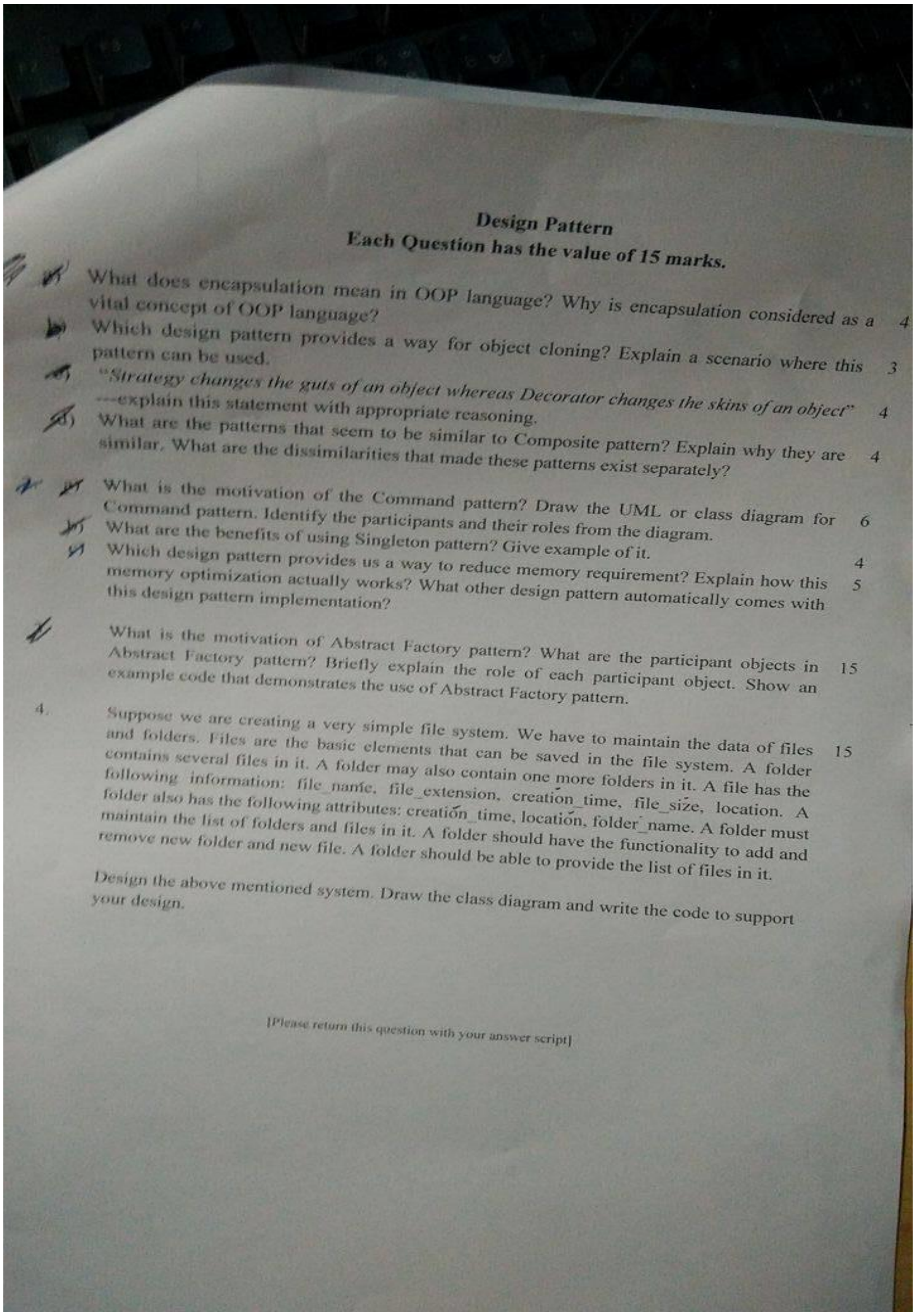
2017

Question



Design Pattern

Each Question has the value of 15 marks.

