PIZZA WORLD

(AP LAB II PROJECT)

Computer Science and Engineering

DEC – 2020

Submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology Department of Computer Science & Engineering

Jaypee University of Engineering and Technology, A-B ROAD, RAGHOGARH, DT. GUNA - 473226, M.P., INDIA

TEAM DESCRIPTION

ABHAY GARG (181B007)

ADITI NEGI (181B013)

SUBJECT FACULTY:

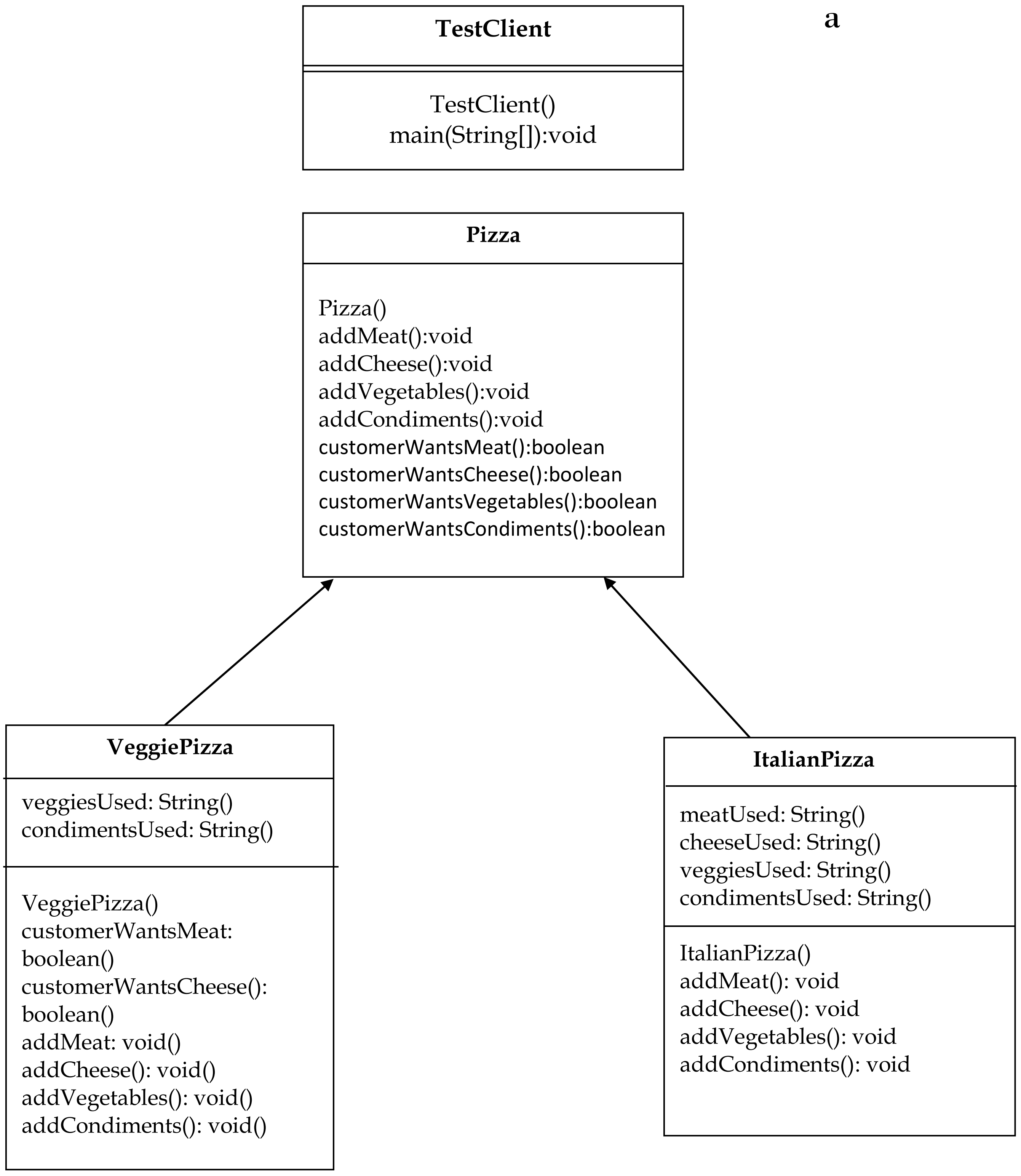
DR. PRATEEK PANDEY

INDEX

<u>**PROJECT PROBLEM**</u>

We have a Pizza shop which makes two types of pizzas one is Italian and other one is Veggie. In both types of pizzas we provide three types of condiments : Oil, Vinegar, Butter. We offer 3 types of meat(Pork, Pepperoni, Chicken) and several types of veggies. Customer can order any pizza based on their preference.

SOLUTION DESIGN



DESIGN PATTERN

In this project we have used Template Method Design Pattern.

Template method design pattern is used to define an algorithm as a skeleton of operations and leave the details to be implemented by the child classes. The overall structure and sequence of the algorithm is preserved by the parent class.

