Assignment 2 CSCI 5308

a.Your name, CSID and Banner

My Name: Abhisha Thaker

CSID: athaker

Banner No.: B00937694

b. A link to your individual assignment repository on Gitlab

https://git.cs.dal.ca/courses/2023-summer/csci-5308/assignment2/athaker

c. A description of the classes and code in the bad package of why the code violates the corresponding principle.

1. SRP –Single Responsibility Principle Violation

The Single Responsibility Principle states the class should have one single reason to change. Class should only have one single responsibility or job and shouldn't be responsible for unrelated tasks

Student Class – have several responsibilities that are not related to the class and to each other.

1. Setters and Getters for student info such as bannerno, name, age, email and dob.
2. Managing enrolled courses – getEnrolledCoursesByBanner()
3. Managing grades by course – getGradesbyCourse()
4. Managing fees paid by the student per course and calculating pending fees and total fees - getFeesbyCourse (), getTotalFeesPaid () and pendingFees ()
5. Managing professors who teach the course – getTaughtByCourse ()

Main Class – Creates objects of this class and calls the respective methods.

'Student' class violates SRP as one class is doing multiple unrelated things. Therefore, let's create separate classes for student information, course enrollment, fees management, grades management.

Lines of Code - 162

2. Open/Closed Principle Violation

This principle states that the code should be open for extension, but closed for modification. This principle protects the code from consequences of modifying the code.

It has 2 classes.

1. Employee.java –

   This class violates the OCP, because of following reasons.

   - There are some methods in this class that is not closed for modification. Any change requires modifiying multiple methods within the 'Employee' class. Those methods are as follows –
   - 1. positionPartTimeSalary – It sets salary by positions for parttime employees by introducing conditional if-else statements. If in future, new positions are also added, then it would require modification in this method.
   - 2. positionFullTimeSalary - It sets salary by positions for fulltime employees by introducing conditional if-else statements. If in future, new positions are also added, then it would require modification in this method.
   - 3. setJobResponsibility – It sets job responsibility by different positions by introducing conditional if-else statements. If in future, new positions are added, then it would require modification in this method.
   - 4. setProjectType - It sets project type by different positions by introducing conditional if-else statements. If in future, new projects are assigned and new positions are added, then it would require modification in this method.

2. Main.java - Creates objects of this class and calls the respective methods

Lines of Code - 168

3.LSP – Liskov Substitution Principle Violation

It states that the objects of a superclass should be replaceable with objects of their subclasses without affecting the correctness of the program.

There are three classes

1. SeniorDeveloper class – superclass
2. JuniorDeveloper class – subclass - extends the SeniorDeveloper class

   But in the JuniorDeveloper class, the setYearsExperience() method restricts the years of experience. This behavior is different from the superclass, which does not have any restriction on the years of experience.

   Similarly with setSalary() method.

3. Intern class – subclass - extends the SeniorDeveloper class.

   But in the Intern class, the setYearsExperience() method restricts the years of experience to be less than or equal to 1. This behavior is different from the superclass, which does not have any restriction on the years of experience.

   Similarly with setSalary() method.

   If we use an instance of Intern of JuniorDeveloper in place of SeniorDeveloper, then it will lead to unexpected behavior because the expectations set by the super class ie. SeniorDeveloper doesn't met what's there in the subclass ie. Intern or JuniorDeveloper

Lines of Code - 192

4. Interface – Segregation Principle Violation

It states that classes shouldn't be forced to depend on interfaces that they don't use.

There is an interface 'developer' and 4 classes (Intern.java, juniorDeveloper.java, Manager.java, seniorDeveloper.java) that is implementing the 'developer' interface.

But Intern class implements the developer interface but does not support methods like debugCode(), reviewCode(), or manageTeam(). Similarly, the juniorDeveloper class does not support the reviewCode() and manageTeam() methods. Similarly, the Manager class doesn't support methods like debugCode(), writeCode(), reviewCode().

These classes are forced to implement methods they neither need nor use.

Lines of code - 314

5.Dependency Inversion Principle Violation

Dependency Inversion principle states that high-level modules shouldn't depend on low-level modules. Both should depend on abstractions.

There are 3 classes created

1. 'User Class' – This class directly depends on the concrete implementation of below 2 classes – User Authentication class and Profile Info class. It instantiates the objects in the constructor which makes it tightly coupled. If in future, different implementation is implemented, then it would make it difficult to switch without modifying the 'User' class.
2. User Authentication class

3. Profile Info class
4. Main.java - Creates objects of the user class and calls the respective methods

Lines of code - 190

d. A description of the classes and code in the good package. How did you fix the violations?