What does encapsulation mean in OOP language? Why is encapsulation considered as a vital concept of OOP language? 4

Which design pattern provides a way for object cloning? Explain a scenario where this pattern can be used. 3

"Strategy changes the guts of an object whereas Decorator changes the skins of an object" ---explain this statement with appropriate reasoning. 4

d) What are the patterns that seem to be similar to Composite pattern? Explain why they are similar. What are the dissimilarities that made these patterns exist separately? 4

What is the motivation of the Command pattern? Draw the UML or class diagram for Command pattern. Identify the participants and their roles from the diagram. 6

What are the benefits of using Singleton pattern? Give example of it. 4

Which design pattern provides us a way to reduce memory requirement? Explain how this memory optimization actually works? What other design pattern automatically comes with this design pattern implementation? 5

What is the motivation of Abstract Factory pattern? What are the participant objects in Abstract Factory pattern? Briefly explain the role of each participant object. Show an example code that demonstrates the use of Abstract Factory pattern. 15

4. Suppose we are creating a very simple file system. We have to maintain the data of files and folders. Files are the basic elements that can be saved in the file system. A folder contains several files in it. A folder may also contain one more folders in it. A file has the following information: file_name, file_extension, creation_time, file_size, location. A folder also has the following attributes: creation_time, location, folder_name. A folder must maintain the list of folders and files in it. A folder should have the functionality to add and remove new folder and new file. A folder should be able to provide the list of files in it. 15

Design the above mentioned system. Draw the class diagram and write the code to support your design.

[Please return this question with your answer script]

Answer

1.a

Encapsulation is an Object-Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding. Encapsulation is concerned with:

- Combining data and how it's manipulated in one place
- Only allowing the state of an object to be accessed and modified through behaviors:
- Hiding the details of how the object works

Why it is vital:

Benefits of encapsulation range from protection of intellectual property, ensuring other parts of the program don't change or access data that belongs to another part of the program, or to making a program more manageable and robust by separating parts of code.

Encapsulation helps in isolating implementation details from the behavior exposed to clients of a class (other classes/functions that are using this class) and gives you more control over coupling in your code.

1.b

Prototype Pattern.

Scenerio: Suppose we are doing a sales analysis on a set of data from a database. Normally, we would copy the information from the database, encapsulate it into an object and do the analysis. But if another analysis is needed on the same set of data, reading the database again and creating a new object is not the best idea. If we are using the Prototype pattern then the object used in the first analysis will be cloned and used for the other analysis.

The Client is here one of the methods that process an object that encapsulates information from the database. The ConcretePrototype classes will be classes that, from the object created after extracting data from the database, will copy it into objects used for analysis.

1.c

In Strategy Pattern, a class behavior or its algorithm can be changed at run time.

On the other hand, Decorator pattern attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub classing for extending functionality.

A decorator is likely to add functionality (decorate) an object and a strategy is likely to swap functionality.

The difference is that the object passed to the constructor is not being “wrapped” like a decorator. We’re “swapping” functionality in Strategy pattern.

The decorator is additive and the strategy is a replacement. So, strategy changes the guts but decorator changes the skins of an object.

1.d

Similar patterns like composite:

1. **Decorator: Decorator** is often used with Composite. When decorators and composites are

used together, they will usually have a common parent class. So decorators will

have to support the Component interface with operations like Add, Remove, and

GetChild.

2. **Flyweight: Flyweight** lets you share components, but they can no longer refer to their

parents.

3. **Iterator**: Iterator can be used to traverse composites.
4. **Chain of Responsibility**: Chain of Responsibility often the component-parent link is used for a Chain of Responsibility (251).