Template means Preset format like HTML templates which has a fixed preset format. Similarly in the template method pattern, we have a preset structure method called template method which consists of steps. This steps can be an abstract method which will be implemented by its subclasses.

This behavioral design pattern is one of the easiest to understand and implement. This design pattern is used popularly in framework development. This helps to avoid code duplication also.

AbstractClass contains the templateMethod() which should be made final so that it cannot be overridden. This template method makes use of other operations available in order to run the algorithm but is decoupled for the actual implementation of these methods. All operations used by this template method are made abstract, so their implementation is deferred to subclasses.

ConcreteClass implements all the operations required by the templateMethod that were defined as abstract in the parent class. There can be many different ConcreteClasses.

JUSTIFICATION:

The template method is used in frameworks, where each implements the invariant parts of a domain's architecture, leaving "placeholders" for customization options.

The template method is used for the following reasons :

Let subclasses implement varying behavior (through method overriding)

Avoid duplication in the code, the general workflow structure is implemented once in the abstract class’s algorithm, and necessary variations are implemented in the subclasses.

Control at what points subclassing is allowed. As opposed to a simple polymorphic override, where the base method would be entirely rewritten allowing radical change to the workflow, only the specific details of the workflow are allowed to change.

CODE

TestClient.java

```
public class TestClient {

	public static void main(String[] args) {

		System.out.println("Start: TemplateMethod\n");

		//
		// Create ItalianPizza
		//

		Pizza customer1Pizza = new ItalianPizza();
		customer1Pizza.templateMethod();

		System.out.println("\n");

		//
		// Create VeggiePizza
		//

		Pizza customer2Pizza = new VeggiePizza();
		customer2Pizza.templateMethod();

	}

}
```

Pizza.java

```
public abstract class Pizza {

	//
	// TemplateMethod


	final void templateMethod() {

		cutBase();
}



        if (customerWantsMeat()) {
                addMeat();
        }

        if (customerWantsCheese()) {
                addCheese();
        }

        if (customerWantsVegetables()) {
                addVegetables();
        }

        if (customerWantsCondiments()) {
                addCondiments();
        }

                packThePizza();

}

public void cutBase() {
        System.out.println("The Pizza is cut into slices");
}

public void packThePizza() {
        System.out.println("Pack the Pizza");
}

abstract void addMeat();

abstract void addCheese();

abstract void addVegetables();

abstract void addCondiments();


boolean customerWantsMeat() {
        return true;
}
```

```
	boolean customerWantsCheese() {
		return true;
	}

	boolean customerWantsVegetables() {
		return true;
	}

	boolean customerWantsCondiments() {
		return true;
	}

}
```

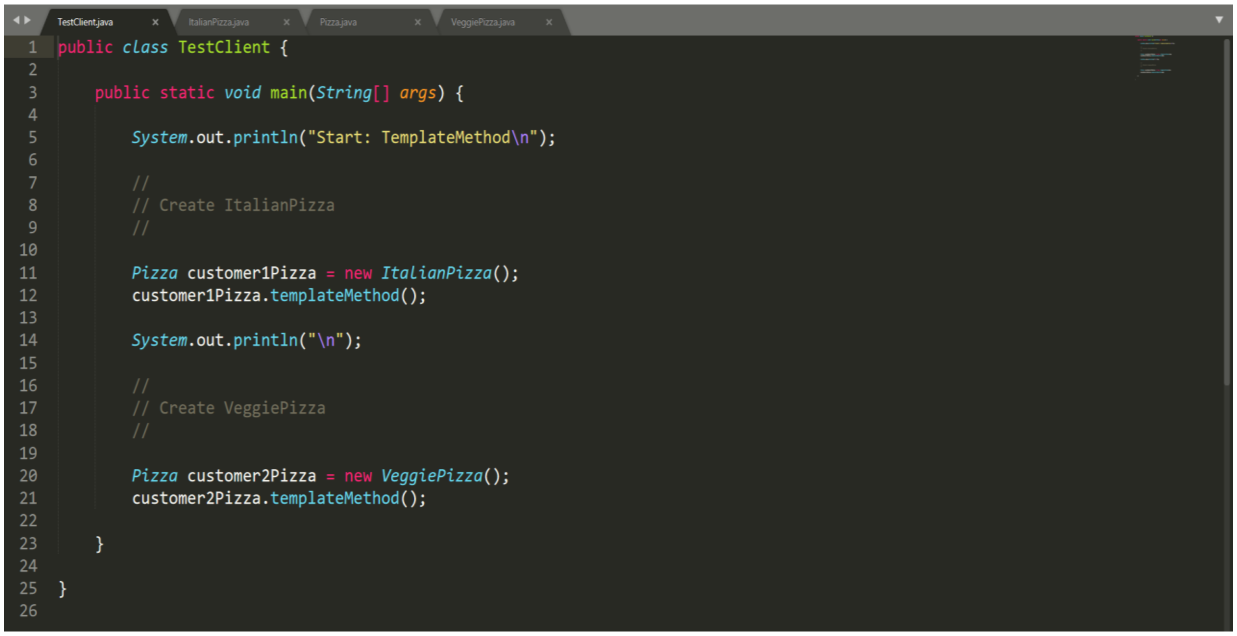
VeggiePizza.java

```
public class VeggiePizza extends Pizza {

	String[] veggiesUsed = { "Lettuce", "Tomatoes", "Onions", "Sweet Pappers" };
	String[] condimentsUsed = { "Oil", "Vinegar" };


	boolean customerWantsMeat() {
		return false; //false
	}

	boolean customerWantsCheese() {
		return false; //false
	}

	//
	//
	//

	@Override
	void addMeat() {


	}

	@Override
	void addCheese() {



	}

	@Override
	void addVegetables() {
		System.out.println("\n");
		System.out.println("Adding the veggies: ");

		for (String veggie : veggiesUsed) {
			System.out.println(veggie + " ");
		}

	}

	@Override
	void addCondiments() {
		System.out.println("\n");
		System.out.println("Adding the condiments: ");

		for (String condiment : condimentsUsed) {
			System.out.println(condiment + " ");
		}

	}

}
```

