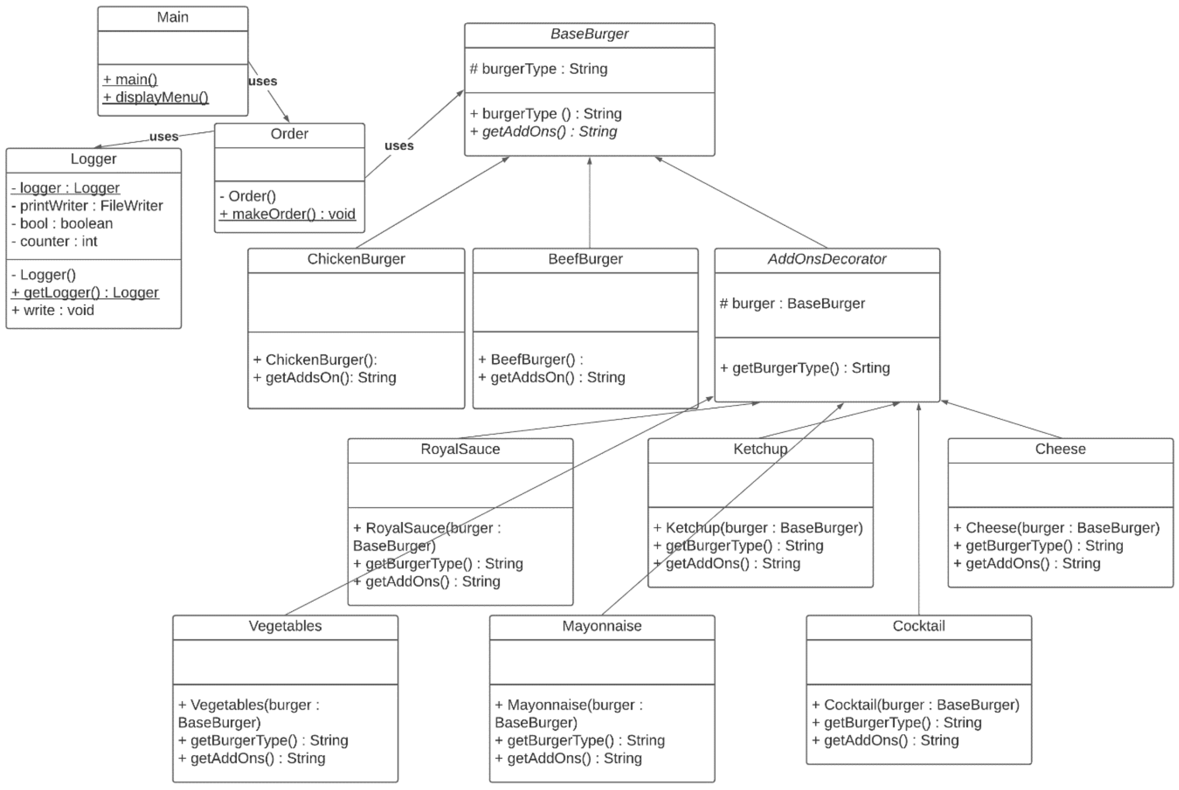
DESIGN PATTERNS - CPIT252

Final Project (Royal Resturant)

