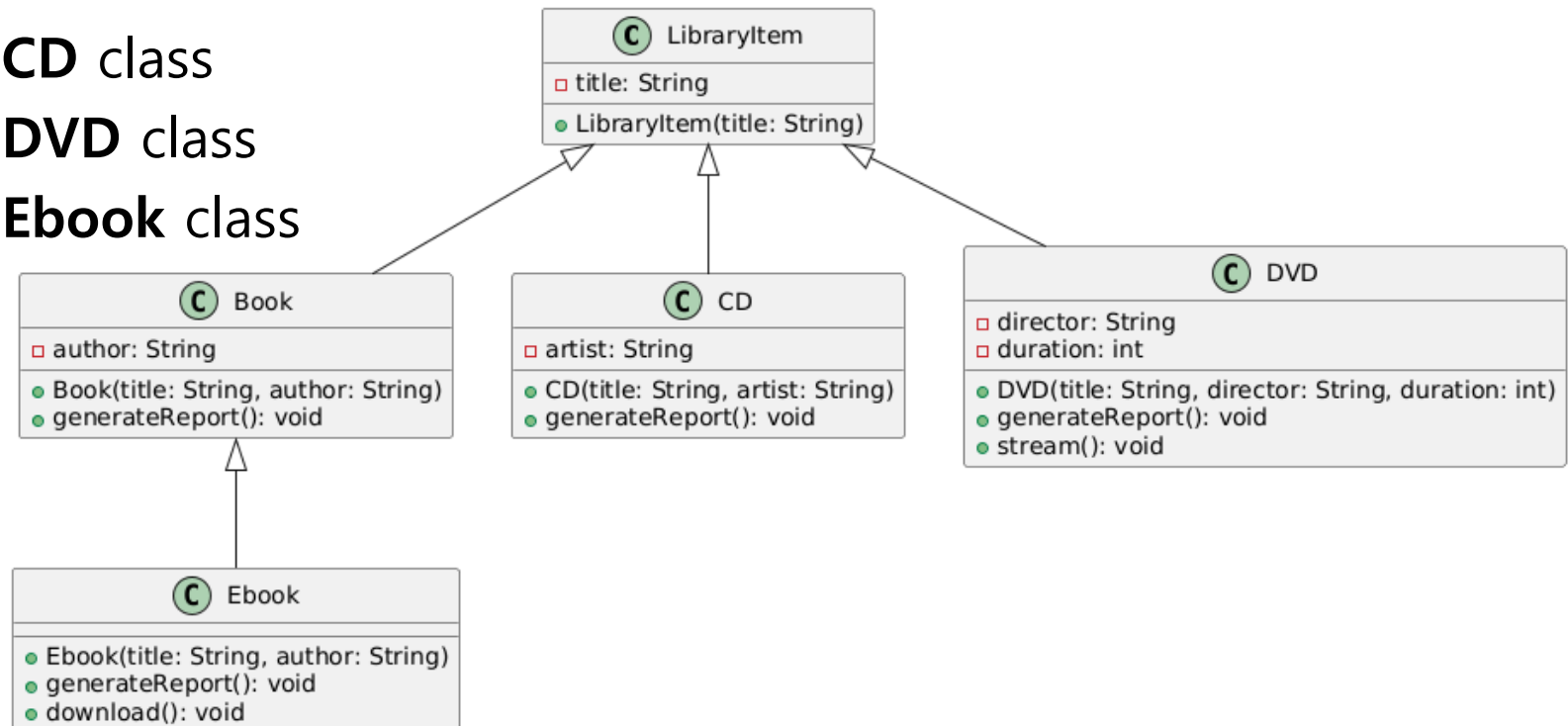
Lab1

- **LibraryItem** class will be split into multiple classes, each with **SRP**
  - **LibraryItem** class handles item basic data
  - **CheckoutManager** class manages the checkout status of LibraryItem
 - **ReportGenerator** interface handles the report generation for LibraryItem