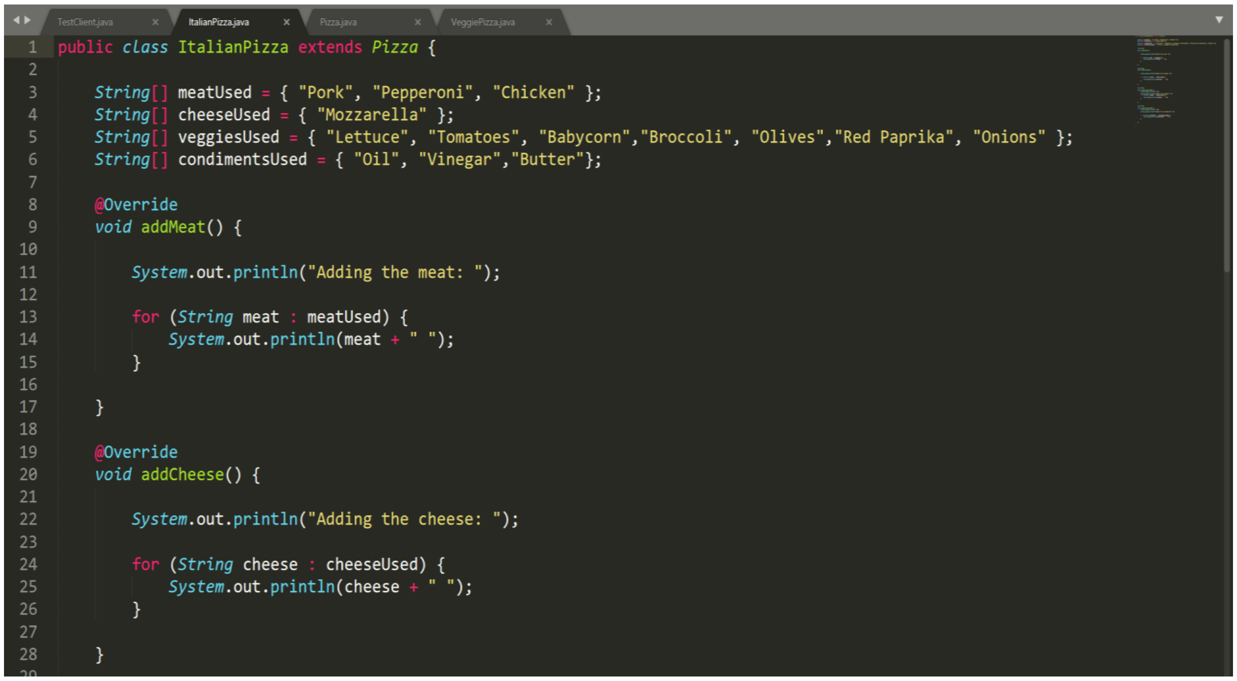
ItalianPizza.java

```
public class ItalianPizza extends Pizza {

	String[] meatUsed = { "Pork", "Pepperoni", "Chicken" };
	String[] cheeseUsed = { "Mozzarella" };
	String[] veggiesUsed = { "Lettuce", "Tomatoes", "Babycorn","Broccoli",
"Olives","Red Paprika", "Onions" };
	String[] condimentsUsed = { "Oil", "Vinegar","Butter"};

	@Override
	void addMeat() {
		System.out.println("Adding the meat: ");

		for (String meat : meatUsed) {
			System.out.println(meat + " ");
		}

	}

	@Override
	void addCheese() {

		System.out.println("Adding the cheese: ");

		for (String cheese : cheeseUsed) {
			System.out.println(cheese + " ");
		}

	}

	@Override
	void addVegetables() {
		System.out.println("\n");
		System.out.println("Adding the veggies: ");
		for (String veggie : veggiesUsed) {
			System.out.println(veggie + " ");
		}

	}

	@Override
	void addCondiments() {
		System.out.println("\n");
		System.out.println("Adding the condiments: ");

		for (String condiment : condimentsUsed) {
			System.out.println(condiment + " ");
		}

	}

}
```