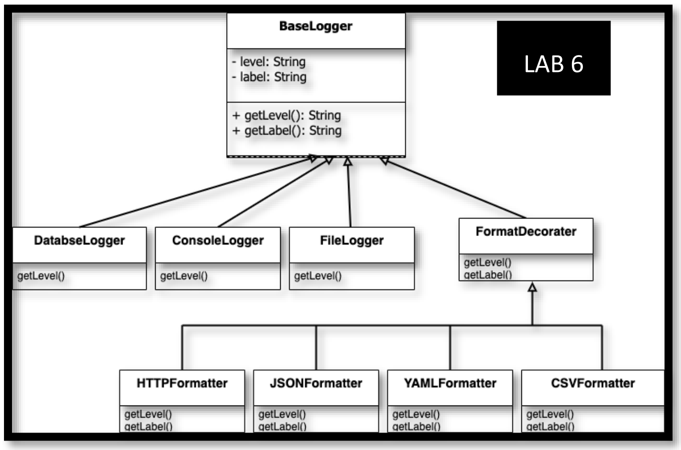
Shehab

CONTENTS

UML & DESCRIPTION

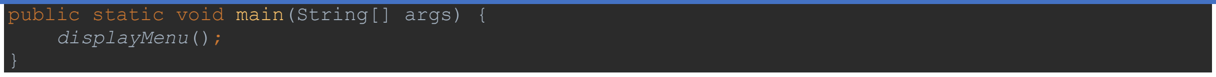


As we can see from the diagram, the three types of design patterns have been covered in a way where logger class has covered the creational part “**Singleton**”, and “**Decorator**” has covered the structural part. (Lab 6 gave us a hint on that).



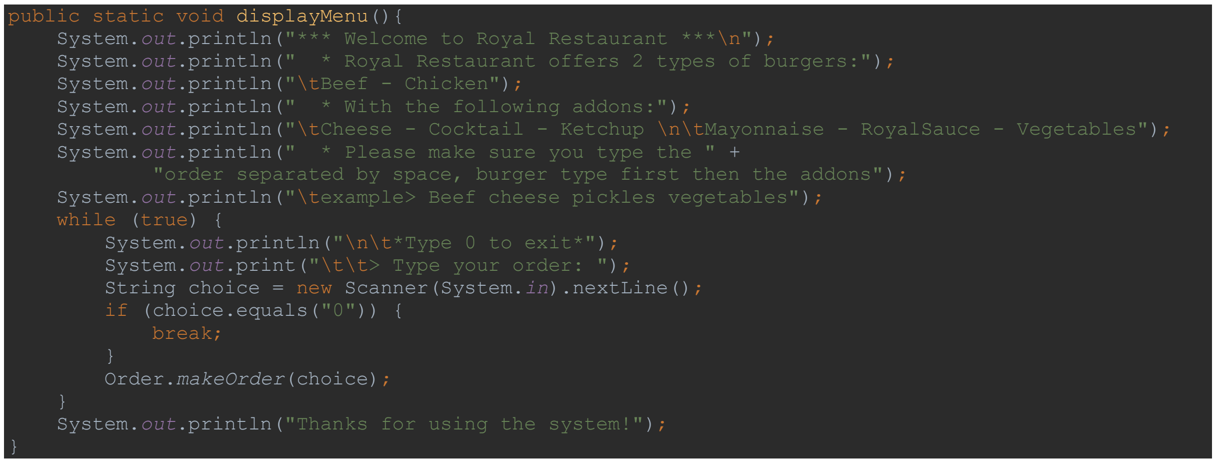
And in the end, “**Interpreter**” covering the behavioral part where “**Order**” class use what users enters as string converting it to code.

DATAFLOW



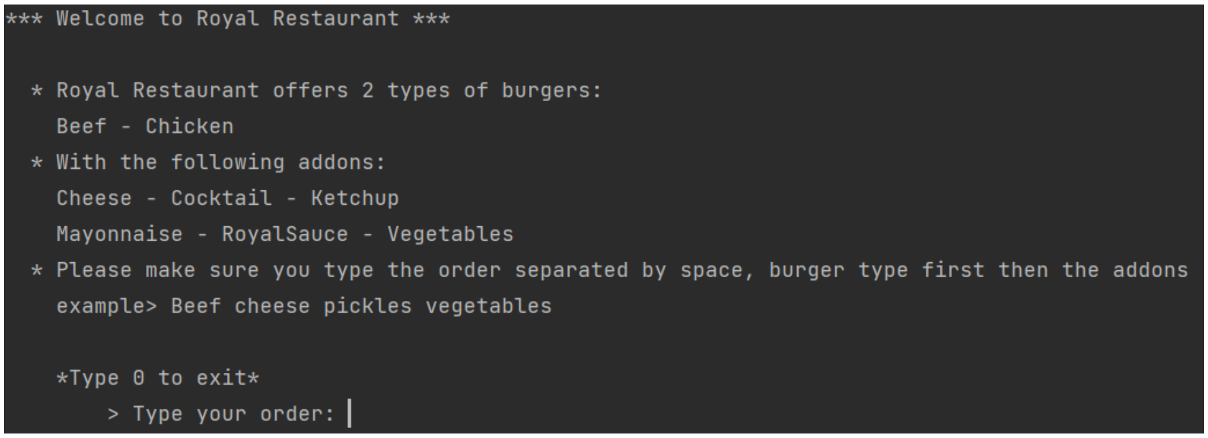
```
public static void main(String[] args) {
    displayMenu();
}
```