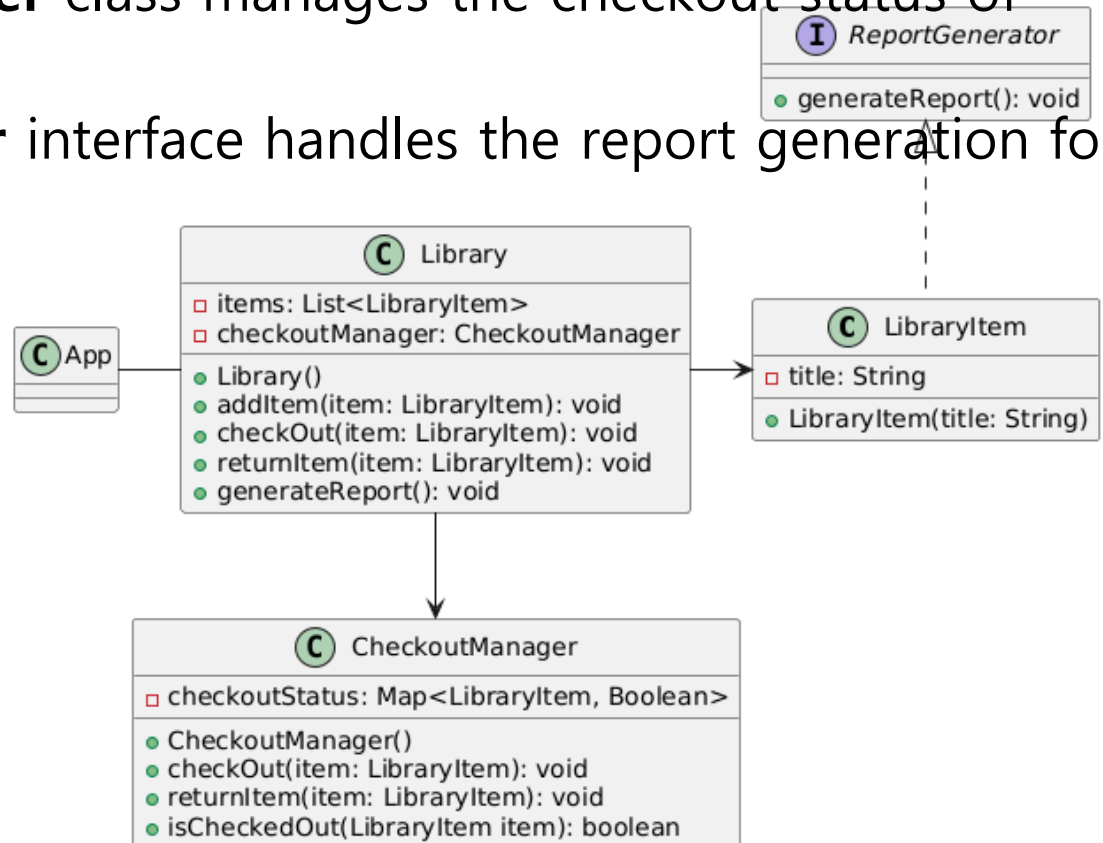
Lab1

□ **LibraryItem** class will be refactored into a base class, with specific subclasses for each media type (Book, CD, DVD, Ebook), based on **OCP**, allowing the easy addition of new types of media without altering existing code.

