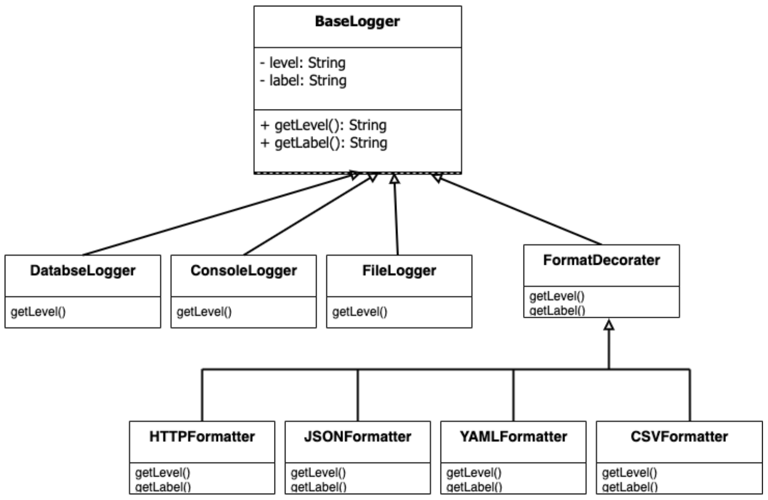
DESIGN PATTERNS - CPIT252

LAB_6

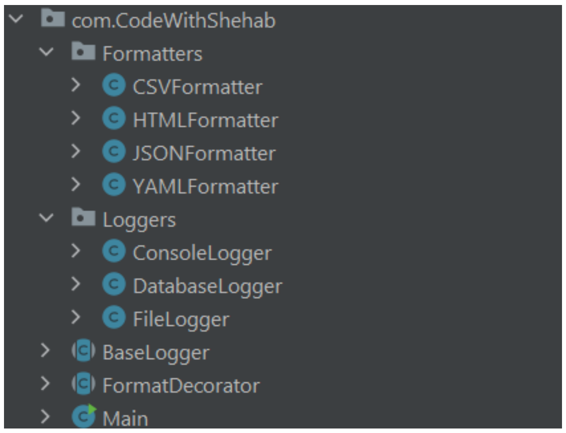
Shehab Al Harithi - 1846003

CONTENTS

DISTRIBUTION OF THE PROJECT



Based on the UML provided on the following website https://cpit252.gitlab.io/labs/lab-6/ the project was distributed in as it shown just to look more comfortable:



```
com.CodeWithShehab
    Formatters
        CSVFormatter
        HTMLFormatter
        JSONFormatter
        YAMLFormatter
    Loggers
        ConsoleLogger
        DatabaseLogger
        FileLogger
    BaseLogger
    FormatDecorator
    Main
```

FORMATTERS PACKAGE

CSVFORMATTER CLASS

```java
import com.CodeWithShehab.BaseLogger;
import com.CodeWithShehab.FormatDecorator;

public class CSVFormatter extends FormatDecorator {
    public CSVFormatter(BaseLogger logger) {
        this.logger = logger;
    }

    @Override
    public String getLevel() {
        return "Info " + logger.getLevel();
    }

    @Override
    public String getLabel() {
        return logger.getLabel() + ", CSVFormatter";
    }
}
```

YAMLFORMATTER CLASS (ADDED)

```java
import com.CodeWithShehab.BaseLogger;
import com.CodeWithShehab.FormatDecorator;

public class YAMLFormatter extends FormatDecorator {
    public YAMLFormatter(BaseLogger logger) {
        this.logger = logger;
    }

    @Override
    public String getLevel() {
        return "info " + logger.getLevel();
    }

    @Override
    public String getLabel() {
        return logger.getLabel() + ", YAMALFormatter";
    }
}
```

JSONFORMATTER CLASS (ADDED)

```java
import com.CodeWithShehab.BaseLogger;
import com.CodeWithShehab.FormatDecorator;

public class JSONFormatter extends FormatDecorator {
    public JSONFormatter(BaseLogger logger) {
        this.logger = logger;
    }

    @Override
    public String getLevel() {
        return "info " + logger.getLevel();
    }

    @Override
    public String getLabel() {
        return logger.getLabel() + ", JSONFormatter";
    }
}
```

HTMLFORMATTER CLASS (ADDED)

```java
import com.CodeWithShehab.BaseLogger;
import com.CodeWithShehab.FormatDecorator;

public class HTMLFormatter extends FormatDecorator {
    public HTMLFormatter(BaseLogger logger) {
        this.logger = logger;
    }

    @Override
    public String getLevel() {
        return "Info " + logger.getLevel();
    }

    @Override
    public String getLabel() {
        return logger.getLabel() + ", HTMLFormatter";
    }
}
```