First, main() will call **displayMenu()** method to display the menu…



```
public static void displayMenu(){
    System.out.println("*** Welcome to Royal Restaurant ***\n");
    System.out.println("  * Royal Restaurant offers 2 types of burgers:");
    System.out.println("\tBeef - Chicken");
    System.out.println("  * With the following addons:");
    System.out.println("\tCheese - Cocktail - Ketchup \n\tMayonnaise - RoyalSauce - Vegetables");
    System.out.println("  * Please make sure you type the " +
            "order separated by space, burger type first then the addons");
    System.out.println("\texample> Beef cheese pickles vegetables");
    while (true) {
        System.out.println("\n\t*Type 0 to exit*");
        System.out.print("\t\t> Type your order: ");
        String choice = new Scanner(System.in).nextLine();
        if (choice.equals("0")) {
            break;
        }
        Order.makeOrder(choice);
    }
    System.out.println("Thanks for using the system!");
}
```

displaying the following menu…

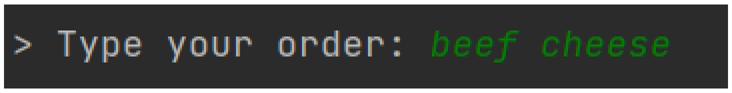


```
*** Welcome to Royal Restaurant ***

  * Royal Restaurant offers 2 types of burgers:
    Beef - Chicken
  * With the following addons:
    Cheese - Cocktail - Ketchup
    Mayonnaise - RoyalSauce - Vegetables
  * Please make sure you type the order separated by space, burger type first then the addons
    example> Beef cheese pickles vegetables

    *Type 0 to exit*
        > Type your order: 
```

User will be asked to type his order based on burger type and addons that has been provided with an example where the burger type must be mentioned first. <u>(will discuss why when we get to the addson part)</u>. if the user enters 0 program will terminate, otherwise his order will be created.

<u>Note: We assumed that user won't enter invalid input, so there is no error handling in the project.</u>

For example, user has entered…



```
> Type your order: beef cheese
```

```java
public static void makeOrder(String str) {
    Logger logger = Logger.getInstance();
    BaseBurger burger = null;

    String[] list = str.split(" ");

    for (int i = 0; i < list.length; i++) {
        switch (list[i].toLowerCase()) {
            case "cheese":
                burger = new Cheese(burger);
                break;
            case "chicken":
                burger = new ChickenBurger();
                break;
            case "beef":
                burger = new BeefBurger();
                break;
            case "cocktail":
                burger = new Cocktail(burger);
                break;
            case "ketchup":
                burger = new Ketchup(burger);
                break;
            case "mayonnaise":
                burger = new Mayonnaise(burger);
                break;
            case "royalsauce":
                burger = new RoyalSauce(burger);
                break;
            case "vegetables":
                burger = new Vegetables(burger);
                break;
        }
    }

    System.out.println("\n\tBurger type is: " + burger.getBurgerType() + ". \n\tAddons are    : " +
burger.getAddOns());
    logger.write("\n\tBurger type is: " + burger.getBurgerType() + ". \n\tAddons are    :" +
burger.getAddOns());
}
```

so, **str = "beef burger"**. a logger object will be created then brought; <u>logger will be presented that in a bit</u>. A "**BaseBurger**" object will be declared left null for now. **str** will be split in an array where <u>space</u> is the regular expression. For loop will iterate the size of the array times.