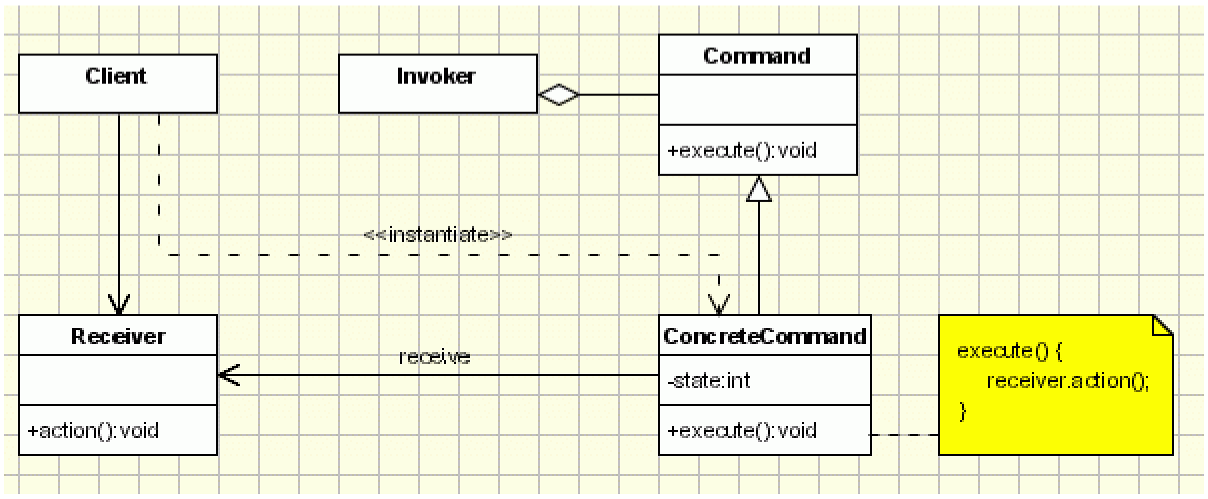
2.a

Command Pattern:

Motivation:

Sometimes it is necessary to issue requests to objects without knowing anything about the operation being requested or the receiver of the request." The Command design pattern suggests encapsulating ("wrapping") in an object all (or some) of the following: an object, a method name, and some arguments. Java does not support "pointers to methods", but its reflection capability will do nicely. The "command" is a black box to the "client". All the client does is call "execute()" on the opaque object

UML:



Participants and their role:

1. Command - declares an interface for executing an operation;
2. ConcreteCommand - extends the Command interface, implementing the Execute method by invoking the corresponding operations on Receiver. It defines a link between the Receiver and the action.
3. Client - creates a ConcreteCommand object and sets its receiver;
4. Invoker - asks the command to carry out the request;
5. Receiver - knows how to perform the operations;

<u>2.b</u>

Singleton Pattern:

Benefits:

1. Instance control: Singleton prevents other objects from instantiating their own copies of the Singleton object, ensuring that all objects access the single instance.
2. Flexibility: Since the class controls the instantiation process, the class has the flexibility to change the instantiation process.

Sometimes it is necessary to issue requests to objects without knowing anything about the operation b

Example:

The Singleton pattern is used in the design of logger classes. This classes are ussualy implemented as a singletons, and provides a global logging access point in all the application components without being necessary to create an object each time a logging operations is performed.

2.c

Flyweight pattern saves memory by sharing flyweight objects among clients. The amount of memory saved generally depends on the number of flyweight categories saved .

How this works?

The more flyweights are shared, the greater the storage savings. The savings increase with the amount of shared state. The greatest savings occur when the objects use substantial quantities of both intrinsic and extrinsic state, and the extrinsic state can be computed rather than stored. Then you save on storage in two ways: Sharing reduces the cost of intrinsic state, and you trade extrinsic state for computation time.

What other pattern comes with its implementation:

Factory and Singleton patterns - Flyweights are usually created using a factory and the singleton is applied to that factory so that for each type or category of flyweights a single instance is returned.

3.

Abstract Factory Pattern:

Motivation:

Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories.

Modularization is a big issue in today's programming. Programmers all over the world are trying to avoid the idea of adding code to existing classes in order to make them support encapsulating more general information. Take the case of an information manager which manages phone number. Phone numbers have a particular rule on which they get generated depending on areas and countries. If at some point the application should be changed in order to support adding numbers form a new country, the code of the application would have to be changed and it would become more and more complicated.

In order to prevent it, the Abstract Factory design pattern is used. Using this pattern, a framework is defined, which produces objects that follow a general pattern and at runtime this factory is paired with any concrete factory to produce objects that follow the pattern of a certain country. In other words, the Abstract Factory is a super-factory which creates other factories (Factory of factories).

Participants and their role:

- **AbstractFactory** - declares a interface for operations that create abstract products.
- **ConcreteFactory** - implements operations to create concrete products.

- **AbstractProduct** - declares an interface for a type of product objects.
- **Product** - defines a product to be created by the corresponding ConcreteFactory; it implements the AbstractProduct interface.
- **Client** - uses the interfaces declared by the AbstractFactory and AbstractProduct classes.

Code: https://www.tutorialspoint.com/design_pattern/abstract_factory_pattern.htm