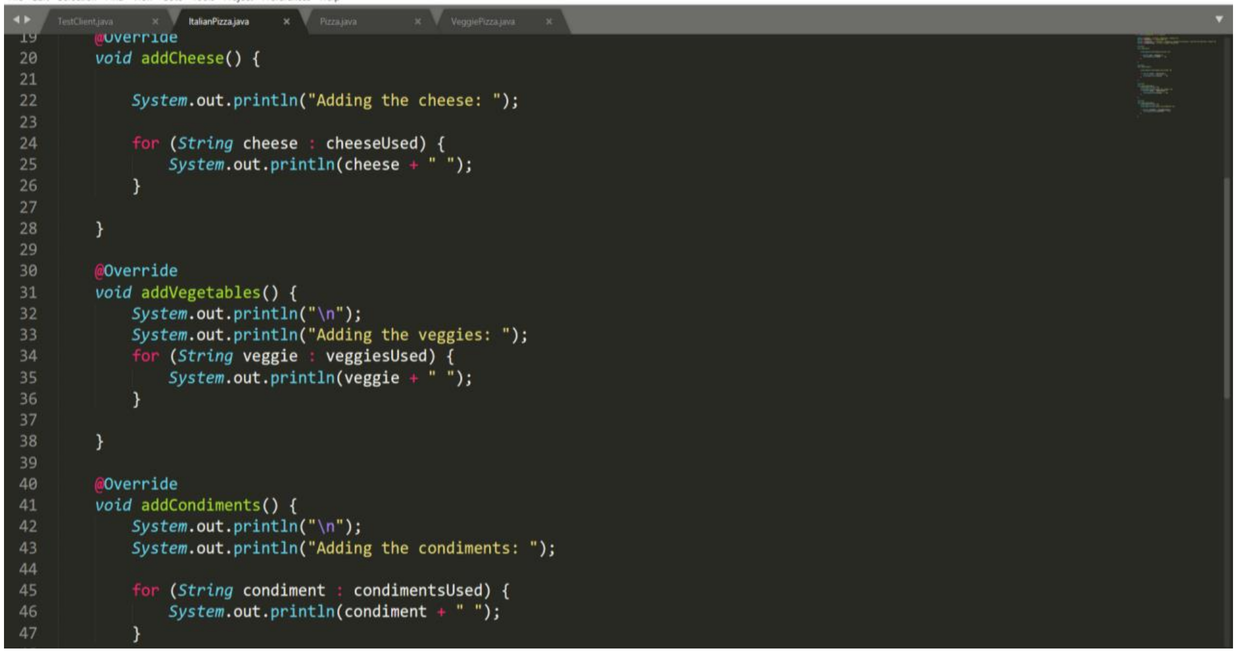
SCREENSHOTS


TestClient.java ItalianPizza.java Pizza.java VeggiePizza.java

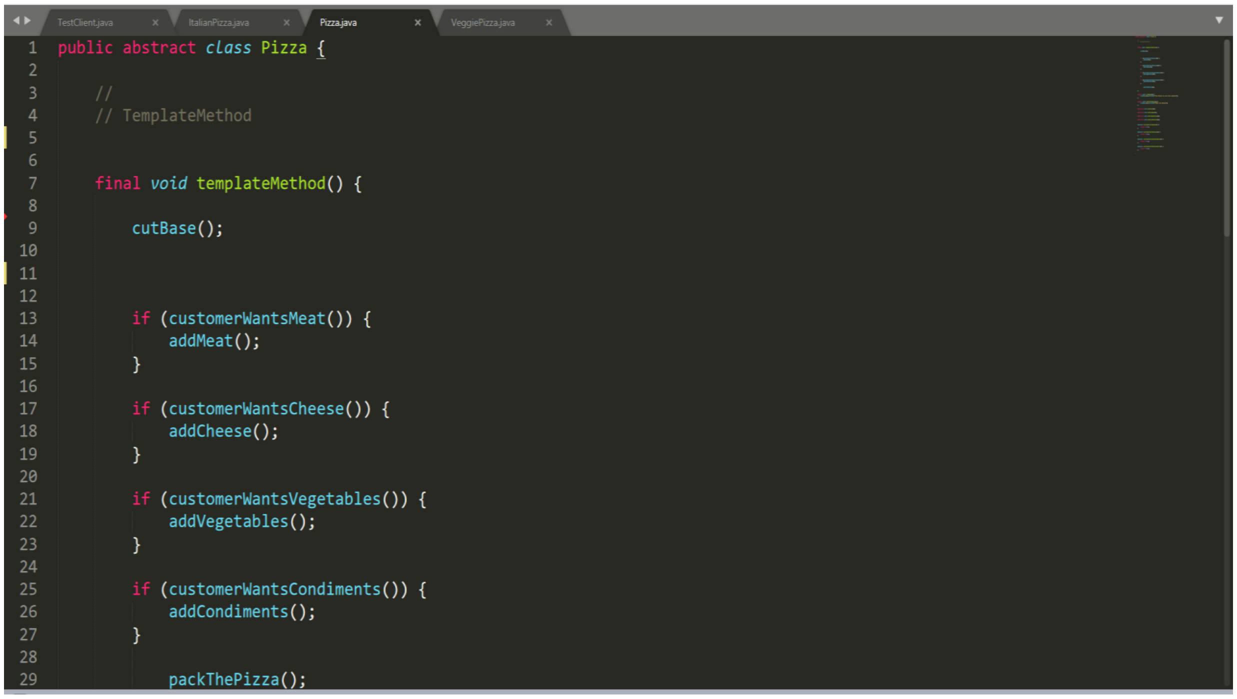
```java
public class TestClient {

    public static void main(String[] args) {

        System.out.println("Start: TemplateMethod\n");

        //
        // Create ItalianPizza
        //

        Pizza customer1Pizza = new ItalianPizza();
        customer1Pizza.templateMethod();

        System.out.println("\n");

        //
        // Create VeggiePizza
        //

        Pizza customer2Pizza = new VeggiePizza();
        customer2Pizza.templateMethod();

    }

}

