**[CIS 634 OBJECT ORIENTED SOFTWARE ENGINEERING SECTION 50](https://bb-csuohio.blackboard.com/webapps/blackboard/execute/courseMain?course_id=_129878_1" \o "Fall 20 (1) CIS 634 Object-Oriented Software Engr Section 50)**

**DESIGN PATTERNS & TESTING TOOL**

**QUIZ APPLICATION**

**Group 8-** Group Assignment Design Patterns

**Group members:**

1. **Bheemavarapu kavya Chowdary**
2. **Nuthakki Bhargav**
3. **Chakkala Sai Pujith**

**Observer Design Pattern**

In this pattern “Observer” an object, called “subject”, maintains a list of its dependencies, called “observers”, and automatically notifies them of any change of state, usually calling one of its methods.

**Singleton Design Pattern:**

The singleton pattern is a class that can have only one instance Singleton pattern is a design pattern which restricts a class to instantiate its multiple objects. It is nothing but a way of defining a class. Class is defined in such a way that only one instance of the class is created in the complete execution of a program or project. It is used where only a single instance of a class is required to control the action throughout the execution. A singleton class shouldn’t have multiple instances in any case and at any cost.

**For Quiz application**

This is a singleton application.

**Prototype Design Pattern:**

The prototype pattern is a creational design pattern. Prototype patterns is required, when object creation is time consuming, and costly operation, so we create object with existing object itself. One of the best available way to create object from existing objects are **clone() method**. Prototype allows us to hide the complexity of making new instances from the client.

**Decorator Design Pattern:**

Decorators provide a way of calling Higher-Order functions. The Higher-Order functions simply take a function, modify it and return a new function with added functionality. The key here is that they don’t modify the original function, they simply add some extra functionality which means they can be reused at multiple places. This special feature makes decorators so powerful.

**For Quiz application**

**Reason:**

Decorators isn't a widely implemented JS feature. It is still in its proposal stage. Babel 7 by default allows decorators as a default plugin in their stage 0 configuration. <https://babeljs.io/docs/plugins/transform-decorators/>

**Factory Design Pattern:**

Factory method is a [creational design pattern](https://www.geeksforgeeks.org/design-patterns-set-1-introduction/), i.e., related to object creation. In Factory pattern, we create object without exposing the creation logic to client and the client use the same common interface to create new type of object.

**For Quiz application**

**Reason:**

This is a small application , so there are no large components or any extra functionalities which needed to be separated in order to maintain code.

This functionality has to be implemented while writing the code at the starting point of application.

**Builder Design Pattern:**

Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

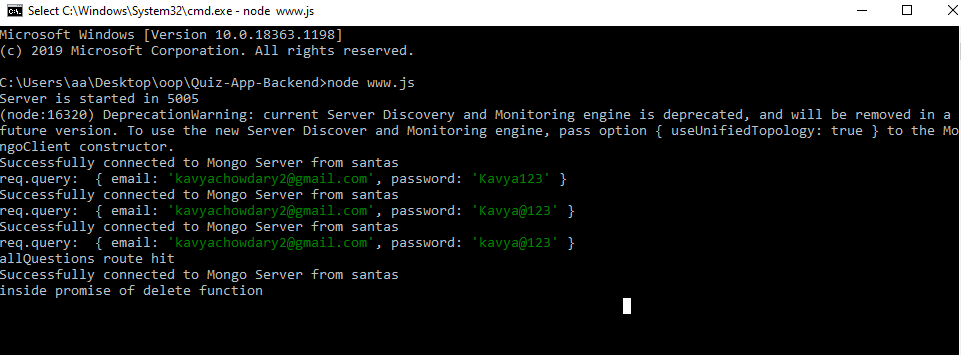
**For Quiz application**

**Reason:**

This should be used only when you want to build different immutable objects using same object building process. In quiz application we are not using any of such.

**For Quiz application**

**This is how it logs the data when user performs a operation**



**PARSER**

**ABOUT PROJECT:**

In this project the application has to read the data from a text file and parse the data. It has to calculate total number of hours from the parsed data and return the output.

The text file contains date ,time spent on the day and the work done.

So this program will read each and every line from the text file ,if time parttern is available it converts into 24 hrs time format then calculates the time spent on that particular line and stores it. When it reads the next line it repeats the same operation as a above and adds it to the stored time. After parsing the last line of the file it returns the total time.

This Application is a Springboot application. The Program is written in java.The testing tool used is Junit.

In this application we have implemented **Singleton Design Pattern** and **Decorator Design Pattern.**

**Singleton Design Pattern:**

This class is the reason for the complete execution of a program.

**@SpringBootApplication**

**public class Assignment1Application extends SpringBootServletInitializer {**

**public static void main(String[] args) {**

**SpringApplication.run(Assignment1Application.class, args); }}**

**Decorator Design Pattern:**

These are the decorators which have been used in this application

**@SpringBootApplication**

**@Service**

**@RestController**

**@Autowired**

**@GetMapping**

**TESTING:**

**JUnit**

JUnit is a unit testing framework for Java programming language.

Testing is the process of checking the functionality of an application to ensure it runs as per requirements.

JUnit tests can be run automatically and they check their own results and provide immediate feedback.

**Adding dependencies to POM :**

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId

**Test XMLfile:**

[<testsuite failures="**0**" skipped="**0**" errors="**0**" tests="**1**" time="**2.105**" name="**com.example.demo.Assignment1ApplicationTests**"](file:///C:\Users\aa\Downloads\TEST-com.example.demo.Assignment1ApplicationTests.xml)

**Expected output**

**Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated**

**Actual output**

