327 Final Exam Review

12/8/2021

Professor Barron

Logistics

- In class, on paper
- ~90 minutes, but you can stay till the end of the period
- 1 single sided note page allowed (8.5x11)
  - turned in with your exam
  - must be your own
- Done individually
- Don’t discuss the exam until grades are given back (in case make-up exams are necessary)
- Bring your Yale ID to authenticate yourself when you turn in

Topics

- C++
  - Comparisons with Python
  - Recognize and understand syntax and semantics we covered
    - Constructors/Destructors
    - References
    - Inheritance
    - Polymorphism
    - Operator overloading
    - Abstract classes
    - public/private/protected
    - const

Topics

- Design patterns
    - Iterator
    - Decorator
    - Observer
    - Strategy
    - State
    - Singleton
    - Template method
    - Adapter
    - Façade
    - Flyweight
    - Command
    - Factory method
    - Abstract factory
    - Composite
    - Memento

Topics

- Concurrency
  - Challenges
  - Threads/Processes
  - Multiprocessing pools
  - Queues
  - async/await coroutines

Topics

- Refactoring
    - technical debt
    - code smells
    - bloaters
    - oop abusers
    - dispensables
    - couplers
    - change preventers

Types of questions

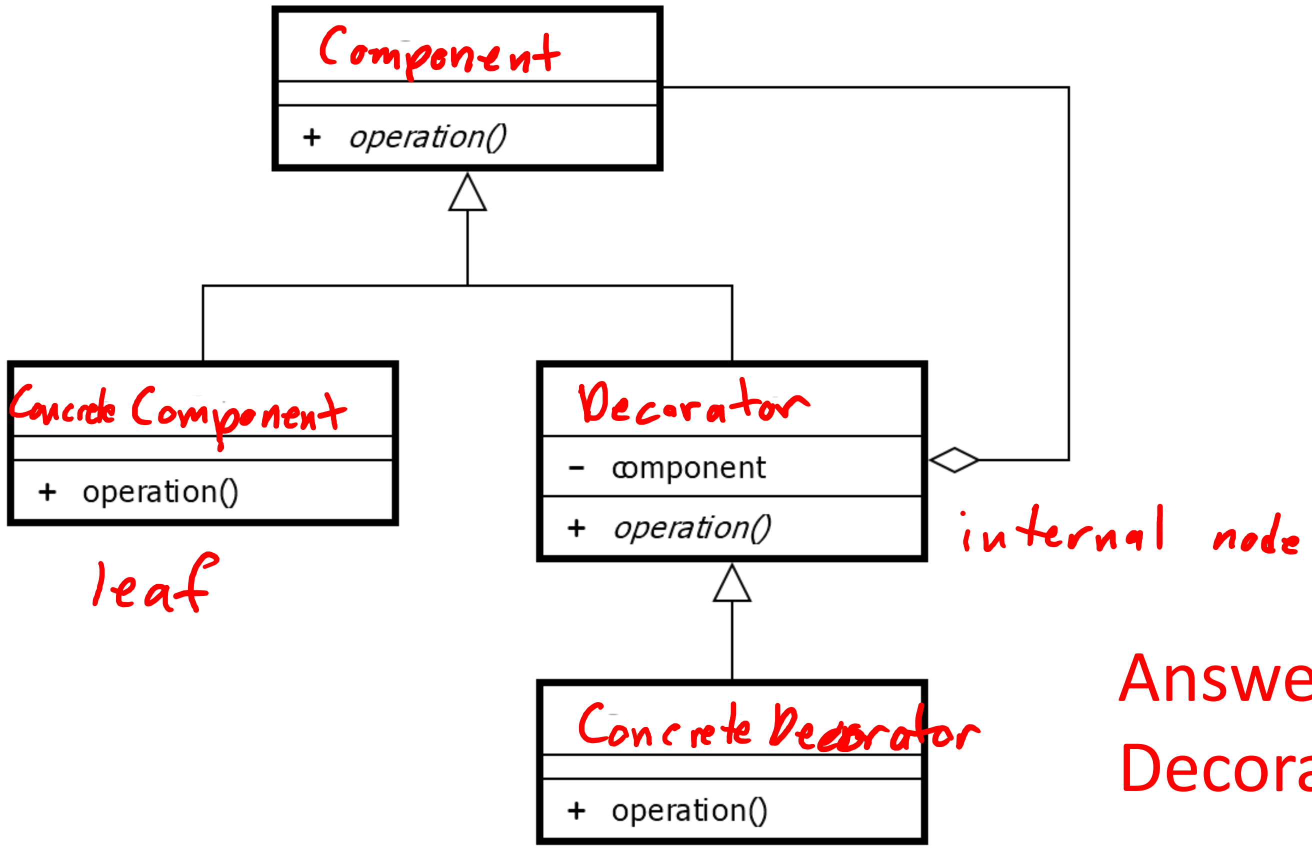
- Similar to midterm
- Many questions like those on the quizzes
  - true/false, multiple choice, fill in the blank
- Shorts answers
  - code
  - descriptions or explanations
- When asked to write code it will be brief snippets
  - writing code on paper can be awkward
  - we’ll be lenient in a lot of cases as long as the main idea we’re asking about is correct

