**Software Design Final Assignment 2020**

**Patterns Essay**

**by**

**Varun Kumar Tiwari**

**(A00268806)**

**Degree of Bachelor of Engineering with Honors**

**in SOFTWARE ENGINEERING**

**ATHLONE INSTITUTE OF TECHNOLOGY**



**Background**

The concept of patterns (in general) was originally articulated by Christopher Alexander and colleagues in the late 1970s [The Timeless Way of Building, 1979; A Pattern Language—Towns, Buildings, Construction, 1977] (They had 253 patterns.) [1]

**Design patterns in java**

Design patterns are used to give developers with templates on how we can solve problem in software development without repeating the mistake every time. For Java developers is crucial to understand design patterns and know where and when to use them. When one has mastered this skill, it can become an amazing productivity multiplier.

**The two main use of design patterns:**

* Common language for developers: The patterns provides a common language for the developers to use in certain problems. For example, if a developer tells another developer that I am using Singleton, the other developer should know what it means.
* Capture good practices: Design patterns store solutions that have been proven successful and have been applied to solve issues. So, if one learns these patterns and the related problem, even an unexperienced developer can know much about software design.

**Object orientated design principle is the base of Design pattern.**

* Program to an interface and not an implementation
* Give importance to object composition over inheritance

## Design pattern categories

There are 23 patterns which are categorized under these 3 groups:

**1. Creational patterns** [2]**:** These patterns focus on creation of classes and inheritance. These patterns can also consist patterns related to the object creation.

**2. Structural patterns** [2]**:** These patterns are a display of how the classes are composed; composition of objects and classes which will lead to creating interfaces.

**3. Behavioral patterns** [2]**:** These patterns show the way in which objects interact with each other within the software.

|  |  |  |
| --- | --- | --- |
| **Creational patterns** | **Structural patterns** | **Behavioral patterns** |
| Builder  Factory  Prototype  Singleton  Abstract | Bridge adapter  Decorator Composite  Flyweight Façade  proxy | Chain of responsibility  Command Interpreter  Iterator Mediator  Memento Strategy  Template method Visitor |

Decorator Pattern Description

The **decorator design pattern** allowed me to add the functionality and behavior dynamically to an object. This was done without having any effect on the behavior of other existing objects present in the same class.

Here inheritance is used to extend the behavior of the class. This process takes place during compile time, and all the instances of the class then gets the extended behavior.

the decorator pattern helps us to add functionality to an object (not the class) while runtime, and we can apply this customized functionality to an individual object based on our requirement and choice.

In my project I have used this pattern. It has some very nice advantages such as if I want to extend this project further more, I won’t have to change the previously written functions. Instead I can add as many functions as I want

Summary

The Decorator Pattern is used for adding additional functionality to a particular object as opposed to a class of objects. It is easy to add functionality to an entire class of objects by subclassing an object, but it is impossible to extend a single object this way. With the Decorator Pattern, you can add functionality to a single object and leave others like it unmodified.

**Key use cases:** [3]

1. Add additional functionalities/responsibilities dynamically
2. Remove functionalities/responsibilities dynamically
3. Avoid too much of sub-classing to add additional responsibilities.

**Drawbacks:** [3]

### Overuse of Open Closed principle (Open for extension and Closed for modification). Use this feature sparingly where the code is least likely changed.

1. Too many small classes and will add maintenance overhead.

# References

|  |  |
| --- | --- |
| [1] | C. Alexander, "The Timeless Way of Building," 1979. [Online]. Available: https://en.wikipedia.org/wiki/The\_Timeless\_Way\_of\_Building. |
| [2] | G. T. Sachin Araballi, "Design Patterns | Set 1," 2015. [Online]. Available: geeksforgeeks.org/design-patterns-set-1-introduction/. |
| [3] | A. Mandliya, "Decorator Design Pattern in Java," 11 Feb 2013. [Online]. Available: https://www.codeproject.com/Tips/468951/Decorator-Design-Pattern-in-Java. |

# STUDENT PLAGIARISM DISCLAIMER FORM

****

## PLAGIARISM DISCLAIMER

STUDENT NAME: Varun Kumar Tiwari

STUDENT NUMBER: A00268806

PROGRAMME: Bachelor of Engineering in Software Engineering

YEAR: Year 4

MODULE: Software Design

LECTURER: Declan Byrne

ASSIGNMENT TITLE:

DUE DATE: 21/4/2020

DATE SUBMITTED: 21/4/2020

ADDITIONAL INFORMATION:

I understand that plagiarism is a serious academic offence, and that AIT deals with it according to the AIT Policy on Plagiarism.

I have read and understand the AIT Policy on Plagiarism and I agree to the requirements set out therein in relation to plagiarism and referencing. I confirm that I have referenced and acknowledged properly all sources used in preparation of this assignment. I understand that if I plagiarise, or if I assist others in doing so, that I will be subject to investigation as outlined in the AIT Policy on Plagiarism.

I understand and agree that plagiarism detection software may be used on my assignment. I declare that, except where appropriately referenced, this assignment is entirely my own work based on my personal study/or research. I further declare that I have not engaged the services of another to either assist in, or complete this assignment.

## Signed: Varun Kumar Tiwari

**Dated: 27/4/2020**