For our input **str** it'll iterate only two times first time the burger type will be specified…

BaseBurger Abstract Class

```java
public abstract class BaseBurger {
    protected String burgerType = "Unknown
burger";

    public String getBurgerType() {
        return burgerType;
    }

    public abstract String getAddOns();
}
```

BeefBurger Class extending BaseBurger

```java
public class BeefBurger extends BaseBurger {
    public BeefBurger() {
        super.burgerType = "Beef burger";
    }

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

As we can see **BeefBurger's** constructor will set the burger type to beef, and the addons so far are empty for now let's see how the addons will be added…

As the second iteration goes after declaring the type of the burger, the same object will be passed in the **Cheese's** constructor, check the following table…

BaseBurger Abstract Class

```
public abstract class BaseBurger {
    protected String burgerType =
"Unknown burger";

    public String getBurgerType() {
        return burgerType;
    }

    public abstract String
getAddOns();
}
```

AddOnsDecorator Abstract Class Extending BaseBurger

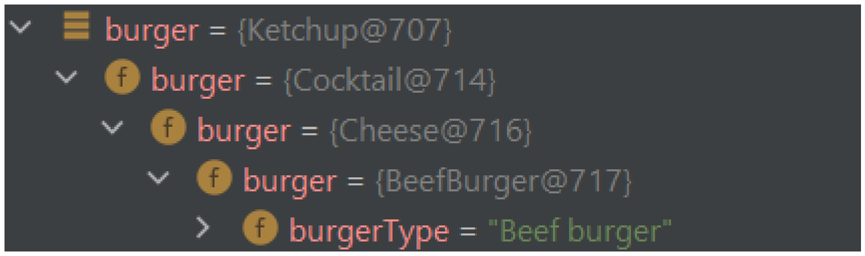
```
public abstract class AddOnsDecorator
extends BaseBurger{
    protected BaseBurger burger;

    public abstract String
getBurgerType();
}
```

Cheese extending AddOnsDecorator

```
public class Cheese extends AddOnsDecorator
{
    public Cheese(BaseBurger burger) {
        super.burger = burger;
    }
    @Override
    public String getBurgerType() {
        return burger.getBurgerType();
    }

    @Override
    public String getAddOns() {
        return burger.getAddOns() + "
,Cheese";
    }
}
```

Cheese constructor do set the burger type same as what's provided as a parameter, and that's why the burger type must be specified first. Then, it wraps the burger object inside of it. Therefore, when we want to print this object, and bring its addons we will call the **getAddsOns()** method and this method will return all the addons, since it's an object wrapped inside of another…

If we assumed **str** was "beef cheese Cocktail Ketchup"



```
burger = {Ketchup@707}
  burger = {Cocktail@714}
    burger = {Cheese@716}
      burger = {BeefBurger@717}
        burgerType = "Beef burger"
```

Now let's see **"Logger"** class…

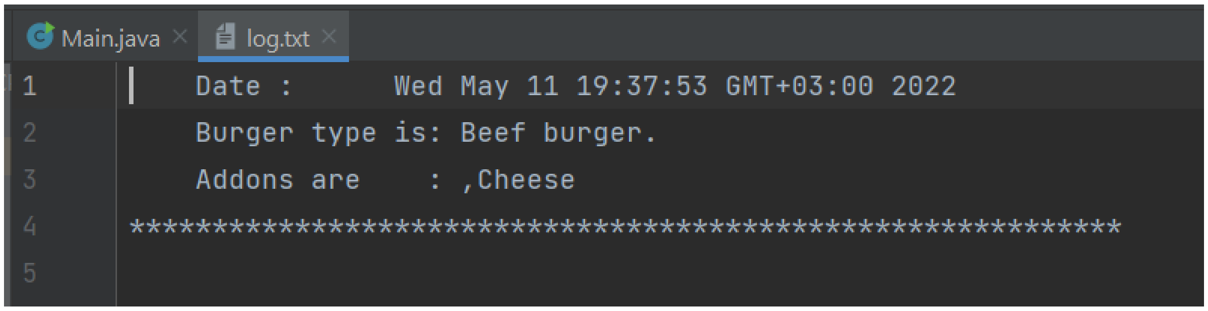
```
public class Logger {

    private static Logger instance = null;
    private PrintWriter printWriter;

    private Logger() {
        try {
            printWriter = new PrintWriter(new FileWriter("log.txt", true), true);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static Logger getInstance() {
        if (instance == null)
            return instance = new Logger();
        else
            return instance;
    }