```


TestClient.java ItalianPizza.java Pizza.java VeggiePizza.java

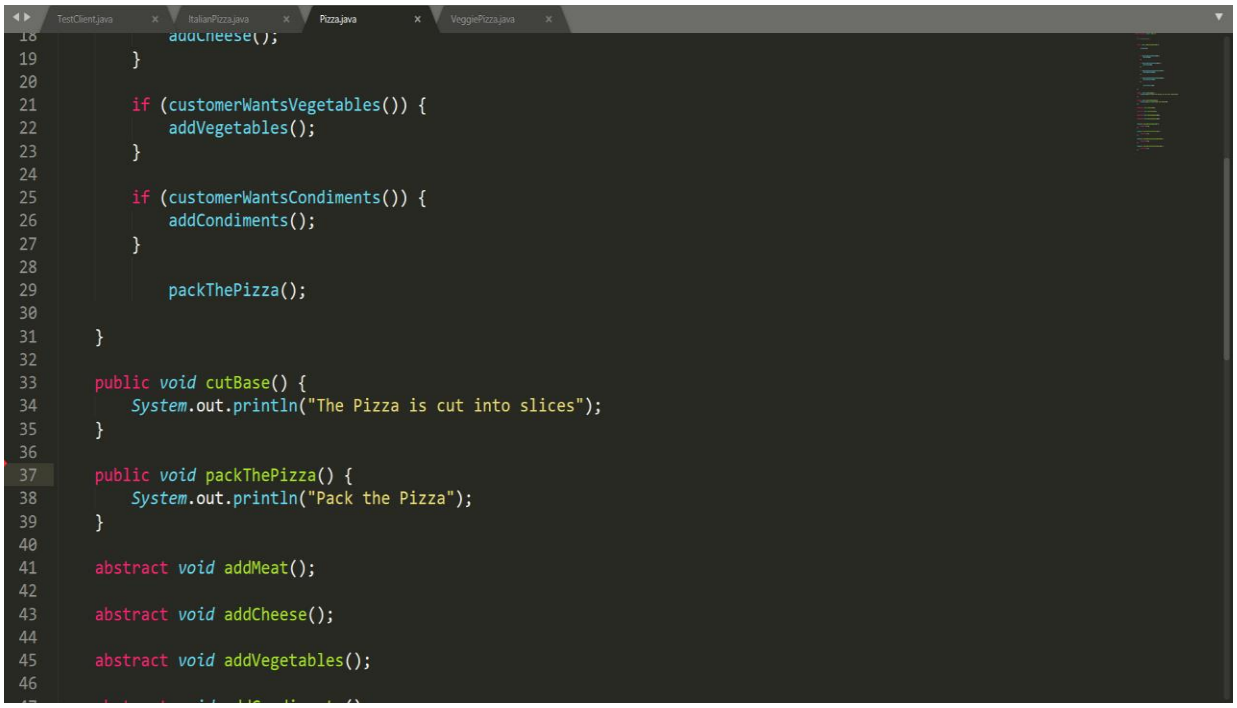
```java
public class ItalianPizza extends Pizza {

    String[] meatUsed = { "Pork", "Pepperoni", "Chicken" };
    String[] cheeseUsed = { "Mozzarella" };
    String[] veggiesUsed = { "Lettuce", "Tomatoes", "Babycorn","Broccoli", "Olives","Red Paprika", "Onions" };
    String[] condimentsUsed = { "Oil", "Vinegar","Butter"};

    @Override
    void addMeat() {

        System.out.println("Adding the meat: ");

        for (String meat : meatUsed) {
            System.out.println(meat + " ");
        }

    }

    @Override
    void addCheese() {

        System.out.println("Adding the cheese: ");

        for (String cheese : cheeseUsed) {
            System.out.println(cheese + " ");
        }

    }
```



