In-Class Activity

Friday, April 26, 2019 9:00 PM

M3

What are the similarities between Flyweight and Singleton Design Patterns?
What are the fundamental differences between Flyweight and Singleton Design Patterns?

M5

how do you approach the function of adding Primitive instances to Composite? What's the tradeoffs of your solution to other alternatives?

M6

Please select if the method declaration has violated Liskov Substitution Principle or not, and give your explanations.

1. public Course recommend(UndergradCourse pCourseID);
2. public UndergradCourse recommend(Course pCourseID);
3. public UndergradCourse recommend(Object pCourseID);
4. public Course recommend(Course pCourseID) throw SomeCheckedException;

Explanations to your answers above

M7

Choose the proper participant (subject, observer, concrete subject, concrete observer) in the observer design pattern who could perform the described responsibilities.

1. Defines an updating interface for objects that should be notified of changes in a subject.
2. Implements the updating interface to keep its state consistent with the subject's.
3. Provides an interface for notifying observers.
4. Stores state that should stay consistent with the subject's.
5. Maintains a reference to a ConcreteSubject object.
6. Sends a notification to its observers when its state changes.
7. Provides an interface for attaching and detaching Observer objects
8. Stores state of interest to ConcreteObserver objects.

M8

Evaluate if the sentence in each question can be a consequence of applying Visitor design pattern.

1. A visitor gathers related operations and separates unrelated ones.
2. Adding new ConcreteElement classes becomes easy.
3. ConcreteElement interface has to provide public operations that access an element's internal state, so it may compromise its encapsulation.
4. Visitor pattern makes it relatively easy to accumulate state since single visitor can visit each element in the object structure.

M8-2

Please select the most appropriate design pattern (Decorator, Adaptor, Proxy, Composite) for the given description of intention.

1. This design pattern allows you to add responsibilities to objects without subclassing. It avoids the explosion of subclasses that can arise from trying to cover every combination of responsibilities statically.
2. This design pattern allows you to structure classes so that many related objects can be treated uniformly, and multiple objects can be treated as one.
3. The intent for this design pattern is to provide a stand-in for a subject when it's inconvenient or undesirable to access the subject directly.
4. This design pattern is used to resolve incompatibilities between two existing interfaces proving identical functionalities.

M10

Questions about the refactoring and the code demo.

What problems you have identified in the code demo?
How do you propose to fix those problems?
What do you think is *Refactoring*

M11

Please select the three heuristics from the Usability Heuristics for User Interface Design (https://www.nngroup.com/articles/ten-usability-heuristics/), and evaluate how JetUML complies or violates those heuristics.

M12

Which of the various pressures contributed to the outcome of this case?

Mission impossible: being asked to create or accept a product schedule that is clearly impossible to meet
Mea Culpa: delivering products without key functionality or with known defects
Rush Jobs: delivering products of subpar quality to meet schedule pressures
Red Lies: telling clients or management known falsehoods about product schedule or performance
Fictionware: promising features that are infeasible
Nondiligence: inadequate review of requests for proposals, contracts or specifications

A mid-level sales manager at LifeDesign, a medical systems engineering firm, receives a request for proposal (RFP) to develop a radiation-delivery system for use in outpatient hospital settings. The RFP specifies that the product must have a guaranteed failsafe mechanism to prevent radiation overdoses due to user input error. The sales manager reviews the RFP with his development team, who tell him that A) the system cannot be built to specifications within the desired timeframe and B) there is no way in this kind of system to build in a guaranteed failsafe against operator error; proper radiation dosages vary so much by patient that there is no way for the system to enforce safe limits by itself; it must rely on accurate user input of the dosage instructions on the prescription. The best we can offer, the team tells him, is a warning system that will loudly prompt the user to verify and double-verify the correct input. But, the team tells him, it is well-known that such warnings are often mindlessly canceled by users who find them annoying, so this mechanism is hardly

guaranteed to prevent operator error. The sales manager goes back to the client with a proposal that promises delivery of all functionality within the specified time frame.

When they receive the proposal, which has been accepted and contracted with the client, the engineering team begins to design the system. But it is three weeks into the work before anyone notices the delivery date – the design team leader immediately appeals to the sales manager, reminding him that he was told this was an impossible date to meet. He tells her that this is unfortunate but that they are under contract now, and there will be severe financial penalties for a late delivery. He reminds her that upper management won't be happy with any of them in that case, and tells her that her team had better find a way to get it done. The team leader goes back to her group, frustrated but resigned, and tells them to get it done somehow. No one has even noticed yet that the contract also includes language about the guaranteed failsafe against bad user input.

Halfway through the delivery schedule, the sales manager and design team leader sit down with the client for an update. Despite being well behind schedule, they put on a good show for the client, who leaves confident that all is as it should be and that the product will deliver full functionality on time. Weeks before the due date, the team leader is telling her team that their jobs depend on an on-time delivery. Her engineering team has resorted to desperate measures – especially the group responsible for the system software, who are taking shortcuts, hiding system errors, doing sloppy coding and subcontracting work out to third-parties without proper review of their qualifications. When the product is finally assembled, there is no time to test the full system in the actual client/user environment, so the quality-assurance team of LifeDesign relies on computer simulations to test its operation. These simulations make many flawed assumptions due to the rush schedule and a lack of information about the user context/environment. None of the simulations assume erroneous dosage input by users.

The product is delivered on time, and represented as having full functionality. As a result, the client tells its therapists using the new system that it has an advanced failsafe mechanism that will prevent radiation overdoses from user error. Two years into its use, dozens of patients receiving radiation from the system, including young children and others with curable diseases, have been poisoned by radiation overdoses. Those who are not already ill or dying from overdose-related effects must now live with the knowledge that they have a severely elevated chance of developing fatal bladder or kidney cancers from the overdoses. An external review of the system by federal regulators and litigators reveals that not only was there no 'guaranteed' failsafe mechanism, the warning system that did exist was so buggy that it would often not be triggered at all, or would behave so erratically that most users dismissed it whenever it was triggered. Lawsuits are piling up, the media is on the story, and everyone is pointing fingers – at the radiation therapists, at the client, and especially at LifeDesign.