**Design principles & Patterns**

**Week1-Assignments**

* **Design Patterns and Principles:-**

Exercise 1: Implementing the Singleton Pattern

**Code:-**

// Singleton Pattern Example

class Singleton {

// static instance

private static Singleton instance;

// private constructor

private Singleton() {

System.out.println("Singleton instance created");

}

// static method to get the instance

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

public void showMessage() {

System.out.println("Hello from Singleton!");

}

}

public class SingletonDemo {

public static void main(String[] args) {

// get the single instance

Singleton obj1 = Singleton.getInstance();

Singleton obj2 = Singleton.getInstance();

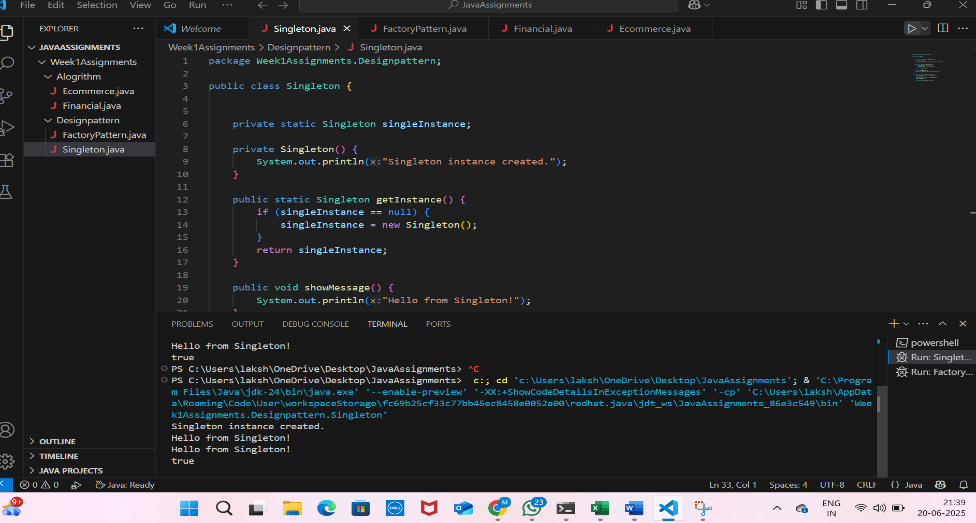
obj1.showMessage();

// Check if both references are same

System.out.println("Are both objects same? " + (obj1 == obj2));

}

}

**Output:-**

Exercise 2: Implementing the Factory Method Pattern

Code:

class ConsoleLogger implements Logger {

    @Override

    public void log(String message) {

        System.out.println("Console Log: " + message);

    }

}

// Concrete Product 2: File Logger

class FileLogger implements Logger {

    @Override

    public void log(String message) {

        System.out.println("File Log: " + message + " (logged to file)");

        // In a real application, you would write to a file here

    }

}

// Concrete Product 3: Database Logger

class DatabaseLogger implements Logger {

    @Override

    public void log(String message) {

        System.out.println("Database Log: " + message + " (logged to database)");

        // In a real application, you would insert into a database here

    }

}

// Creator Abstract Class

abstract class LoggerFactory {

    // The factory method (abstract)

    public abstract Logger createLogger();

    // An operation that uses the created logger

    public void logMessage(String message) {

        Logger logger = createLogger(); // Call the factory method

        logger.log(message);

    }

}

// Concrete Creator 1: Console LoggerFactory

class ConsoleLoggerFactory extends LoggerFactory {

    @Override

    public Logger createLogger() {

        return new ConsoleLogger();

    }

}

// Concrete Creator 2: File LoggerFactory

class FileLoggerFactory extends LoggerFactory {

    @Override

    public Logger createLogger() {

        return new FileLogger();

    }

}

// Concrete Creator 3: Database LoggerFactory

class DatabaseLoggerFactory extends LoggerFactory {

    @Override

    public Logger createLogger() {

        return new DatabaseLogger();