LOGGERS PACKAGE

CONSOLELOGGER CLASS

```java
import com.CodeWithShehab.BaseLogger;

public class ConsoleLogger extends BaseLogger {
    public ConsoleLogger() {
        label = "Console logger";
    }

    @Override
    public String getLevel() {
        return "info";
    }
}
```

DATABASELOGGER CLASS (ADDED)

```java
import com.CodeWithShehab.BaseLogger;

public class DatabaseLogger extends BaseLogger {
    public DatabaseLogger() {
        label = "database logger";
    }

    @Override
    public String getLevel(){
        return "error";
    }
}
```

FILELOGGER CLASS (ADDED)

```
public class FileLogger extends BaseLogger {
    public FileLogger() {
        label = "File logger";
    }

    @Override
    public String getLevel() {
        return "debug";
    }
}
```

ABSTRACT CLASSES

FORMATDECORATOR CLASS

```
public abstract class FormatDecorator extends BaseLogger {
     protected BaseLogger logger;

    // All format decorators have to reimplement the getLabel method
    public abstract String getLabel();
}
```

BASELOGGER CLASS

```
public abstract class BaseLogger {
    protected String label = "Unknown label";

    public String getLabel() {
        return label;
    }

    public abstract String getLevel();
}
```

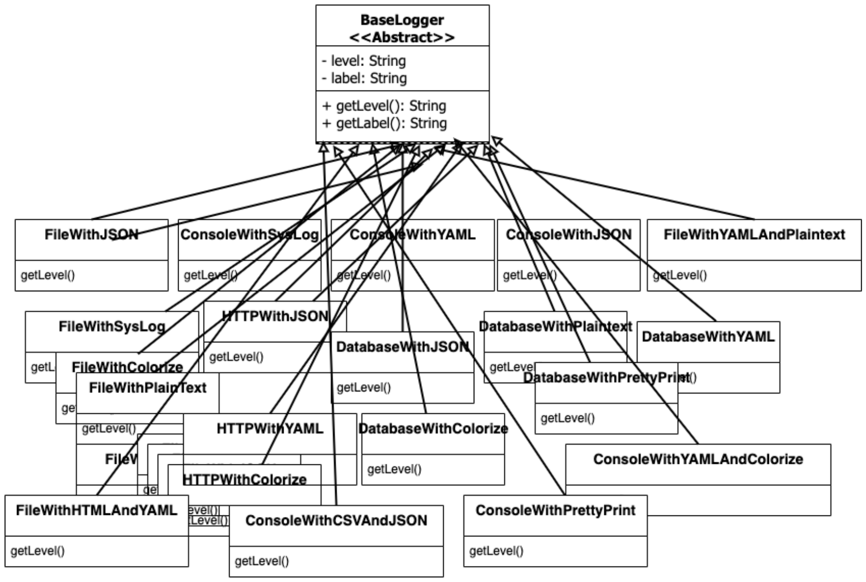
MAIN CLASS(MODIFIED)

```java
import com.CodeWithShehab.Formatters.CSVFormatter;
import com.CodeWithShehab.Formatters.HTMLFormatter;
import com.CodeWithShehab.Formatters.JSONFormatter;
import com.CodeWithShehab.Formatters.YAMLFormatter;
import com.CodeWithShehab.Loggers.ConsoleLogger;
import com.CodeWithShehab.Loggers.DatabaseLogger;
import com.CodeWithShehab.Loggers.FileLogger;

public class Main {

    public static void main(String[] args) {
        BaseLogger logger = new FileLogger();
        System.out.println(logger.getLabel()
                + ". Level: " + logger.getLevel());

        // create a console logger
        BaseLogger logger2 = new ConsoleLogger();
        // decorate it with a CSV and HTML formatters
        logger2 = new CSVFormatter(logger2);
        logger2 = new HTMLFormatter(logger2);
        System.out.println(logger2.getLabel()
                + ". Level: " + logger2.getLevel());

        // create a file logger
        BaseLogger logger3 = new FileLogger();
        // decorate it with a JSON, CSV, and YAML formatters
        logger3 = new JSONFormatter(logger3);
        logger3 = new CSVFormatter(logger3);
        logger3 = new YAMLFormatter(logger3);
        System.out.println(logger3.getLabel()
                + ". Level: " + logger3.getLevel());

        // create a database logger
        BaseLogger logger4 = new DatabaseLogger();
        // decorate it with a JSON formatter
        logger3 = new JSONFormatter(logger3);
        System.out.println(logger4.getLabel()
                + ". Level: " + logger4.getLevel());

    }
```

Part in red rectangle has been added just so database logger can be used because it wasn't used in the original main.