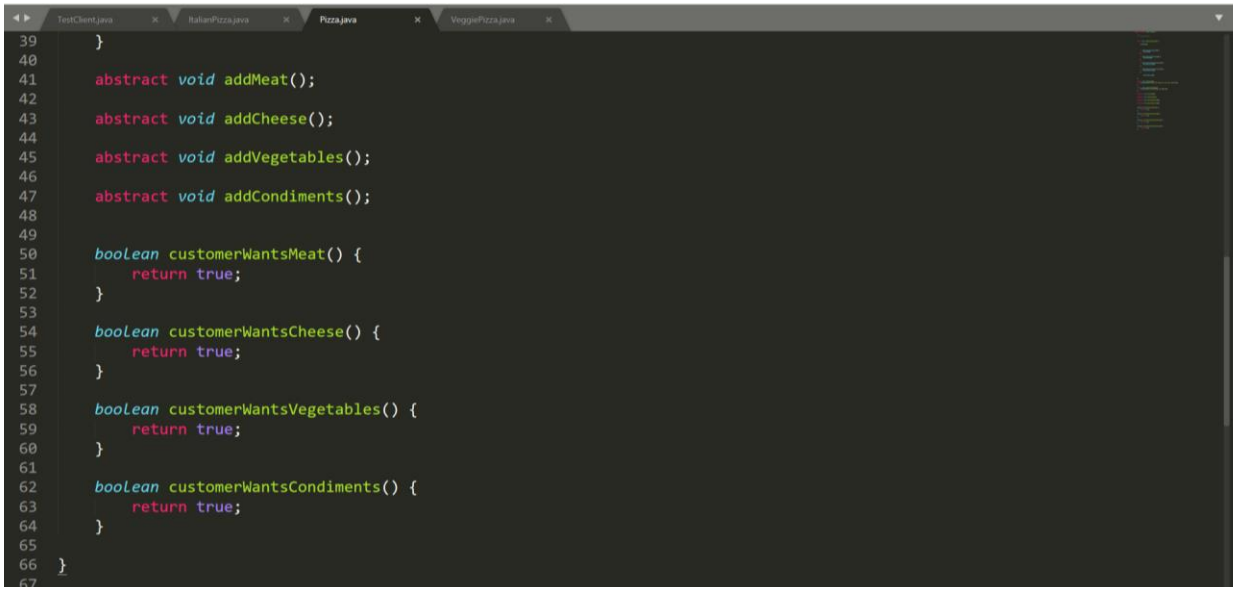
```java
    @Override
    void addCheese() {

        System.out.println("Adding the cheese: ");

        for (String cheese : cheeseUsed) {
            System.out.println(cheese + " ");
        }

    }

    @Override
    void addVegetables() {
        System.out.println("\n");
        System.out.println("Adding the veggies: ");
        for (String veggie : veggiesUsed) {
            System.out.println(veggie + " ");
        }

    }

    @Override
    void addCondiments() {
        System.out.println("\n");
        System.out.println("Adding the condiments: ");

        for (String condiment : condimentsUsed) {
            System.out.println(condiment + " ");
        }
```



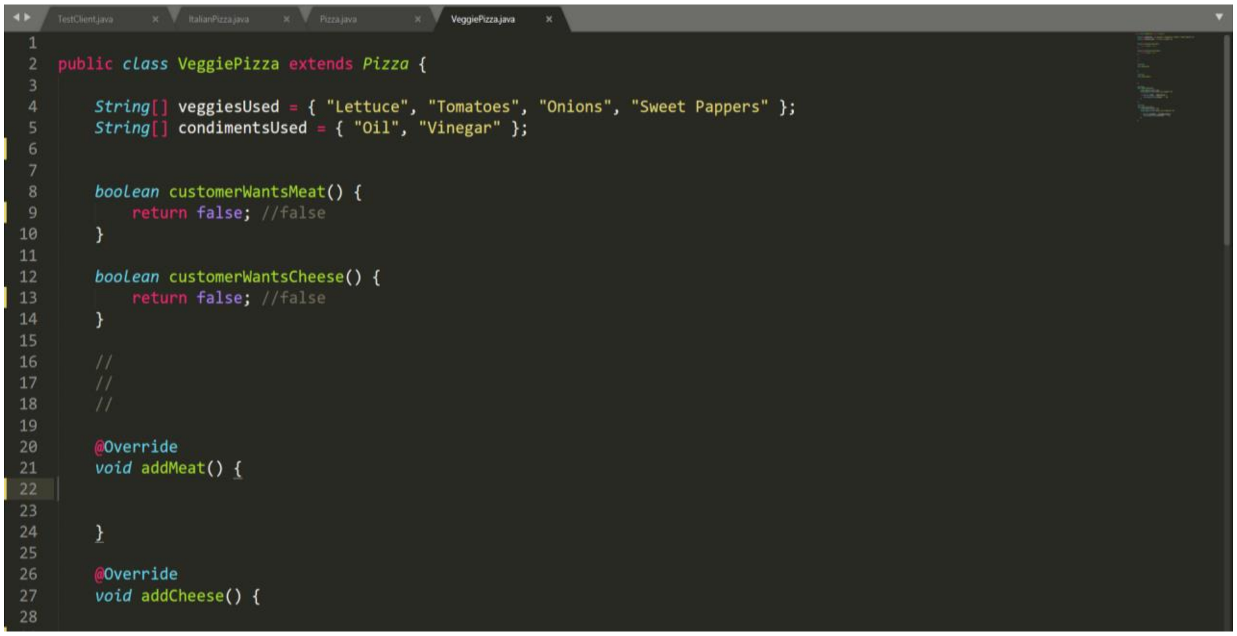
```java
public abstract class Pizza {

    //
    // TemplateMethod


    final void templateMethod() {

        cutBase();



        if (customerWantsMeat()) {
            addMeat();
        }

        if (customerWantsCheese()) {
            addCheese();
        }

        if (customerWantsVegetables()) {
            addVegetables();
        }

        if (customerWantsCondiments()) {
            addCondiments();
        }

            packThePizza();
```