Examples

- Compare the purpose or usage of two patterns
- Describe an approach to refactoring a given code smell
- Given a description of a problem, identify the applicable design pattern and describe how it would be used

Sample Question

The UML diagram below has had its class names removed.
What design pattern does this represent?



Answer:
Decorator or Composite

Sample Question

Imagine you are writing part of the backend logic for a web application for handling pizza delivery orders. When a customer places an order it goes through the following stages.

1. Pending
2. Confirmed
3. In the oven
4. On its way
5. Delivered

When the customer refreshes the page we want to show the current status. What is shown here will vary based on the current stage. For example, once confirmed we could show an estimated wait time. When it's in the oven there could be an animation of a pizza being prepared. Once it's on its way it could show a map with the location of the driver.

Let us focus **only** on the module that handles what view should be sent to the user. That means we are **not** concerned with how the map works, how the restaurant communicates when the pizza is ready, how to structure the HTML responses, etc. We'll assume these are someone else's responsibility and we just want to program to an abstract interface.

5 points: What design pattern that we discussed in class would be most useful here?

10 points: Explain how you would implement this part of the application using this pattern.

Answer

The State pattern would be most useful here, since there are 5 clear states for the pizza delivery. Based on the state of the object, either pending, confirmed, in the oven, on its way, or delivered, the behavior of the page changes.

User:

The User class is the pizza place that updates the status as they are working on the pizza. The user will have a reference to Context change use change_state() based on what stage they are in making the pizza.

Context:

The Context class has a reference to the current_state, which would start as Pending. It would have a method such as status() that would call show_status() on current_state. There would also be a method to change_state() which is a public setter for the state. The state may be changed based on the User.

State Interface:

This is the State interface that interacts with the Context class. There is no data, only one abstract method here called show_status(). Each subclass of State would also have no data and would implement its own version of show_status(). The subclasses of State are Pending, Confirmed, InTheOven, OnItsWay, and Delivered. For each subclass, show_status() would render a different image indicating the status.

Sample Question

Process pool (or thread pool) is one of many concurrency design patterns used to make parallel programming easier. What advantages does it offer over using the Process class directly?

Answer

1. Reduce overhead of creating new processes by reusing a small number of them for many tasks.
2. Convenient interface for dividing tasks, scheduling on the pool, and collecting results.

Answer

Process pool allows us to do multiple tasks for each process if we have more tasks than cores. As soon as a process is done with a task, it goes back to a queue of tasks until all of the tasks are done. Therefore, pool allows us to schedule and distribute tasks as soon as processes are available. Furthermore, by default pool will create a number of processes equal to the number of CPUS on the machine instead of having to create processes individually and join them again. All that is required is to create the Pool(), and then map tasks and inputs to the processes. pool.map() is useful because it can return the results of many independent tasks in a list without having to join the processes and will wait for all of them to finish. However, pool also offers asynchronous methods so results will come as they finish and other behaviors can be done with the finished results.