1. SRP – Single Responsibility Principle Resolution

Have created 4 classes (StudentCourse.java, StudentFees.java, StudentGrades.java and StudentInfo.java) out of one class – Student.java. These 4 classes separate the responsibilities into different classes.

1. StudentInfo.java – This class performs just one responsibility as a whole – i.e. Managing student information such as banner no, name, age, email and dob.
2. StudentCourse.java – This class manages course-related information. It stores and retrieves the professors for the course and the information about which courses are taken by the student.
3. StudentFees.java – This class manages the information about the fees for the enrolled courses and also contains information about how much fees is yet to be paid and also calculates total amount of fees.
4. StudentGrades.java – This class manages the grades obtained by a student for the enrolled courses.

Main. Java - Creates the object of the above class and calls the respective methods as and when required.

All the above 4 classes separate the responsibilities into different classes, which protects the code from consequences of modifying parts of the code.

Lines of code - 204

2.OCP – Open Closed Principle Resolution

We have resolved the OCP by creating the below classes and changing the inheritance hierarchy.

1. **Employee class** – It sets the name, age and email Id of the employee
2. **FullTimeEmployee class** – It extends the Employee class and stores the details of those employees who are full-time employees and calculates their salary and sets their position.
3. **PartTimeEmployee class** - It extends the Employee class and stores the details of those employees who are part-time employees and calculates their salary and sets their position.
4. **Developer class** – It extends the employee class and it’s methods stores the details regarding the developer.

5. **Business Analyst class** - It extends the employee class and it's methods stores the details regarding the business analyst.
6. **Manager class** - It extends the employee class and it's methods stores the details regarding the Manager.
7. **HR class** – It extends the employee class and its methods store the details regarding the HR.

Now, these subclasses which extends the employee class – can add new behaviors and methods without modifying the employee class. If in future, new class – requires to be added such as Machine Learning Enginner or ContractEmployee – It can added seamlessly without modifying the employee class.

Lines of code - 222

3. LSP – Liskov Substitution Principle Resolution

Have created 3 abstract classes

1. abstractinternDev
2. Developer – this abstract class extends the abstractinternDev
3. seniorDev – this abstract class extends the Developer abstract class

Have created 3 concrete classes.

1. Intern – extends abstractinternDev
2. JuniorDeveloper – extends Developer
3. SeniorDeveloper – extends seniorDev

Main.java – Creates instances and calls the respective methods as and when required.

Now, the overridden methods in the subclasses have the same input-output behavior as the superclass, ensuring that objects of the subclasses can be used interchangeably without affecting the correctness of the program.

Lines of code - 189

4. ISP – Interface Segregation Principle Resolution

As a resolution, have created 4 interfaces. These interfaces are implemented by classes that actually need and use them. They are as follows -

1. codeDebugger – implemented by juniorDeveloper and seniorDeveloper
2. codeReviewer – implemented by seniorDeveloper
3. codeWriter – implemented by intern, juniorDeveloper and seniorDeveloper
4. teamManager - implemented by seniorDeveloper and Manager

Main. Java - Creates objects and calls the respective methods as and when required.

Lines of Code - 249

5.DI – Dependency Inversion Principle Resolution

As a resolution, have created 2 interfaces which are implemented by 2 classes

1. AuthenticationInterface – implemented by UserAuthentication class
2. ProfileInformation – implemented by ProfileInfo class
3. User class

And In the user class, have made the constructor depend on the above interfaces and passed it as an argument or dependency to constructor, so that value is passed from the outside, rather than instantiating these classes inside the constructor

Main. Java - Creates objects and calls the respective methods as and when required.

Lines of Code - 209

All lines of code are calculated approximately.

References

[1] https://en.wikipedia.org/wiki/SOLID

[2] https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/

[3] https://blog.knoldus.com/what-is-liskov-substitution-principle-lsp-with-real-world-examples/#:~:text=Simply%20put%2C%20the%20Liskov%20Substitution,the%20objects%20of%20our%20superclass.

[4] https://stackify.com/solid-design-liskov-substitution-principle/

[5] https://stackoverflow.com/questions/56860/what-is-an-example-of-the-liskov-substitution-principle

[6] https://dev.to/tamerlang/understanding-solid-principles-dependency-inversion-1b0f

[7] https://stackify.com/dependency-injection/#:~:text=Dependency%20injection%20is%20a%20programming,inversion%20and%20single%20responsibility%20principles.

[8] https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

[9] https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html