Object Oriented Analysis and Design with Java

UE19CS353

Prof.J.Ruby Dinakar
Department of Computer Science and Engineering

UE19CS353: Object Oriented Analysis and Design with Java

OO Design Principles and Sample Implementation of Patterns in Java

Prof. J.Ruby Dinakar

Department of Computer Science and Engineering

UE19CS353: Object Oriented Analysis and Design with Java

OO Design Principles and Sample Implementation of Patterns in Java

Dependency Inversion Principle (DIP)

Prof. J.Ruby Dinakar

Department of Computer Science and Engineering

Dependency Inversion Principle (DP)

Program to the interface, not the implementation

- High level modules should not depend upon low level modules, both should depend upon abstractions.
- Abstractions should not depend upon details, details should depend upon abstractions.
- The goal of the Dependency Inversion Principle is to avoid tightly coupled code, as it easily breaks the application.
- decouple high-level and low-level classes.
- As a solution, create an abstract layer for low-level classes, so that high-level classes can depend on abstraction rather than concrete implementations.
- According to Robert C Martin Dependency Inversion Principle is a specific combination of the Open-Closed and Liskov Substitution Principles.

Example code violates DIP

Suppose a book store asked us to build a new feature that enables customers to put their favorite books on a shelf.
To implement the new functionality, we create a lower-level Book class and a higher-level Shelf class. The Book class will allow users to see reviews and read a sample of each book they store on their shelves. The Shelf class will let them add a book to their shelf and customize the shelf.

```
class Book {
      void seeReviews() {...}
      void readSample() {...}
}

class Shelf {
      Book book;
      void addBook(Book book) {...}
      void customizeShelf() {...}
}
```

Example code violates DIP

At the high-level Shelf class depends on the low-level Book, the above code violates the Dependency Inversion Principle.
This becomes clear when the store asks us to enable customers to add DVDs to their shelves, too.
To fulfill the demand, we create a new DVD class

```
class DVD {

        void seeReviews() {...}
        void watchSample() {...}

}
```

Now, we should modify the Shelf class so that it can accept DVDs, too.
However, this would clearly break the Open-Closed Principle.
The solution is to create an abstraction layer for the lower-level classes (Book and DVD).
Introduce the Product interface and both classes will implement.

Solution

```
public interface Product {
    void seeReviews();
    void getSample();
}
class Book implements Product {
    @Override
    public void seeReviews() {...}
    @Override
    public void getSample() {...}
}
```

Solution

```
class DVD implements Product {

    @Override
    public void seeReviews() {...}

    @Override
    public void getSample() {...}

}
```

Solution

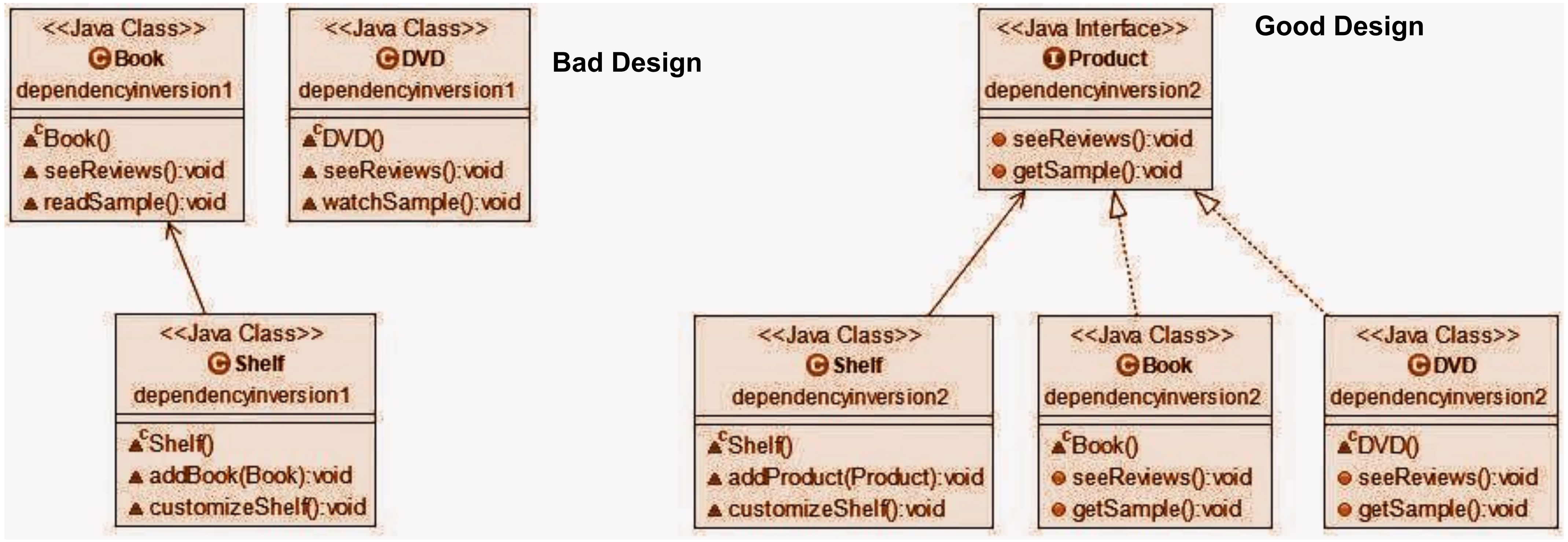
Now, Shelf can reference the Product interface instead of its implementations (Book and DVD).
The refactored code also allows us to later introduce new product types (for instance, Magazine) that customers can put on their shelves, too.

```
class Shelf {
    Product product;

    void addProduct(Product product) {...}

    void customizeShelf() {...}
}
```

The above code also follows the Liskov Substitution Principle, as the Product type can be substituted with both of its subtypes (Book and DVD) without breaking the program.
At the same time, we have also implemented the Dependency Inversion Principle, as in the refactored code, high-level classes don't depend on low-level classes, either.

UML Class Diagram

As you can see on the left of the UML graph below, the high-level Shelf class depends on the low-level Book before the refactoring. Without applying the Dependency Inversion Principle, we should make it depend on the low-level DVD class, too. However, after the refactoring, both the high-level and low-level classes depend on the abstract Product interface (Shelf refers to it, while Book and DVD implement it).



THANK YOU

J.Ruby Dinakar

Department of Computer Science and Engineering

rubydinakar@pes.edu