Washington State University

School of Electrical Engineering and Computer Science

CptS 487 – Software Design & Architecture Spring 2022

Midterm

WSU ID: 011730677

Name: Xinyu Liu

Question:	Max points:	Score:
1&2	12	
3&4	8	
5	10	
6	10	
7	5	
Total	**45**	

1) Multiple choice [2pt each]:

I. Composite pattern lets client _____

(a) separate interface and implementation
(b) access the elements of an aggregate object
(c) treat individual and composition objects uniformly
(d) use an existing class when the interface does not match

II. The "Liskov Substitution" Principle is best described by which of the following?

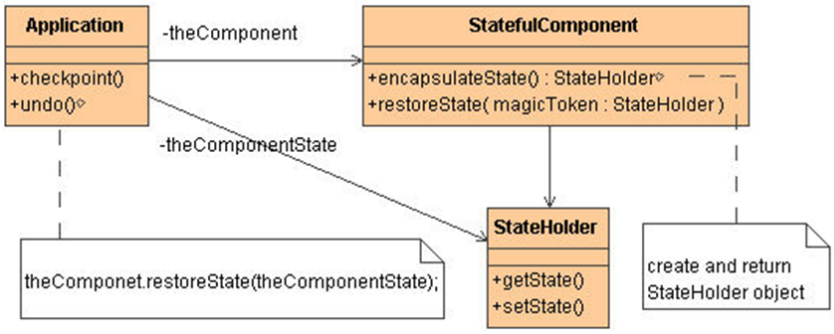
(a) derived object may be treated as if it is the base object.
(b) derived object should be replaced by its base object.
(c) derived objects should be used instead of base objects.
(d) None of the above

III. Develop software architecture include

(a) Dividing software into subsystems
(b) Defining global control flow
(c) Determining the subsystems' interfaces
(d) All of the above

IV. Which pattern does the following model represents?
A

(a) Memento
(b) Factory
(c) Builder
(d) Command



2) True or False [1pt each]:

I. **T / F** In UML class diagram, composition is a kind of association.

II. **T / F** An abstract class is primarily used as a template for defining other classes, and no instances can be created from abstract classes.

III. **T / F** Creational Design Patterns and Structural Design Patterns can not be used together.

IV. T / F Cohesion refers to elements in the same module, whereas coupling refers to elements in different modules.

3) [4pts] Briefly describes the differences between Factory Method pattern and Builder pattern.

They are both creational patterns. However, the builder pattern creates more complicated objects by calling multiple steps on the builder class, whereas factory creates the objects on one call.

