**SKILL LEARNT: DESIGN PRINCIPLES AND PATTERN (WEEK 1)**

**EXERCISE 1: IMPLEMENTING THE SINGLETON PATTERN**

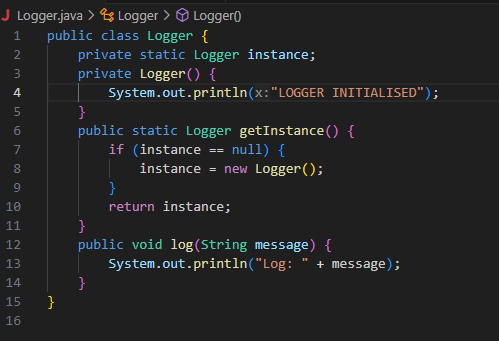
In this exercise, I used the Singleton Design Pattern to create a logging utility class that makes sure that only one instance of the Logger class exists throughout the application's lifecycle. This pattern is useful when just one object is needed to manage actions globally.

To do this, I set up a new Java project called SingletonPatternExample and created a Logger class. It has a private static instance, a private constructor, and a public static method called getInstance() that manages access to the single instance. This way, any part of the application that calls the Logger will get the same instance, ensuring consistent and centralized logging.

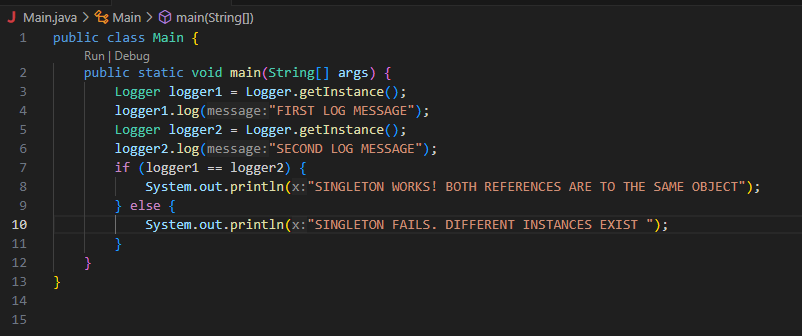
To test the implementation, I developed a separate Main class that accesses the Logger from different points. The output showed that the constructor was called only once, and both references pointed to the same Logger object. This successfully demonstrated the Singleton behavior.

The codes for the Logger.java and Main.java are shown below:

1. **LOGGER.JAVA**



1. **MAIN.JAVA**



**OUTPUT:**

1. **COMPILATION OF THE JAVA FILES**

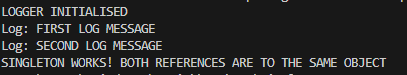


After compilation Logger.class and Main.class files are created.

1. **RUN**



1. **OUTPUT**



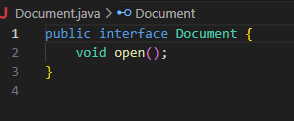
Through this exercise, I learned how to implement the Singleton Design Pattern in Java, ensuring that a class has only one instance throughout the application. I understood the importance of using a private constructor, a static instance, and a public accessor method to control object creation. I also learned how Singleton helps maintain consistent behavior, especially in scenarios like logging, where shared access to a single resource is crucial.

**EXERCISE 2: IMPLEMENTING THE FACTORY METHOD PATTERN**

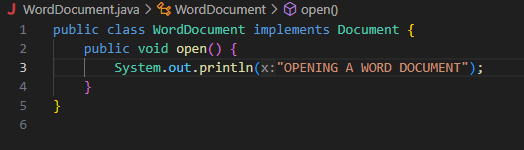
In this exercise, I used the Factory Method Pattern to design a document management system that can create different types of documents like Word, PDF, and Excel. First, I defined a common Document interface. Then, I created concrete classes for each document type. I also developed an abstract DocumentFactory with a factory method called createDocument(), and I implemented separate concrete factory classes for each document variant. Finally, I tested the system by dynamically creating and opening various document types. This showed how the factory method offers flexibility and scalability in creating objects.

There are 9 files under FactoryMethodPatternExample.

