Review Quiz Answers

True/False

Please indicate if each statement is `true` or `false`.

Statements

1. **Continuous Integration** is a software testing technique aimed to expose faults in the *interaction* between integrated units ; it must be done at the end of each agile (development) iteration.

   ANSWER

   - **False**: Continuous Integration (CI) is a *software development practice* where developers integrate code into a shared repository frequently, ideally several times a day!

2. The **dependency inversion principle** in object-oriented design is always being used if we define and use interfaces. ::tip ANSWER **False**: Defining Interfaces does not necessarily mean we comply with DIP; DIP says High-level classes shouldn't have to change because low-level classes change. For example, we may define an interface, but a dependent class might still continue to depend on a subclass of the newly defined interface instead of depending on the interface :::
3. In Object-Oriented programming (OOP), the primary purpose of "polymorphism" is to reuse your code.

   ANSWER

   False: The primary purpose of *Inheritance* is code reuse, but Polymorphism (having many forms) is the ability of an entity in an object-oriented model to refer to objects of different classes at different times

4. Coupling is a measure of how strongly one element in code (such as a class or a method) is *focused* and comprise of responsibilities which belong together.

ANSWER

False: coupling refers to the degree of direct knowledge that one element has of another. In other words, how often do changes in class A force related changes in class B.

5. Open-closed principle states when extending a class, consider that you should be able to pass objects of the subclass in place of objects of the parent class without breaking the client code.

ANSWER

False: OCP states classes should be open for extension but closed for modification.

6. Adapter is a structural design pattern that allows objects with *incompatible* interfaces to collaborate

ANSWER

True

Multiple-Choices

Please indicate the correct choice (there is only one for each question).

Questions

1. In software development, "**requirements**" are about:

   - [x] a. what the software system should do
   - [ ] b. how the software system should work
   - [ ] c. both about what the software system should do and how the system should work
   - [ ] d. none of the above

2. When the **REST principle** is being used to describe the interactions between a *client* and a *server*:

- ☐ a. the **server** maintains the state of the client
- ☐ b. the client can send different kinds of http requests: *create*, *get*, *put*, *post* and *delete*
- ☑ c. the **client** maintains state
- ☐ d. both the client and the server maintain state

3. When using an agile development process, the **product backlog**:

- ☐ a. is a list of the products a company needs to build
- ☑ b. is a prioritized list of backlog items specific to the product under development
- ☐ c. is a list of defect associated with the current product
- ☐ d. is a list of engineering tasks negotiated by the team and the customer about the current development project

4. Which SOLID design principle is most closely described by the given statement: **classes should depend on abstractions rather than implementations.**

- ☐ a. Single Responsibility Principle
- ☐ b. Open/Closed Principle.
- ☐ c. Liskov Substitution Principle.
- ☐ d. Interface Segregation Principle
- ☑ e. Dependency Inversion Principle

5. DRY design principle is associated with which code smell?

- ☑ a. code duplicates
- ☐ b. data clumps
- ☐ c. primitive obsession
- ☐ d. a and b
- ☐ e. none

6. Which statement is always true about XSS?

- ☐ a. It is a server-side attack
- ☑ b. It is an injection attack similar to SQL injection
- ☐ c. a and b

- ☐ d. attacker's malicious code is stored in server-side before being sent and getting executed in the victims' browsers.
- ☐ e. none

Analysis and Design 1

Imagine you have been asked to implement a **Piazza-like** system (Q/A and discussion for courses). One of your team members defines the following User Story:

As a professor, I would like to be able to post announcements so that they appear on my students' feed.

Part 1

You are implementing this User Story; what classes will be in your "model" UML class diagram? (Only name the classes .)

ANSWER

Professor, Announcement, and Student, Feed can be possible model classes.

Part 2

The software system is implemented according to the Client-Server Architecture. To show your understanding of this architecture, describe how the above-stated User Story carries out through the interaction between different entities (user, client, server, database, ...) .

TIP

Here is a possible flow of events: The professor logs into the system as an authenicated user with "making announcements" privileges. She then most likely uses the front-end functionalities to compose a announcement and submit. This will trigger a HTTP "POST" request that is sent to the server containing the announcement's info e.g. announcement author, course/class code, announcement subject, announcement body announcement tags, date/time, etc. The server receives the POST request and creates an announcement object. Then it processes it as follows: it creates a new entry in the "Announcement" table

and stores the announcement details there. It also stores information about its author. This could be in the form of a foreign key constraint on "Users"/"Professors" table. Then, it fetches the list of students based on class/course code (and/or based on the professor) and adds the timestamped announcement to each student's feeds (and possibly sends out a notification). Upon successful completion of this process, the server sends back an appropriate responce i.e. code *201* perhaps in this case, to the client. The client application updates the feeds accordingly to include the newly made announcement. Note that this is just one possible scenario and might very well change as the design of the system changes.