    public void write(String str) {
        printWriter.write("\tDate :\t\t" + new Date());
        printWriter.println("\t" + str);
        printWriter.println("****************************************************************");
    }
}
```

A clear singleton implementation, same as slides which make sure that only one object is created out of this class, we have another object which is printWriter, and this object is sat to autoflush and append so every time println() method is used auto flush will flush it to the file.

Autoflush > after every **println()** it flushes it to the output file.

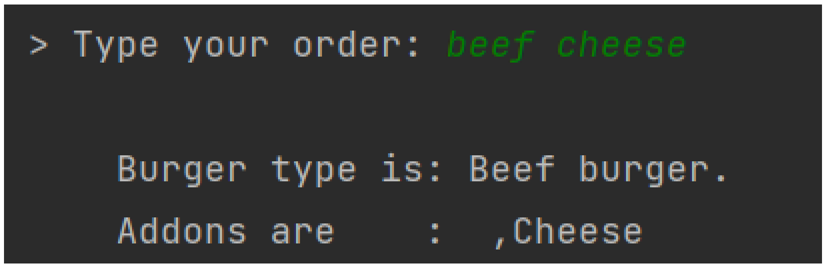
Appened > so the previous orders don't get deleted.

write() method do print what is sent to it plus the date for our example this is the output…



```
    Date :      Wed May 11 19:37:53 GMT+03:00 2022
    Burger type is: Beef burger.
    Addons are    : ,Cheese
************************************************************

```

And this is the console output…

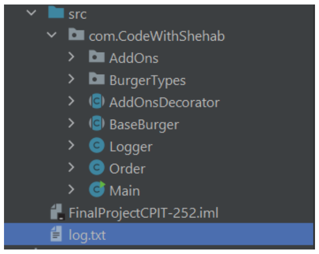


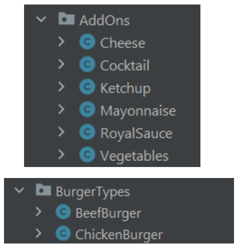
```
> Type your order: beef cheese


    Burger type is: Beef burger.
    Addons are    :  ,Cheese
```

CLASSES

Classes in the project are distributed in 3 sections.

1- Main package contains classes which they are not in any conceptual group.
2- AddOns package containing the addons on burger.
3- Burger types.





ADDONSDECORATOR ABSTRACT CLASS

```java
package com.CodeWithShehab;

public abstract class AddOnsDecorator extends BaseBurger{
    protected BaseBurger burger;

    public abstract String getBurgerType();
}
```

ADDONS PACKAGE

This package contains all the addons on burger object. Implemented using decorator design pattern.

CHEESE CLASS

```java
package com.CodeWithShehab.AddOns;

import com.CodeWithShehab.AddOnsDecorator;
import com.CodeWithShehab.BaseBurger;

public class Cheese extends AddOnsDecorator {
    public Cheese(BaseBurger burger) {
        super.burger = burger;
    }
    @Override
    public String getBurgerType() {
        return burger.getBurgerType();
    }

    @Override
    public String getAddOns() {
        return burger.getAddOns() + " ,Cheese";
    }
}
```

COCKTAIL CLASS

```java
package com.CodeWithShehab.AddOns;

import com.CodeWithShehab.AddOnsDecorator;
import com.CodeWithShehab.BaseBurger;

public class Cocktail extends AddOnsDecorator {

    public Cocktail(BaseBurger burger) {
        super.burger = burger;
    }

    @Override
    public String getBurgerType() {
        return burger.getBurgerType();
    }

    @Override
    public String getAddOns() {
        return burger.getAddOns()+ " ,Cocktail";
}
```

KETCHUP CLASS

```
package com.CodeWithShehab.AddOns;

import com.CodeWithShehab.AddOnsDecorator;
import com.CodeWithShehab.BaseBurger;

public class Ketchup extends AddOnsDecorator {
    public Ketchup(BaseBurger burger) {
        super.burger = burger;
    }

    @Override
    public String getBurgerType() {
        return burger.getBurgerType();
    }

    @Override
    public String getAddOns() {
        return burger.getAddOns() + " ,Ketchup";
    }
}
```

MAYONNAISE CLASS

