QUESTION_1

why you chose those design patterns?

1. Prototype pattern

Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves implementing a prototype interface which tells to create a clone of the current object. This pattern is used when creation of object directly is costly. For example, an object is to be created after a costly database operation. We can cache the object, returns its clone on next request and update the database as and when needed thus reducing database calls.

Why we use it?

Let's consider the case in which we need to make a number of queries to an extensive database to construct an answer. Once we have this answer as a table or Result set, we might want to manipulate it to produce other answers without having to issue additional queries.

As such reason After we fetch (using swimmer data in our program) the data from database to be sorted in different context we don't need to fetch the same data for those tasks we just fetch the data for the first time then by cloning it we can manipulate our task as we need it.

2. Singleton pattern

Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

Why we use it?

As we know that singleton pattern has very benefit on common objects that can be used in each section of the system. If we look at Database Connection class, singleton pattern can be very suitable for this. Because we can manage Db Connection via one instance so we can control load balance, unnecessary connection, shared db connection management, control data inconsistency etc.

3. Facade pattern

Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.

Why we use it?

We use a facade design pattern to define a simplified interface to a more complex subsystem. The facade pattern is ideal when working with a large number of interdependent classes, or with classes that require the use of multiple methods, particularly when they are complicated to use or difficult to understand. And to add an effect to our programs appearance.