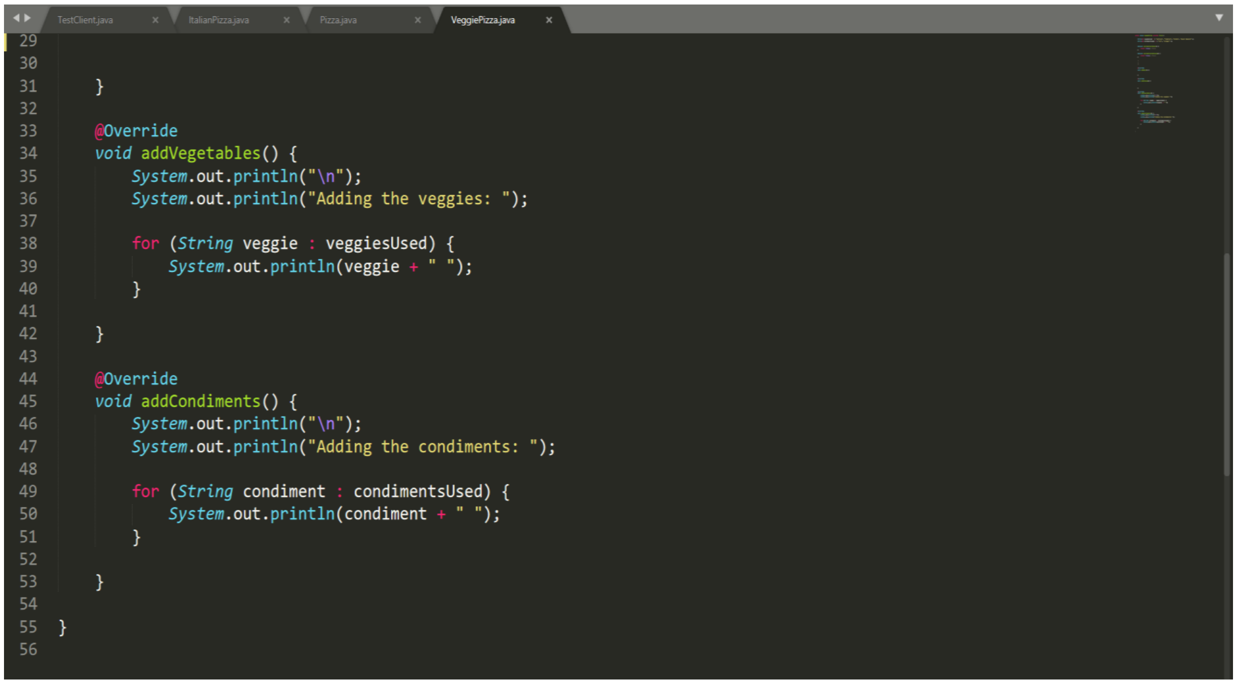
```
            addCheese();
        }

        if (customerWantsVegetables()) {
            addVegetables();
        }

        if (customerWantsCondiments()) {
            addCondiments();
        }

            packThePizza();

    }

    public void cutBase() {
        System.out.println("The Pizza is cut into slices");
    }

    public void packThePizza() {
        System.out.println("Pack the Pizza");
    }

    abstract void addMeat();

    abstract void addCheese();

    abstract void addVegetables();