- **Book** class
- **CD** class
- **DVD** class
- **Ebook** class

LibraryItem

C LibraryItem
□ title: String
● LibraryItem(title: String)

Book

C Book
□ author: String
● Book(title: String, author: String) ● generateReport(): void

CD

C CD
□ artist: String
● CD(title: String, artist: String) ● generateReport(): void

DVD

C DVD
□ director: String □ duration: int
● DVD(title: String, director: String, duration: int) ● generateReport(): void ● stream(): void

Ebook

C Ebook
● Ebook(title: String, author: String) ● generateReport(): void ● download(): void

Lab1

- By applying **LSP**, each subclass of LibraryItem (e.g., Book, CD, DVD, Ebook) will be designed so that it can be substituted for the LibraryItem base class without altering the behavior of the program.
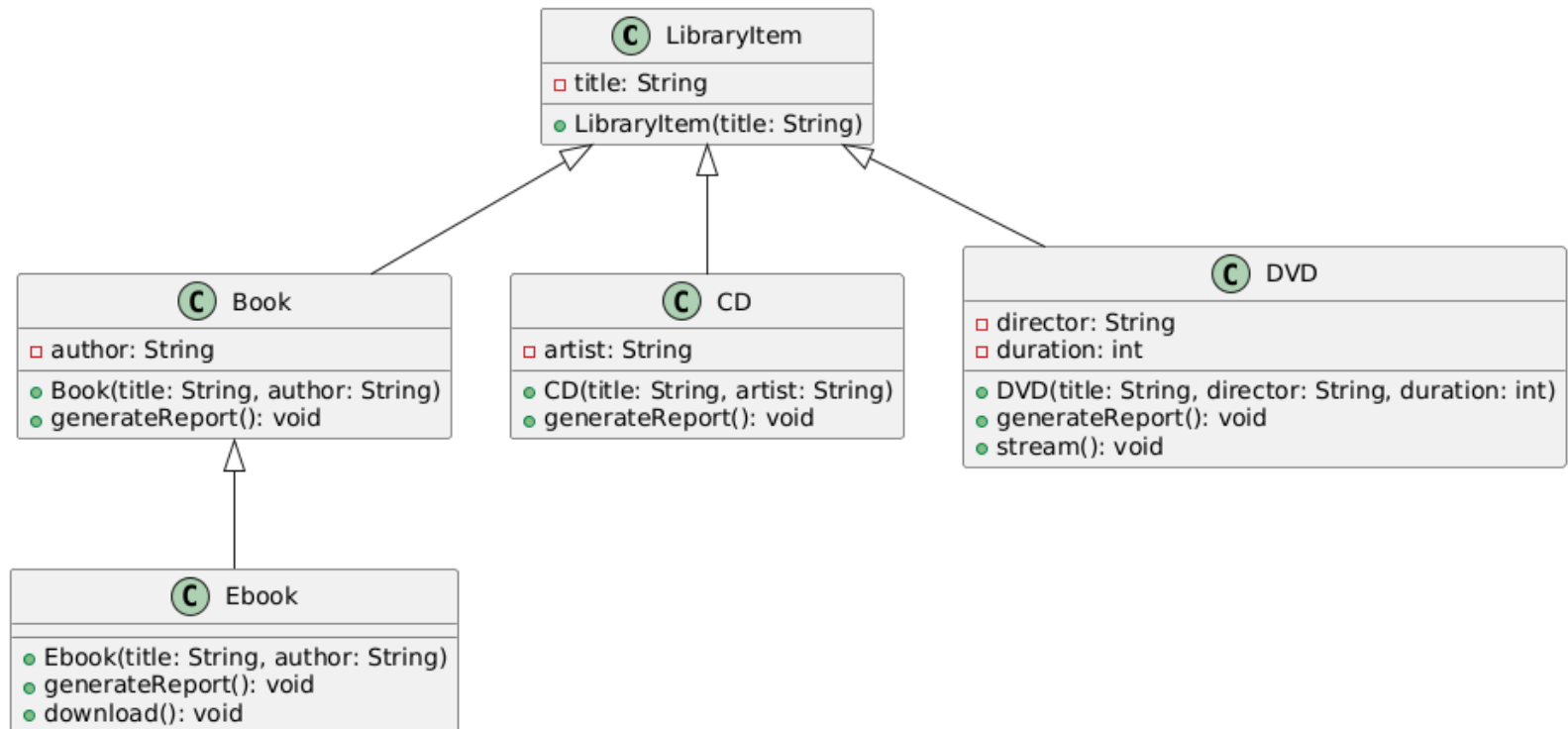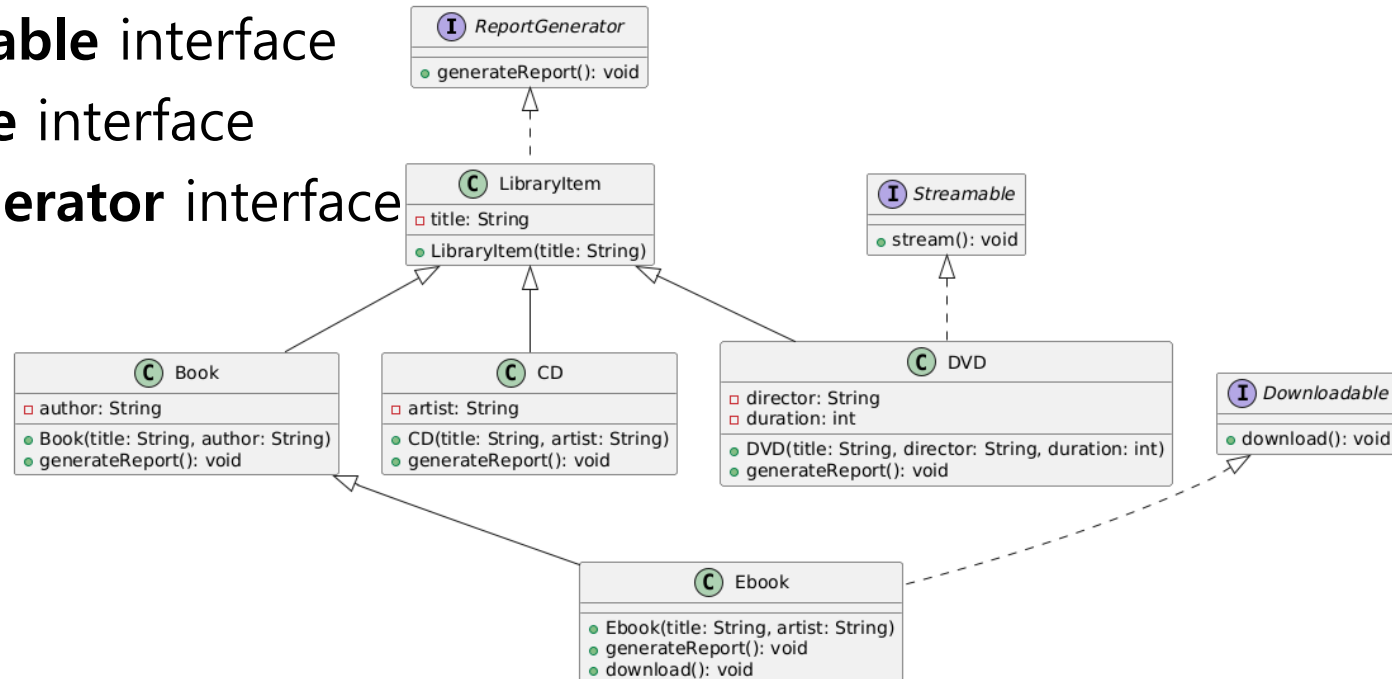
Lab1