```
package com.CodeWithShehab.AddOns;

import com.CodeWithShehab.AddOnsDecorator;
import com.CodeWithShehab.BaseBurger;

public class Mayonnaise extends AddOnsDecorator {

    public Mayonnaise(BaseBurger burger) {
        super.burger = burger;
    }

    @Override
    public String getBurgerType() {
        return burger.getBurgerType();
    }

    @Override
    public String getAddOns() {
        return burger.getAddOns() + " ,Mayonnaise";
    }
}
```

ROYALSAUCE CLASS

```
package com.CodeWithShehab.AddOns;

import com.CodeWithShehab.AddOnsDecorator;
import com.CodeWithShehab.BaseBurger;

public class RoyalSauce extends AddOnsDecorator {

    public RoyalSauce(BaseBurger burger) {
        this.burger = burger;
    }

    @Override
    public String getBurgerType() {
        return burger.getBurgerType();
    }

    @Override
    public String getAddOns() {
        return burger.getAddOns() + " ,Royal sauce";
    }
}
```

VEGETABLES CLASS

```java
package com.CodeWithShehab.AddOns;

import com.CodeWithShehab.AddOnsDecorator;
import com.CodeWithShehab.BaseBurger;

public class Vegetables extends AddOnsDecorator {

    public Vegetables(BaseBurger burger) {
        super.burger = burger;
    }

    @Override
    public String getBurgerType() {
        return burger.getBurgerType();
    }

    @Override
    public String getAddOns() {
        return burger.getAddOns() + " ,Vegetables";
    }
}
```

ADDONSDECORATOR ABSTRACT CLASS

```java
package com.CodeWithShehab;

public abstract class AddOnsDecorator extends BaseBurger{
    protected BaseBurger burger;

    public abstract String getBurgerType();
}
```

BURGERTYPES PACKAGE

CHICKENBURGER CLASS

```java
package com.CodeWithShehab.BurgerTypes;

import com.CodeWithShehab.BaseBurger;

public class ChickenBurger extends BaseBurger {
    public ChickenBurger() {
        super.burgerType = "Chicken burger";
    }

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

BEEFBURGER CLASS

```java
package com.CodeWithShehab.BurgerTypes;

import com.CodeWithShehab.BaseBurger;

public class BeefBurger extends BaseBurger {
    public BeefBurger() {
        super.burgerType = "Beef burger";
    }

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

ORDER CLASS

```
package com.CodeWithShehab;

import com.CodeWithShehab.AddOns.*;
import com.CodeWithShehab.BurgerTypes.*;

public class Order {

    private Order() {
    }

    public static void makeOrder(String str) {
        Logger logger = Logger.getInstance();
        BaseBurger burger = null;

        String[] list = str.split(" ");

        for (int i = 0; i < list.length; i++) {
            switch (list[i].toLowerCase()) {
                case "cheese":
                    burger = new Cheese(burger);
                    break;
                case "chicken":
                    burger = new ChickenBurger();
                    break;
                case "beef":
                    burger = new BeefBurger();
                    break;
                case "cocktail":
                    burger = new Cocktail(burger);
                    break;
                case "ketchup":
                    burger = new Ketchup(burger);
                    break;
                case "mayonnaise":
                    burger = new Mayonnaise(burger);
                    break;
                case "royalsauce":
                    burger = new RoyalSauce(burger);
                    break;
                case "vegetables":
                    burger = new Vegetables(burger);
                    break;
            }
        }