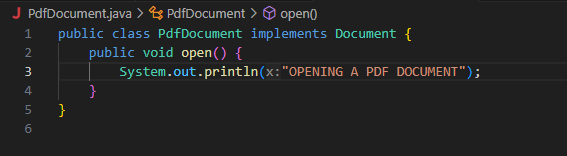
1. **DOCUMENT.JAVA**



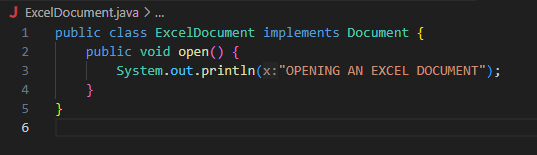
1. **WORDDOCUMENT.JAVA**



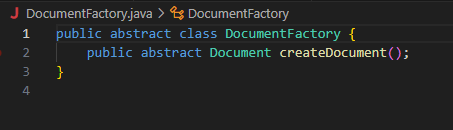
1. **PDFDOCUMENT.JAVA**



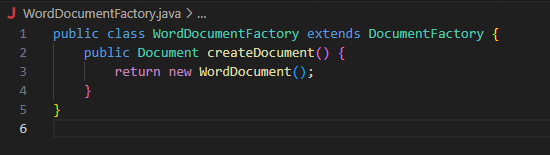
1. **EXCELDOCUMENT.JAVA**



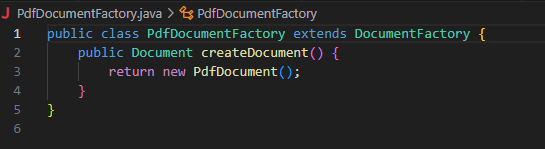
1. **DOCUMENTFACTORY.JAVA**



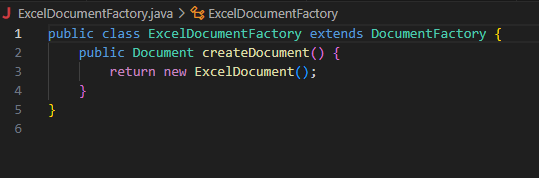
1. **WORDDOCUMENTFACTORY.JAVA**



1. **PDFDOCUMENTFACTORY.JAVA**



1. **EXCELDOCUMENTFACTORY.JAVA**



1. **MAIN.JAVA**



**OUTPUT**

1. **COMPILATION OF JAVA FILES**



1. **RUN**



1. **OUTPUT**



Through this exercise, I learned how to apply the Factory Method Pattern to create objects in a flexible and scalable way. It helped me understand the importance of decoupling object creation from the client code, making the system easier to maintain and extend when new document types are added.

**QUIZZES**

# What is a design pattern in software development?

# Answer: A general reusable solution to a commonly occurring problem.

# How many types of design patterns are there?

# Answer: 3.

# What is the main benefit of using a design pattern?

# Answer: All of the above.

# Which of the following is NOT a creational design pattern?

# Answer: Bridge pattern.

# Which of the following is a behavioral design pattern?

# Answer: Observer Pattern.

# What is a Singleton design pattern?

# Answer: A design pattern that ensures that a class has only one instance.

# Which of the following best describes the Factory Pattern in design patterns?

# Answer: It allows an object to be created without exposing the creation logic to the client and the created object is referred to using a common interface.

# What is the use of the Builder Pattern?

# Answer: It simplifies the creation of complex objects by breaking the creation process into steps.

# What is the primary purpose of the Abstract Factory design pattern in software design?

# Answer: It provides an interface for creating families of related or dependent objects without specifying their concrete classes.

# What is the primary purpose of the Prototype Pattern in design patterns?

# Answer: To create a clone of an existing object rather than creating a new one, typically for performance reasons.

# What is the primary function of the Adapter design pattern in software development?

# Answer: It allows classes with incompatible interfaces to work together by wrapping its own interface around that of an already existing class.

# Which pattern helps in reducing complex conditional logic?

# Answer: Strategy Pattern.

# The Decorator design pattern is also known as:

# Answer: Wrapper.

# Which structural pattern should be used when you want to add responsibilities to an object dynamically?

# Answer: Decorator.

# Which of the following best describes the Composite design pattern?

# Answer: The Composite Pattern composes objects into tree structures to represent part-whole hierarchies.

# Which design pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation?

# Answer: Iterator Pattern.

# Which pattern is specifically concerned with communication between objects?

# Answer: Mediator.

# In which pattern does a surrogate or placeholder class control access to the original object?

# Answer: Proxy.

# Which structural design pattern decouples an abstraction from its implementation so that the two can vary independently?

# Answer: Bridge.

# Which pattern allows an object to change its behavior when its internal state changes?

# Answer: State.

# Which behavioral design pattern should be used when a behavior among objects should be encapsulated and made to operate on an object structure?

# Answer: Visitor.

# Which behavioral design pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable?

# Answer: Strategy.

# Which design pattern is used when there is a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically?

# Answer: Observer.

# Which design pattern would you use to encapsulate a request as an object?

# Answer: Command Pattern.

# What is Template Design Pattern?

# Answer: It defines the skeleton of an algorithm in a method, deferring some steps to subclasses.