Part 3

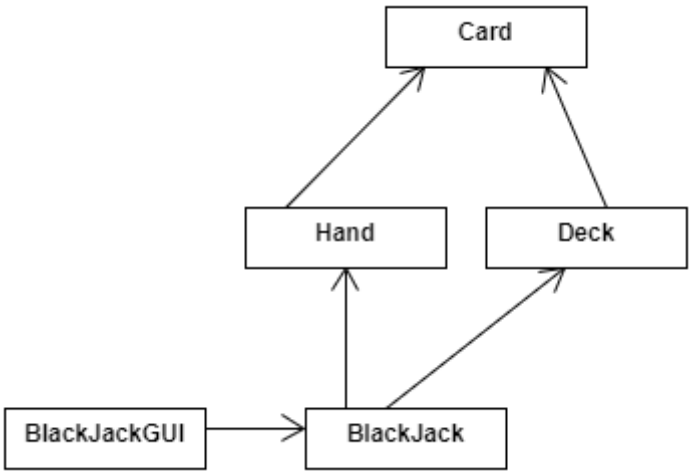
Based on the above-stated User Story (and your general understanding of how a Piazza-like system works), what design pattern(s) [among those we covered in lecture/readings] would be primed for application here. **Name** only one **and elaborate (briefly) on the underlying problem and the proposed solution by the pattern (relate that to how the pattern fits here)**.

ANSWER

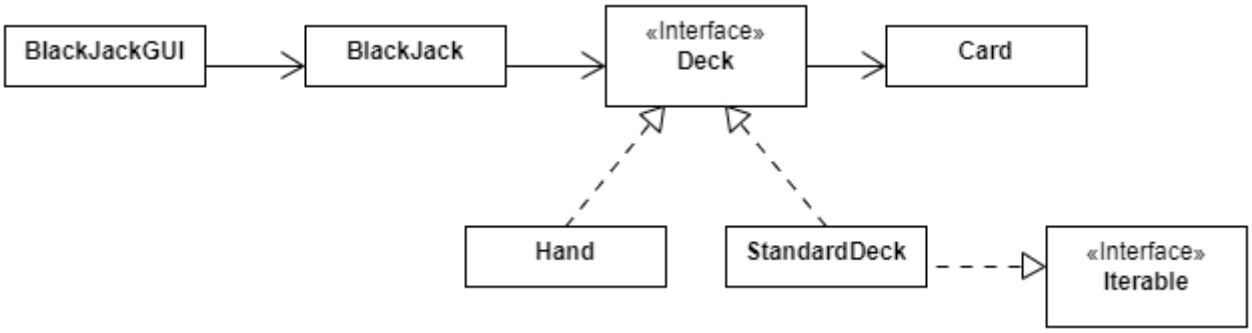
Observer pattern could very well serve the needs of the system here. The announcement is the subject here and students or studens' feeds can be observers (aka subscribers) who are interested in the subject. Whenever the subject's state changes e.g. a new announcement is made, observers are notified/updated automatically. Using the observer pattern here allows us to have loosely coupled observers and subject where the subject does not even need to know about the specific observers. Therefore, it is "easy" to extend the model so that other entities can make announcements (e.g. Course Assistants) and/or other entities can be added as observers.

Analysis and Design 2

A team of students are building card game application. They are considering to implement the game of Blackjack in their first two iterations. Their UML design includes the following:



The team advisor suggested this alternative design:



Part 1

Briefly explain what SOLID design principle(s) the advisor's revised design adheres to.

ANSWER

- Dependency Inversion Principle: An abstraction (Deck interface) is introduced between the high-level classes (Blackjack & BlackjackGUI) and low-level classes (Hand & StandardDeck) that changes the direction of the dependency and splits the dependency between the high-level and low-level modules.
- Open/Closed Principle: To extend the application, e.g. a Blackjack Shoe, all needs to be done is to add another concrete implementation of the Deck interface. The extention (adding a new Deck) will not require any changes to the classes already exsiting in the model.
- Interface Segregation Principle: A concrete implementation of Deck does not have to implement Iterable; the Deck interface is kept small

Part 2

Briefly explain what design patterns (among those we've learned in this course) the advisor's revised design adheres to.

ANSWER

Iterator Pattern: StandardDeck implements Iterable which is an indication of an attempt to employ the Iterator Pattern. It allows iterating over the Cards in a StandardDeck (perhaps in random order to simulate the shuffling and drawing behaviours).

Part 3

You are asked to criticize the advisor's revised design; what would you suggest?

ANSWER

One reasonable argument is that applying the design principles/patterns has made the design more complicated than it should be. The added complexity can slow down the delivery for this iteration. The group could go ahead with their implementation and refactor their code in future iterations when ading new features. Other reasonable arguments can be made here as well.

Design and Analysis 3

Suppose you are designing a software application that will allow the users to perform task management. The user can add tasks to the system and can group tasks together into projects. Projects can be added as sub-projects of other projects, nested arbitrarily deep. Each tasks has an estimated time for completion that is specified when the task is constructed. You want to be able to treat individual tasks and projects in the same way. In particular, you want to be able to get the time needed to complete a task or a project. The time taken to complete a project is the total time needed to complete all the tasks in the project or in sub-projects of that project.