    }

}

// Client Usage

public class FactoryPattern {

    public static void main(String[] args) {

        // Use Console Logger

        LoggerFactory consoleFactory = new ConsoleLoggerFactory();

        consoleFactory.logMessage("This is a console message.");

        System.out.println("--------------------");

        // Use File Logger

        LoggerFactory fileFactory = new FileLoggerFactory();

        fileFactory.logMessage("This is a file message.");

        System.out.println("--------------------");

        // Use Database Logger

        LoggerFactory databaseFactory = new DatabaseLoggerFactory();

        databaseFactory.logMessage("This is a database message.");

    }

}

// Use File Logger

        LoggerFactory fileFactory = new FileLoggerFactory();

        fileFactory.logMessage("This is a file message.");

        System.out.println("--------------------");

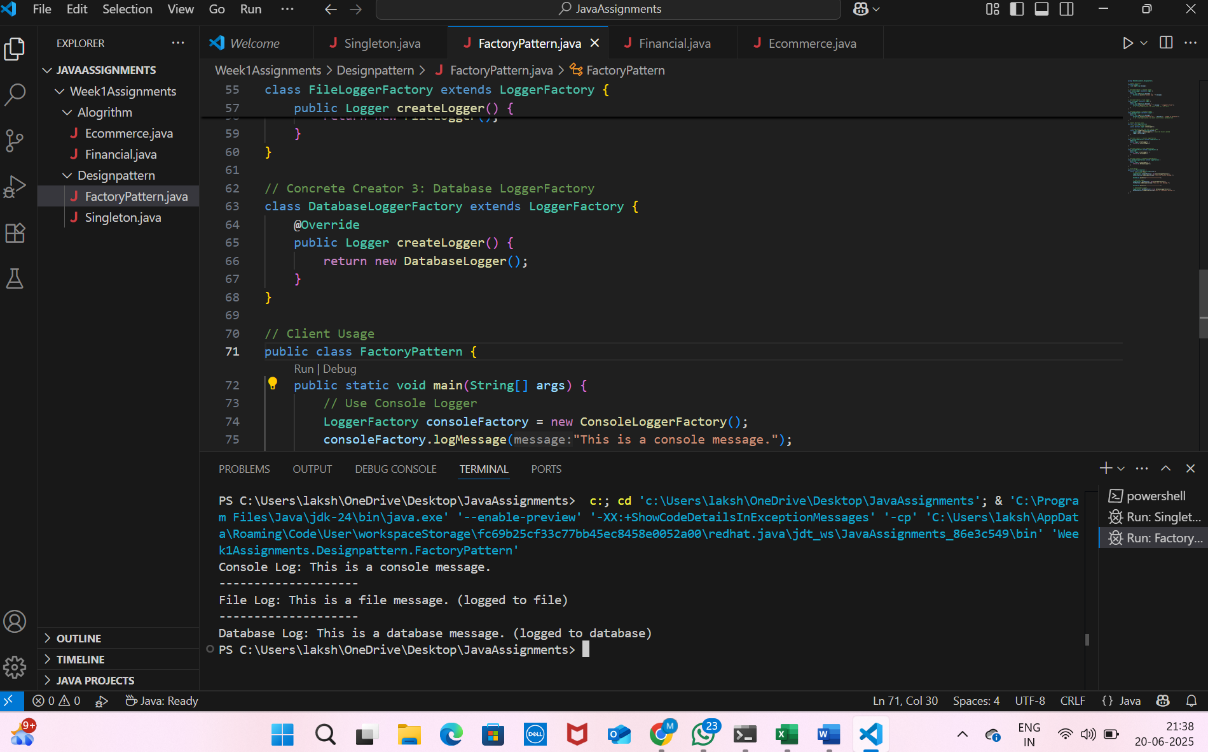
        // Use Database Logger

        LoggerFactory databaseFactory = new DatabaseLoggerFactory();

        databaseFactory.logMessage("This is a database message.");

    }

}

**Output:**