```



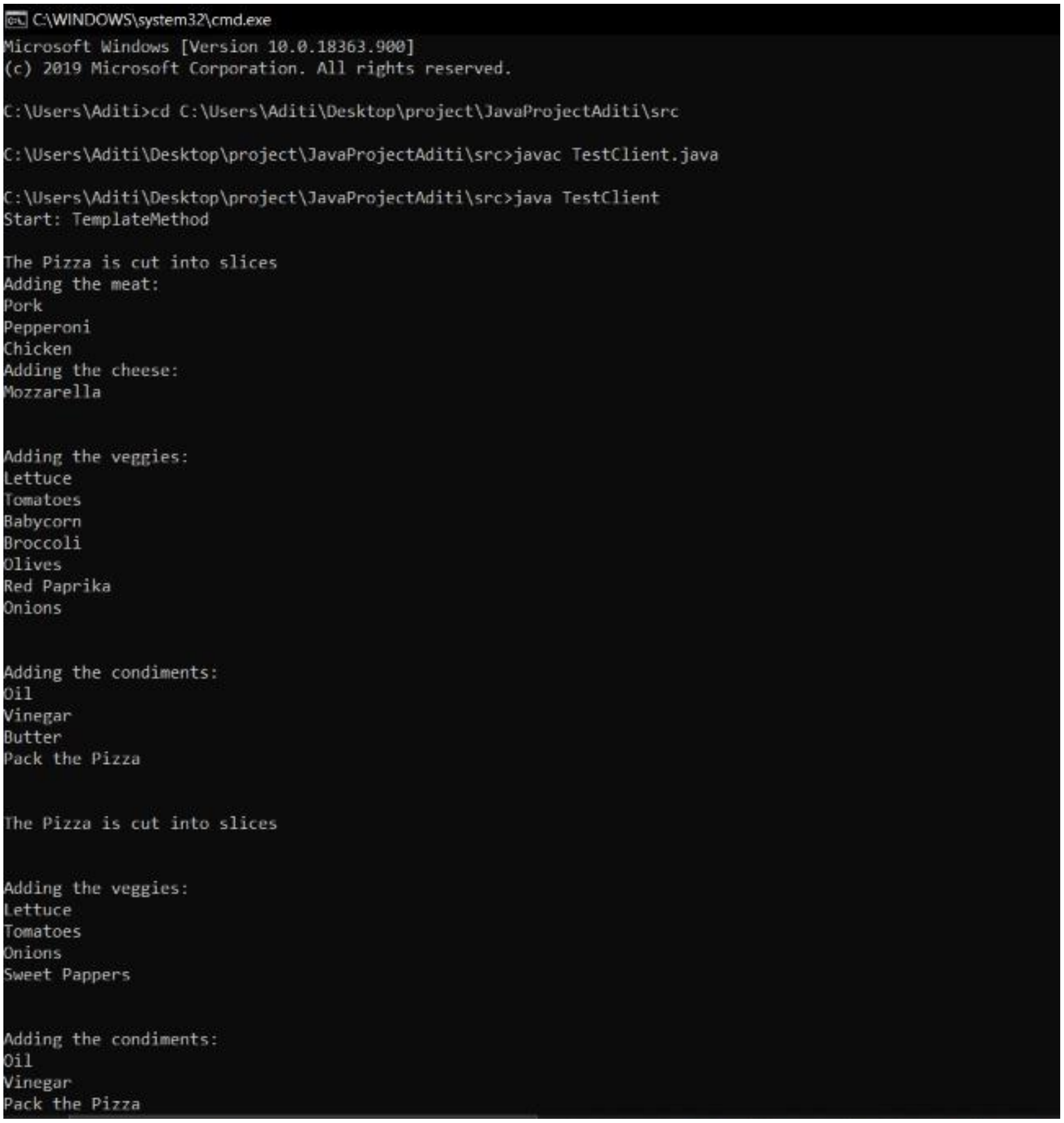
```
    }

    abstract void addMeat();

    abstract void addCheese();

    abstract void addVegetables();

    abstract void addCondiments();


    boolean customerWantsMeat() {
        return true;
    }

    boolean customerWantsCheese() {
        return true;
    }

    boolean customerWantsVegetables() {
        return true;
    }

    boolean customerWantsCondiments() {
        return true;
    }

}
```



```

public class VeggiePizza extends Pizza {

    String[] veggiesUsed = { "Lettuce", "Tomatoes", "Onions", "Sweet Pappers" };
    String[] condimentsUsed = { "Oil", "Vinegar" };


    boolean customerWantsMeat() {
        return false; //false
    }

    boolean customerWantsCheese() {
        return false; //false
    }

    //
    //
    //

    @Override
    void addMeat() {


    }

    @Override
    void addCheese() {

```



```


    }

    @Override
    void addVegetables() {
        System.out.println("\n");
        System.out.println("Adding the veggies: ");

        for (String veggie : veggiesUsed) {
            System.out.println(veggie + " ");
        }

    }

    @Override
    void addCondiments() {
        System.out.println("\n");
        System.out.println("Adding the condiments: ");

        for (String condiment : condimentsUsed) {
            System.out.println(condiment + " ");
        }

    }

}

```



C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Aditi>cd C:\Users\Aditi\Desktop\project\JavaProjectAditi\src

C:\Users\Aditi\Desktop\project\JavaProjectAditi\src>javac TestClient.java

C:\Users\Aditi\Desktop\project\JavaProjectAditi\src>java TestClient
Start: TemplateMethod

The Pizza is cut into slices
Adding the meat:
Pork
Pepperoni
Chicken
Adding the cheese:
Mozzarella


Adding the veggies:
Lettuce
Tomatoes
Babycorn
Broccoli
Olives
Red Paprika
Onions


Adding the condiments:
Oil
Vinegar
Butter
Pack the Pizza


The Pizza is cut into slices


Adding the veggies:
Lettuce
Tomatoes
Onions
Sweet Pappers


Adding the condiments:
Oil
Vinegar
Pack the Pizza
```