- It will be refactored to define separate interfaces for different types of operations (**ISP**), ensuring that classes only implement the methods they need, in order to provide focused interfaces that prevent classes from implementing unnecessary methods.
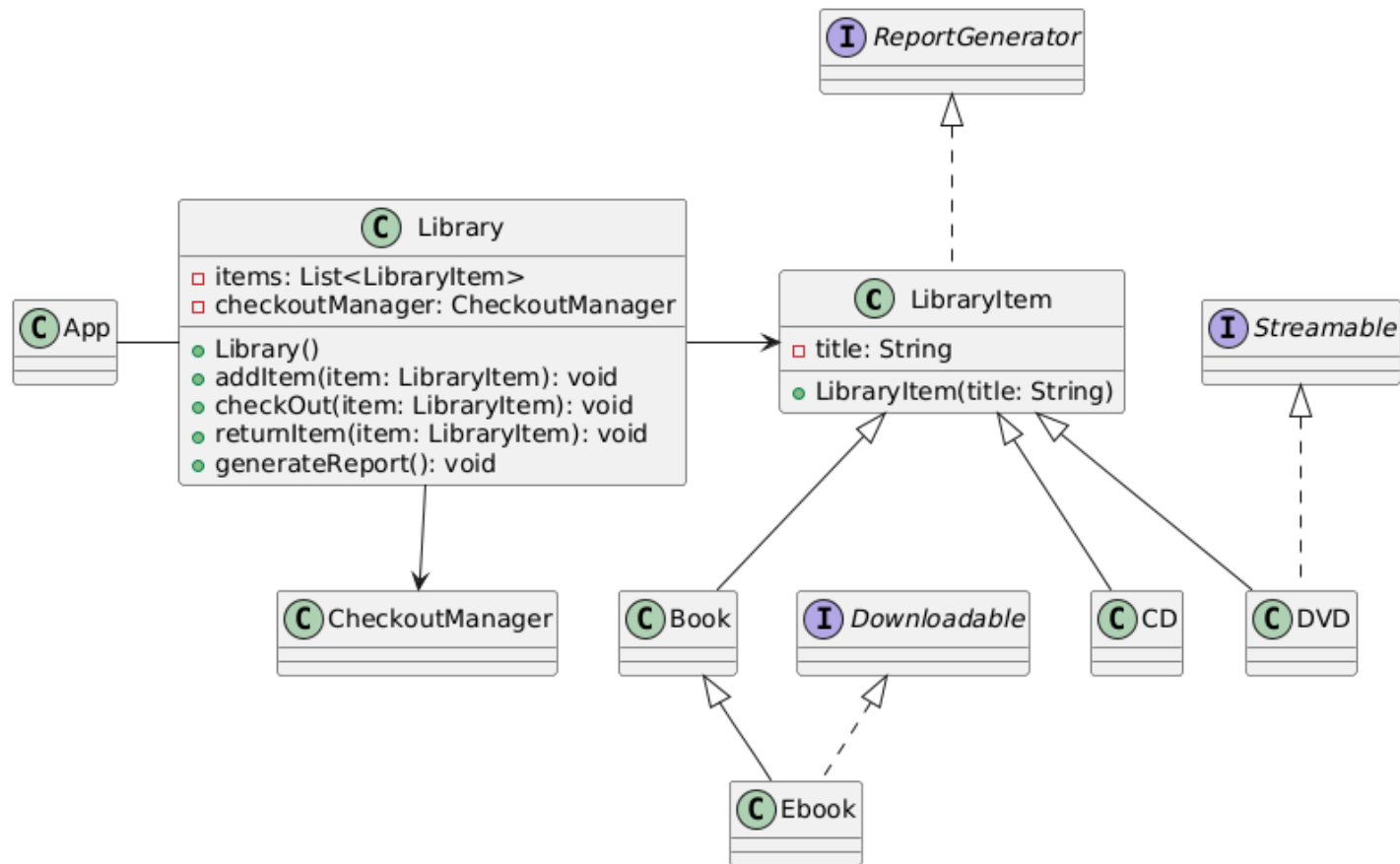
  - **Downloadable** interface
  - **Streamable** interface
  - **ReportGenerator** interface

Lab1

- By applying **DIP**, **Library** class will no longer depend directly on the concrete class. Instead, it will depend on an abstract interface or base class.

Submit to e-learning

- Add your code (e.g., one example from SOLID principles) in the Lab1 assignment. (yourcode 없을 시 -1)
- Submit the Lab1 assignment (including the report) to e-learning **(due by 9/18)**.