DALHOUSIE UNIVERSITY

CSCI 5308

Advance Topics in Software Development Concepts

Assignment 2

Name : Sagarkumar Pankajbhai Vaghasia

CSID : vaghasia

Banner ID : #B00878629

GitLab Link : https://git.cs.dal.ca/courses/2022-summer/csci-5308/vaghasia

1) Dependency Inversion principle

Bad Package:

- I have implemented four classes in the bad package : Student, CourseFee, CourseOffer and Main.
- The Student class includes two methods : getInput() and isStudentValid(). These two methods take input from student as name and password and validates it for having null or not.
- The CourseFee has one method coursesTotalFee(CourseOffer courseOffer). This method enables student to select courses and displays total fees payable by the student.
- The CourseOffer applies the offer on the course and passes to the coursesTotalFee().
- The Main class only creates objects of Student, CourseFee and CourseOffer, it does not have any methods.

- Dependency Inversion principle states that high level modules should not depend on the low-level modules; both should depend on abstractions.
  In the current scenario, CourseFee class is dependent on CourseOffer class for computing fees after applying offer. So, higher-level module (CourseFee class)depends on the concrete class(CourseOffer), and it is coupled with the concrete class. Thus, it violates dependency inversion principle.

Good Package:

- I created ICourseFee interface to remove the coupling from Main class and CourseFee class.
- I have created ICourseOffer interface. Previously, CourseOffer was coupled with CourseFee but now CourseOffer class implements ICourseOffer and passes the ICourseOffer’s object to the CourseFee class. Therefore, I removed coupling between CourseOffer and CourseFee classes.

- No changes are made in the Student class as it is not depending on any other class.
- In the Main class previously, I passed the objects of CourseFee and CourseOffer but now I am passing the objects of interfaces ICourseOffer and ICourseFees.
- By performing the above steps I fixed dependency inversion principle.

2) Single Responsibility Principle

Bad Package:

- I have implemented two classes in the bad package : Main and Student.
- The Main class here only creates the object of Student class and calls methods of Student class.
- The Student class has three variables : name, age and level. This class contains three methods : calculateFees(), saveStudentDetails() and printStudentReport().
- The calculateFees() method returns the fees according to the level of study i.e., undergrad, grad or PhD.
- The saveStudentDetails() method either inserts the student details in the database or updates the student details in the database.
- The printStudentReport() method prints the report of the student in various format.

- Single Responsibility principle states that every class in a program should have one responsibility in a program. Every class should have only one reason to change. If the management decides to change the way of fees calculation, then the whole student class has to be changed. If the management wants some changes in printing report type or they are going to apply another database management system, then also the Student class needs to be change. In the current scenario, Student class have 3 reasons to change. Thus, Student class violates single responsibility principle.

Good Package:

- I have created four classes : Student, StudentFeesCalculate, StudentReport and Main.
- Previously, the Student class had 3 reasons to change but now this class has only function saveStudentDetails() which performs database operations.
- I have created StudentFeesCalulcate class which is having single responsibility of calculating fees based on level of study.

- The StudentReport class is now separated from Student Class. This class is only responsible for printing student reports.
- The Main class in the good package is only creating the objects and calling methods. Previously, as there was only Student class, so the object of student class was created from main and all the methods are called by using that object. But now as there are different classes for different responsibilities, so objects of StudentFeesCalculate and StudentReport is created, and methods related to these classes are called from Main.
- Every class in the good package have one responsibility to change. Thus, by doing this way I fixed single responsibility principle.

3) Interface Segregation Principle

Bad Package:

- I have created an interface ICuisine having two methods bakeDish() and prepareDrink(). I have also created three classes : Mocktail, Pizza and Main.
- The Main class do not have any methods, it only creates the object of an interface for Pizza and Mocktail classes. It is only responsible for calling methods for both the classes.
- The Mocktail class implements ICuisine interface. Here, this class must implement both the methods of an interface. The Mocktail class only uses prepareDrink() method but there is also a do-nothing method bakeDish() which is not utilized by Mocktail class.
- The Pizza class also implements ICuisine interface. This class also have to implement both the methods of an interface. Th Pizza class only uses bakeDish() method but there is also a do-nothing method which is prepareDrink() which is not utilized by Pizza class.

- Interface Segregation principle states that no code should be forced to depend on methods it does not use. In both the classes Pizza and Mocktail there are methods which are not going to be used by those classes. Thus, it classes Interface Segregation principle.

Good Package:

- To remove the violation, I split the interface in two small interfaces : IBakeDish and IPrepareDrink. In IBakeDish only methods related to baking are declared and IPrepareDrink only methods related to preparing drinks are declared.
- Previously, in Mocktail class there was a method that class had to implement which is of no use. After creating small interface IPrepareDrink, Mocktail class only implements the IPrepareDrink having prepareDrink() method.

- Similarly, in Pizza class there was a method that class had to implement which is of no use. After creating small interface IBakeDish, Pizza class only implements the IBakeDish having bakeDish() method.
- The Main class in the good package is used to create objects of IBakeDish for Pizza and IPrepareDrink for Mocktail. This class is only responsible for invoking methods related to the classes.
- By creating small interfaces, I removed the forceful implementation of the methods which the classes do not require. Therefore, the interface segregation principle violation is fixed.