OOSE Quiz [Spring 2020]

- This is a closed book exam. You may **not** use any written/printed/online notes.
- This quiz has 5 problems (including this) and total of **40 points**.
- You have 45 minutes to complete the quiz.
- You may **not** use any communication devices (phone, etc.) while examination is in progress.
- Don't spend too much time on any problem.
- Good luck!

True/False

Each statement is worth **1 point**.

Statements

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.
2. The **Post/Redirect/Get (PRG)** is a web development design pattern that lets the *page* shown after a *form submission* be reloaded without ill effects, such as submitting the form another time.
3. The "**planning fallacy**" is a phenomenon in which predictions about how much time will be needed to complete a future task display a pessimism bias and *overestimate* the time needed.
4. The **dependency inversion principle** in object-oriented design is always being used if we define and use interfaces.

```
// Answers
1. false (the definition is for Integration testing)
2. true
3. false (planning fallacy has an optimistic bias and underestimates)
4. false (not every use of interface is about dependency inversion)
```

Multiple-Choices

Each question is worth **1 point**.

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 up to five kinds of http requests: *create*, *get*, *put*, *post* and *delete*
    - [x] 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
    - [x] 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
    - [x] e. Dependency Inversion Principle

Scenario

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.) [**2 pts**]

```
- Professor
- Student
- Feed

You many have other reasonable classes (like Announcement) but be careful with
e.g. a class called Post (is it used as announcement, or is it the "behaviour" of
posting that is turned to a class).
```

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, ...) . [**4 pts**]

```
Professor clicks on "post" button on the client application.

The client application collects the "announcement" provided by the professor
sends a request to the server to create and store the Announcement.

The server recieves the requests. It creates an Announcement object with the
provided text. Stores the announcement in the database. It then pushes the
announcement to the students' feed and sends "notification" to all students (the
latter would involve more interaction with the database to e.g. update feed,
retrive emails, etc). 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 update
"feed" which would include the newly made announcement.
```

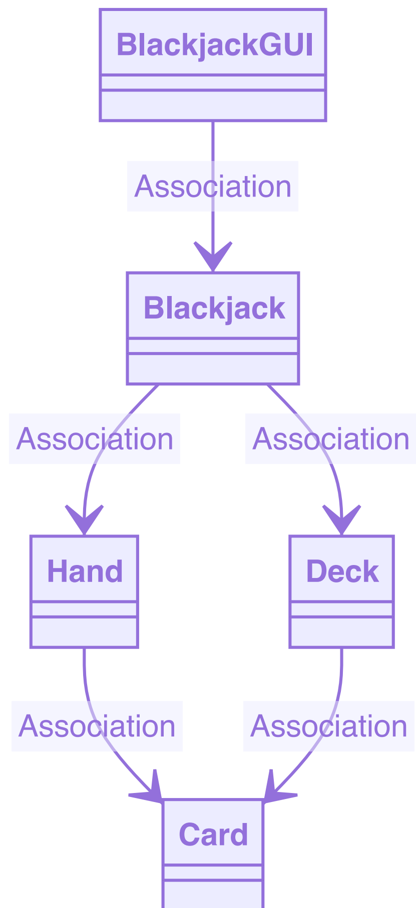
Part 3

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). [**10 pts**]

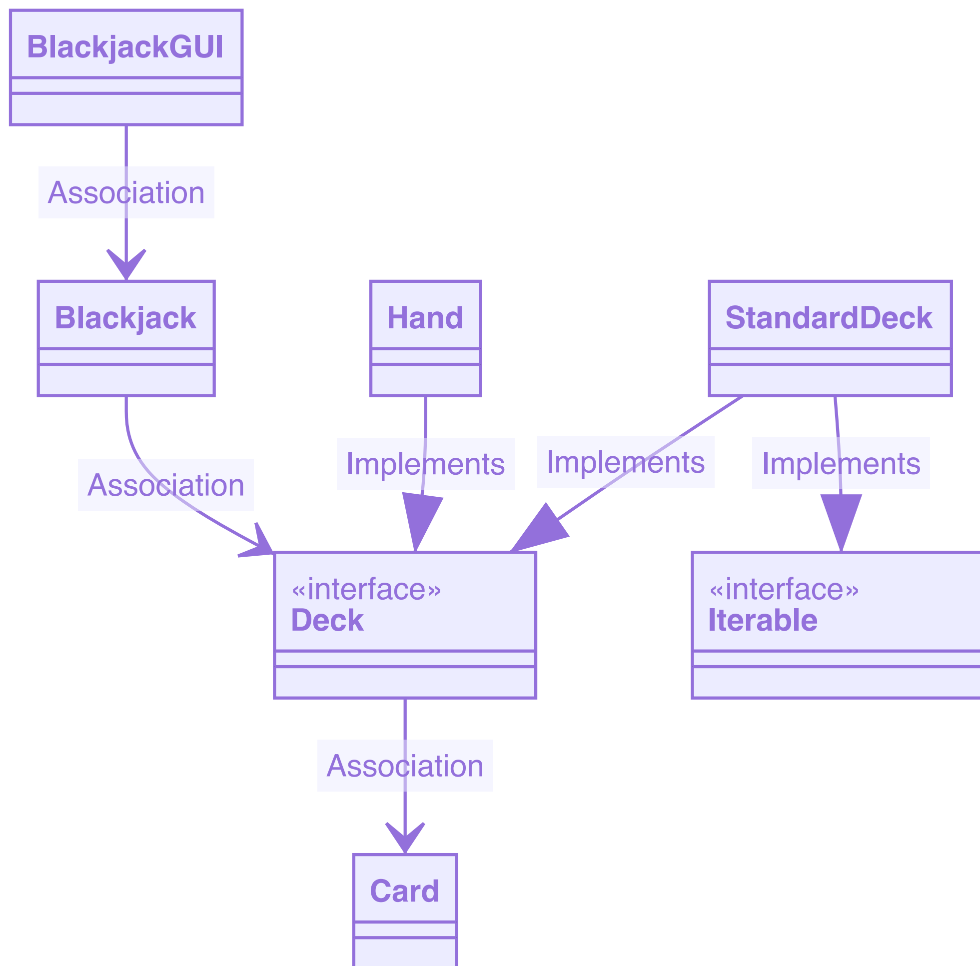
```
-- Observer Desgin Pattern --
The user story indicates the need for some sort of a subscription mechanism to
notify multiple students about any announcements that was made by the professor.

In this model, the students would be observer. The announcement would be the
subject (observable).

Using the Observer pattern allows us to have loosely coupled observers to
subject. 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.
```

Design Review

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. [**7 pts**]

```
-- Dependency Inversion Principle -- [2 pts]
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 -- [2 pts]
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 -- [3 pts]
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. [**3 pts**]

```
-- 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 is the best point you have? [**2 pts**]

```
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.

Consider any other reasonable argument as an acceptable answer; if in doubt,
please consult with the instructor before assigning a grade point.
```

Update your Resume!

After taking this course, you've updated your resume and under EN.601.421 OOSE, you've added the following:

Collaborated with a team of students for development of a group software project using GitHub workflow.

During a phone interview, a recruiter asks you to elaborate on using GitHub workflow for your OOSE course project.

Summarize your answer (which you would give to the recruiter) in a paragraph. We expect you to highlight two GitHub workflows in your answer. [**4pts**]

```
Look for (a subset of) the following in an answer:

• Communicating in issues
• Creating branches (for adding/updating features)
• Use commits to keep track of your progress as you work on a feature branch
• Using pull request for merging changes

(Or anything else that can be considered as part of the GitHub workflow)
```