        System.out.println("\n\tBurger type is: " + burger.getBurgerType() + ". \n\tAddons are    :
" + burger.getAddOns());
        logger.write("\n\tBurger type is: " + burger.getBurgerType() + ". \n\tAddons are    :" +
burger.getAddOns());
    }
}
```

The constructor is private because there’s no need to create an object form this class.

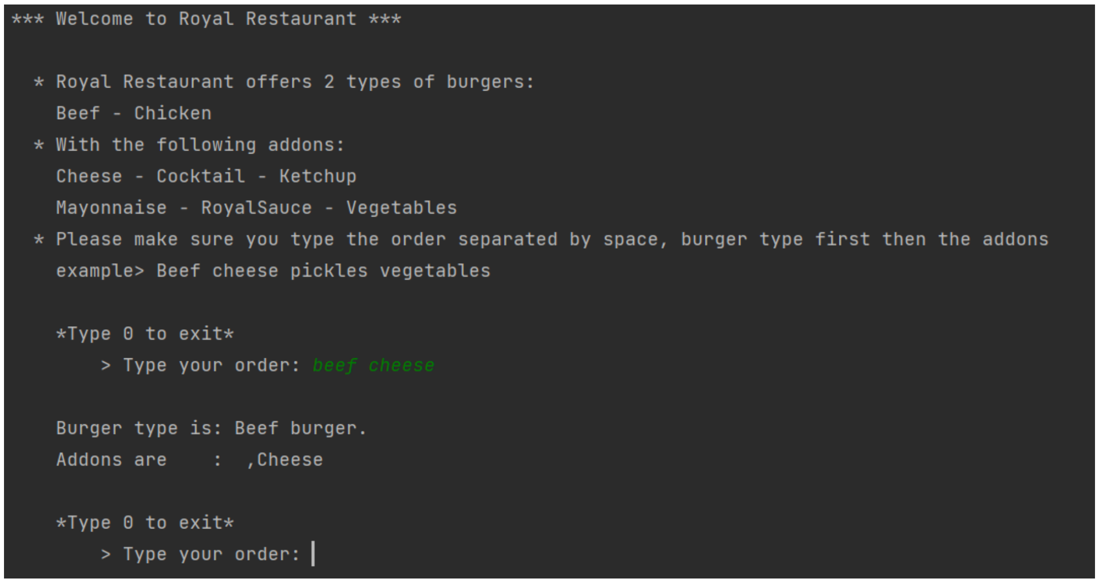
LOGGER CLASS

```java
package com.CodeWithShehab;

import java.io.*;
import java.util.Date;

public class Logger {

    private static Logger instance = null;
    private PrintWriter printWriter;

    private Logger() {
        try {
            printWriter = new PrintWriter(new FileWriter("log.txt", true), true);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static Logger getInstance() {
        if (instance == null)
            return instance = new Logger();
        else
            return instance;
    }

    public void write(String str) {
        printWriter.write("\tDate :\t\t" + new Date());
        printWriter.println("\t" + str);
        printWriter.println("************************************************************");
    }
}
```

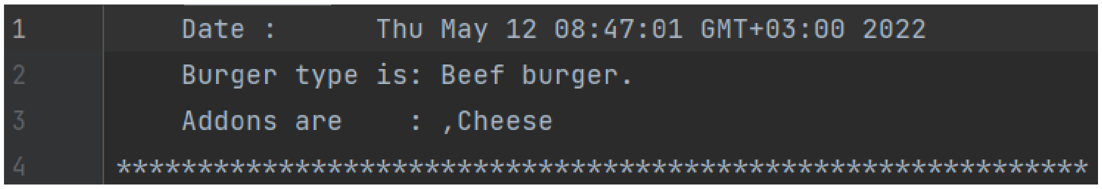
MAIN CLASS

```java
package com.CodeWithShehab;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        displayMenu();
    }