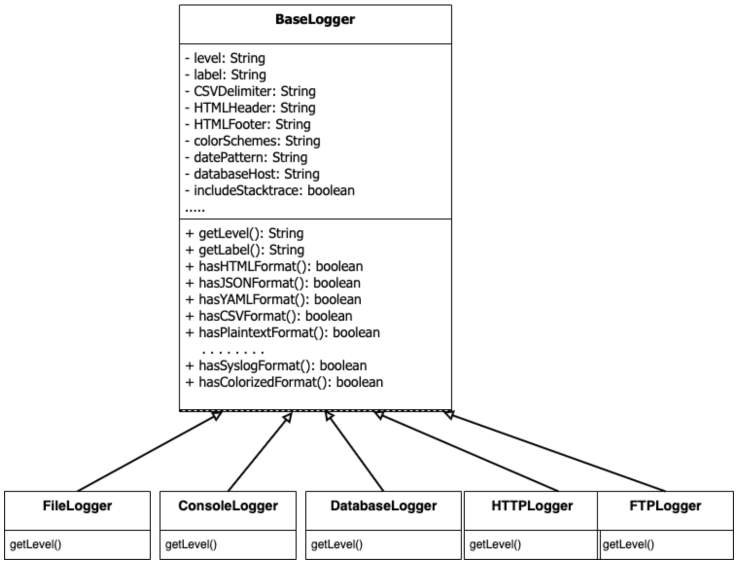
EXPLAIN HOW THE DECORATOR DESIGN PATTERN IS A BETTER ALTERNATIVE TO THE PREVIOUS TWO ATTEMPTS?

FIRST SOLUTION

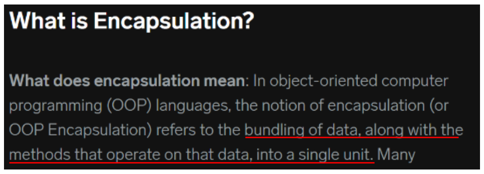


We can clearly see that the design is so bad since we are building a separated class for every format we are trying to use. So, if one day we added new format or new logger this may cause us creating more and more classes.

SECOND SOLUTION



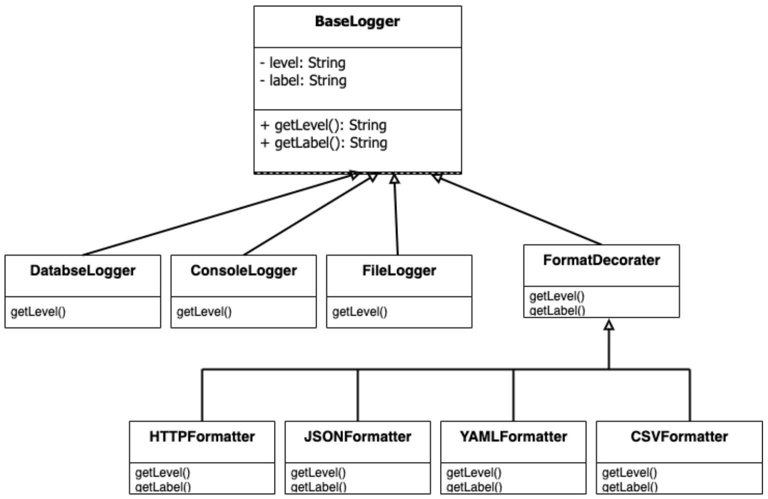
The problem with this solution is that each logger is having methods that its not using and not even needing and this breaks the **encapsulation** concept plus the project size will be bigger since each class is inheriting a lot of unnecessary methods which do increase the size of the class on disk.



What is Encapsulation?

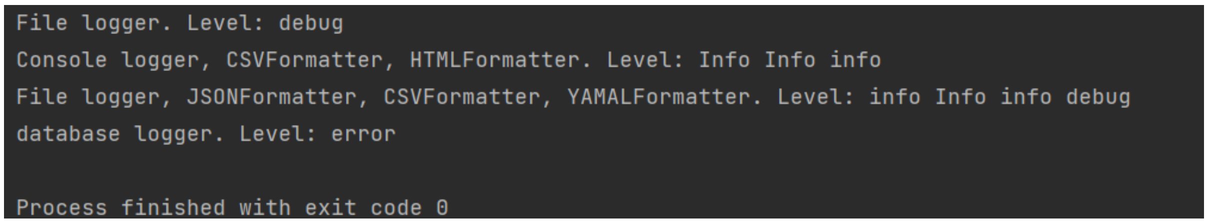
What does encapsulation mean: In object-oriented computer programming (OOP) languages, the notion of encapsulation (or OOP Encapsulation) refers to the bundling of data, along with the methods that operate on that data, into a single unit. Many

DECORATOR SOLUTION



As we can see from the UML no waste of memory no duplication of classes nothing is wasted. If we ever needed to add a new logger, we can just make it implement the **BaseLogger** class methods and so as for the formats we can just make it implement the **FormatDecorater.**

OUTPUT



```
File logger. Level: debug
Console logger, CSVFormatter, HTMLFormatter. Level: Info Info info
File logger, JSONFormatter, CSVFormatter, YAMALFormatter. Level: info Info info debug
database logger. Level: error

Process finished with exit code 0
```