Part 1

Based on the software description, write two "must have" (functional) requirement in form of User Stories.

POSSIBLE ANSWER

- As a user, I want to group tasks into a project so that I can better manage my tasks.
- As a user, I want to assign estimated completion time to each task so that the software give me the time needed to complete a project.

Part 2

This application conforms to the Client-Server software architecture. To show your understanding of this architecture, describe one use-case (a scenario involving a user using the proposed software) and indicate the interaction between different entities (user, client, server, database, ...) involved in the use-case.

POSSIBLE ANSWER

- (User clicks on "add task" button on the client application.
- The client application collects the information provided for the task and sends a request to the server to create and store the task.
- The server recieves the requests. It creates a task with the provided information. Stores the task in the database. Upon sucessfull completion of this process, the server sends a response to the client application.
- The client application, upon reciving the server's response, displays the newly added task in the list of tasks.

Part 3

Based on the software description, what design pattern(s) [among those we covered in lecture/readings] would apply to the design of this application. **Name** the design pattern and **elaborate** (briefly) on the problem and proposed solution (how it fits here).

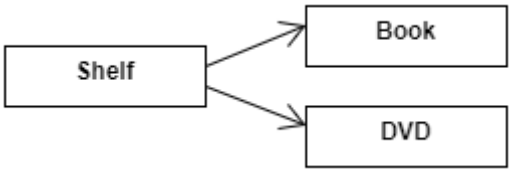
POSSIBLE ANSWER

Composite Design Pattern:

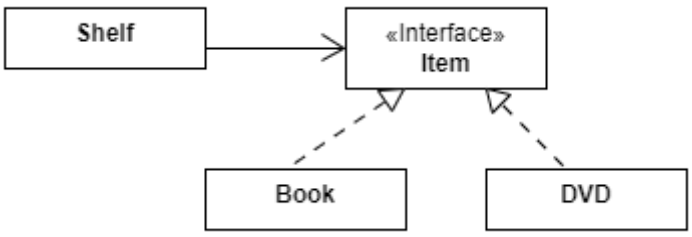
- Problem: We need to manipulate a hierarchical collection of "primitive" and "aggregate" objects. Moreover, we need to process (treat) aggregate objects the same way as primitive objects.
    - The primitive object is a Task.
    - The aggregate object is a Project.
- Solution:
    - Following the pattern, we can create an abstraction like AbstractTask class with an abstract method getEstimatedCompletionTime().
    - We implement a Task class that extends AbstractTask.
    - We implement a Project class that extends AbstractTask but also contains a collection of AbstractTasks with methods to add/remove AbstractTasks.

Analysis and Design 4

A team of students are building a Bookstore Management Software. The UML design for the *first iteration* includes the following:



The team advisor suggested this alternative design:



Explain what SOLID design principle(s) the advisor's revised design adheres to.

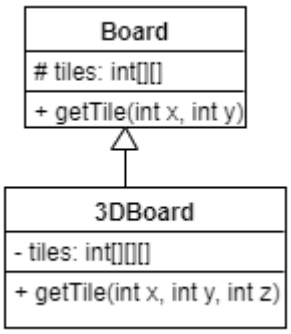
ANSWER

Dependency Inversion Principle: An abstraction (Item interface) is introduced between the high-level classes (Shelf) and low-level classes (Book & DVD) that changes the direction of the dependency and splits the dependency between the high-level and low-level modules.

Open/Closed Principle: To extend the application, e.g. to use other "items", all needs to be done is to add another concrete implementation of the Item interface. The extention (adding a new Item) will not require any changes to the classes already exsiting in the model.

Analysis and Design 5

Assume you have implemented a board game (e.g. chess) that uses a 2D square-shaped board. You have defined a class `Board` for it that stores all the peices (i.e. tiles) in it in an array. You have also implemented a bunch of methods inside this class including setters and getters, among which there is, of course, `getTile()` which accepts `x` and `y` coordinates and returns the tile that is in `(x, y)` location. Later, the requirements change and now you need to support 3D boards as well. A 3D board *is a* board, so you decide to extend from the exising `Board` class and come up with the following design



Is this a reasonable design? Answer the question by taking into account the SOLID principles. If it is, why? If not, what principle(s) does it go against and how/why? Explain in details. Also, if it is not a sensible design, explain how you would fix/improve it.

ANSWER

This design goes against Liskov substitution principle (LSP). At first glance, it may seem a good decision to make `3DBoard` a subclass of `Board` class. `Board` provides a 2D board and functionalities over two dimensions, and 3DBoard can inherit all that and add a new

dimension. The problem though is that many of `Board` 's methods, such as the given `getTile()` , are designed to work with two dimension units, not three. So, they lose their context/meaning when it comes to three dimensions. For instance, `getUnit(int, int)` loses its meaning in `3DBoard` as `getUnit(int, int, int)` makes more sense. Thus, a client code attempting to use the `3DBoard` class as its base class Board would be very out of luck, which is what LSP is about i.e. being able to substitue base class objects with subclass objects.

Edit this page