    public static void displayMenu(){
        System.out.println("*** Welcome to Royal Restaurant ***\n");
        System.out.println("  * Royal Restaurant offers 2 types of burgers:");
        System.out.println("\tBeef - Chicken");
        System.out.println("  * With the following addons:");
        System.out.println("\tCheese - Cocktail - Ketchup \n\tMayonnaise - RoyalSauce -
Vegetables");
        System.out.println("  * Please make sure you type the " +
                "order separated by space, burger type first then the addons");
        System.out.println("\texample> Beef cheese pickles vegetables");
        while (true) {
            System.out.println("\n\t*Type 0 to exit*");
            System.out.print("\t\t> Type your order: ");
            String choice = new Scanner(System.in).nextLine();
            if (choice.equals("0")) {
                break;
            }
            Order.makeOrder(choice);
        }
        System.out.println("Thanks for using the system!");
    }
}
```

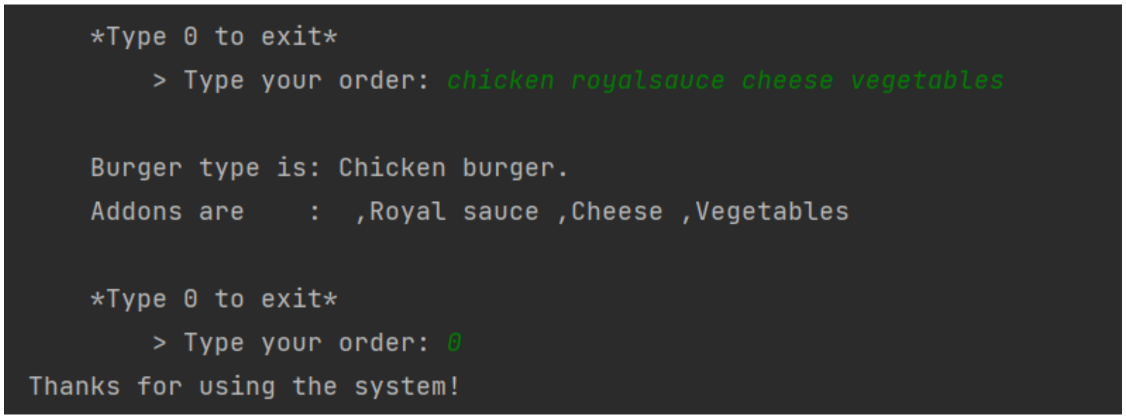
OUTPUT



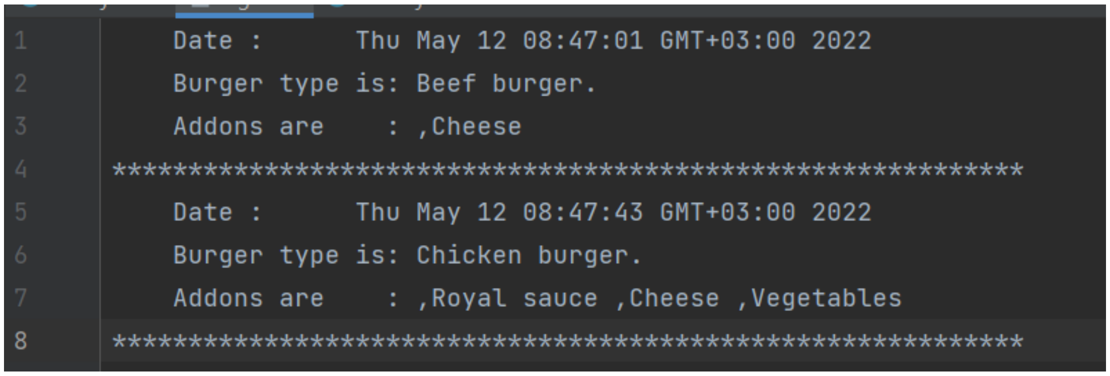
```
*** Welcome to Royal Restaurant ***

  * Royal Restaurant offers 2 types of burgers:
    Beef - Chicken
  * With the following addons:
    Cheese - Cocktail - Ketchup
    Mayonnaise - RoyalSauce - Vegetables
  * Please make sure you type the order separated by space, burger type first then the addons
    example> Beef cheese pickles vegetables

    *Type 0 to exit*
        > Type your order: beef cheese

    Burger type is: Beef burger.
    Addons are     :  ,Cheese

    *Type 0 to exit*
        > Type your order: 
```



```
        Date :       Thu May 12 08:47:01 GMT+03:00 2022
        Burger type is: Beef burger.
        Addons are     : ,Cheese
************************************************************
```



```
    *Type 0 to exit*
        > Type your order: chicken royalsauce cheese vegetables

    Burger type is: Chicken burger.
    Addons are     :  ,Royal sauce ,Cheese ,Vegetables

    *Type 0 to exit*
        > Type your order: 0
Thanks for using the system!
```



```
        Date :       Thu May 12 08:47:01 GMT+03:00 2022
        Burger type is: Beef burger.
        Addons are     : ,Cheese
************************************************************
        Date :       Thu May 12 08:47:43 GMT+03:00 2022
        Burger type is: Chicken burger.
        Addons are     : ,Royal sauce ,Cheese ,Vegetables
************************************************************
```