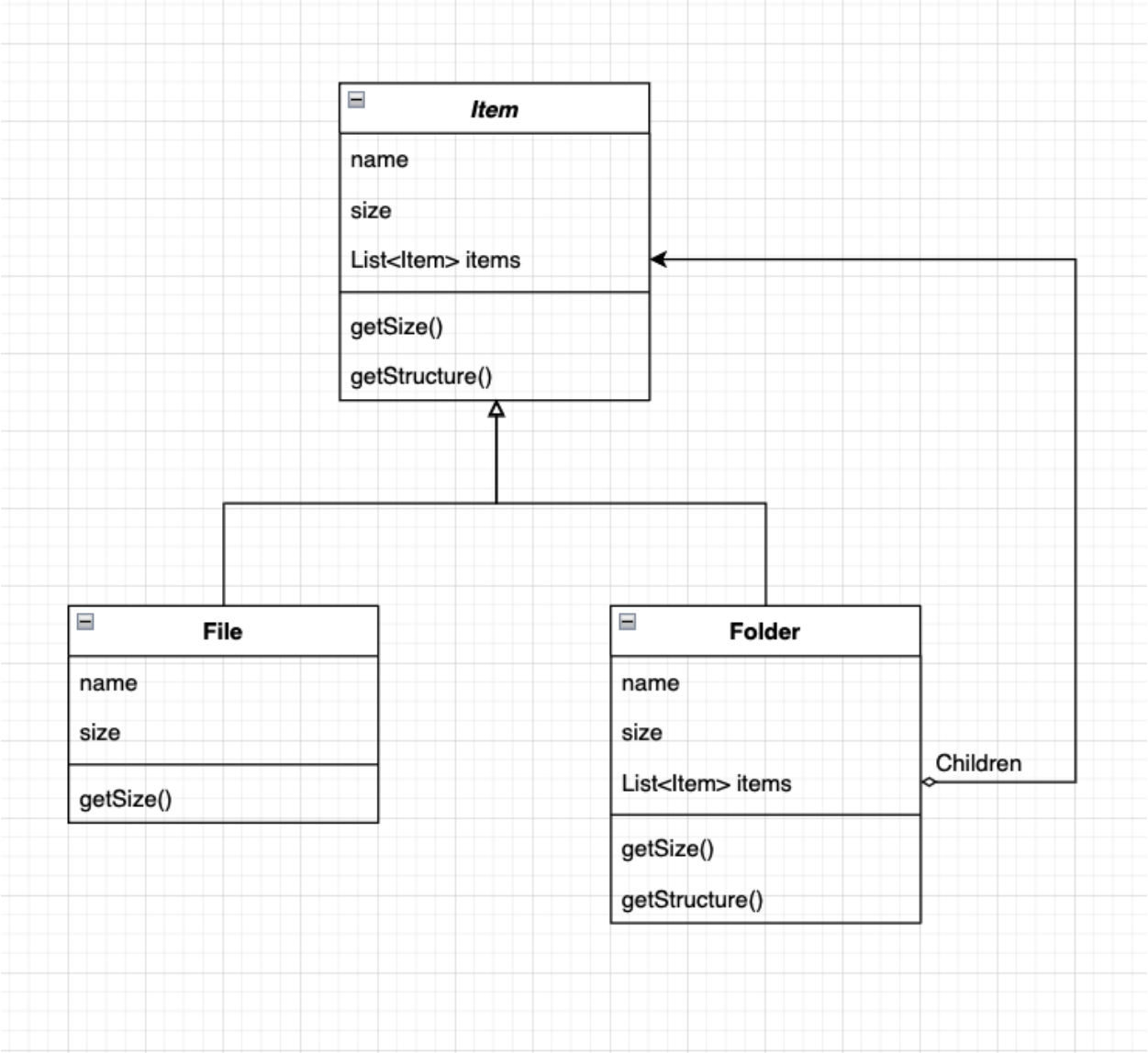
4) [4pts] Contrast and compare an Abstract Class vs. an Interface

Abstract class and interface both enforce a pattern for their subclasses. Abstract can have some implementations but interface can only have method signature. Abstract is used when subclasses inherit certain attributes and patterns, in contrast interfaces focus more on common methods and functionality.

5) [10pts] A folder in a file system contains items that could be other folders or files. The file system keeps track of the folder names and of file names and sizes. We also want to be able to display the whole file structure, with the names of all folders and the names and sizes of all files.

What's the most appropriate design pattern to use to address the problem? Show an appropriate class diagram with the attributes and operations required for this specific example *and* required for the design pattern.

The best way to design this file system is to use a composite pattern. The base class "Item" can be inherited by subclasses "file" and "folder". Inside each folder, it can hold a list of items.



Assume all attributes are public for lazy access.

6) [10pts] Out of the "SOLID" principles, pick one of them, and explains how your team's design of the project so far adheres to said principle.

In our project design, we used **Liskov substitution** for all the enemy classes including grunts and bosses. The abstract class *Enemy* has the attributes of hp, hitbox and behaviors such as draw(), move(), fire(), isOutOfScreen(), which are common to both grunt and boss. Doesn't matter whether grunt or boss is received, the game only renders enemy objects by iterating on List<*Enemy*> and calling Enemy.draw() on each object.

7) [5pts] Pick one design pattern that you have used in the coding of the team project, and explain the pros (and cons, if any) of adopting said pattern; Or, if you have not (personally) implemented any patterns yet, pick one that you think might be a good idea to incorporate into your project eventually, and explain the hypothetical pros and cons in your own words.

We're using the factory pattern in our project for enemy creation.

Pros: We encapsulate the enemy creation implementation in a separate class. The type of class returned by factory is the abstract class rather than specific subclass because the game doesn't need to know what type of enemy it is.

Cons: We haven't run into any, but I would imagine in Milestone 3 when we have to import enemies by running config, we will have to modify or add another factory for config generated enemies.