TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN CUỐI KÌ MÔN MẪU THIẾT KẾ

ỨNG DỤNG QUẢN LÝ CÔNG TY

Người hướng dẫn: **GV NGUYỄN THANH PHƯỚC**
Người thực hiện: **LÊ HOÀNG LONG – 518H0035**
TRẦN NGỌC THÁI SƠN – 518H0262
NGUYỄN HỮU TÀI – 518H0558
Lớp **: 18H50205**
Khoá **: 22**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN CUỐI KÌ MÔN MẪU THIẾT KẾ

ỨNG DỤNG QUẢN LÝ CÔNG TY

Người hướng dẫn: **GV NGUYỄN THANH PHƯỚC**
Người thực hiện: **LÊ HOÀNG LONG – 518H0035**
TRẦN NGỌC THÁI SƠN – 518H0262
NGUYỄN HỮU TÀI – 518H0558
Lớp **: 18H50205**
Khoá **: 22**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

LỜI CẢM ƠN

Đầu tiên, em xin bày tỏ lòng biết ơn sâu sắc đến Trường Đại học Tôn Đức Thắng và khoa Công Nghệ Thông Tin đã tạo điều kiện để lớp học được mở cũng như cung cấp mọi điều kiện tốt nhất cho khóa học. Đặc biệt, chúng em xin chân thành cảm ơn Giảng viên Nguyễn Thanh Phước đã quan tâm, giúp đỡ, hướng dẫn em mọi lúc để chúng em có thể hoàn thành bài báo cáo trong thời gian qua.

Bài báo cáo này là sản phẩm của riêng chúng em, nên không thể tránh khỏi những sai sót. Kính mong nhận được sự chỉ bảo đóng góp ý kiến của các thầy cô, cũng như mọi người để chúng em ngày càng hoàn thiện thêm kiến thức.

Chúng Em xin chân thành cảm ơn!

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi / chúng tôi và được sự hướng dẫn của TS Nguyễn Văn A;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm

Tác giả

(ký tên và ghi rõ họ tên)

Lê Hoàng Long

Trần Ngọc Thái Sơn

Nguyễn Hữu Tài

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Đây là ứng dụng quản lý công ty áp dụng các mẫu thiết kế đã học vào để thực hiện.

MỤC LỤC

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

CHƯƠNG 1
GIỚI THIỆU TÍNH NĂNG CHƯƠNG TRÌNH

1.1 Giới thiệu về Design Pattern

Design pattern là các giải pháp tổng thể đã được tối ưu hóa, được tái sử dụng cho các vấn đề phổ biến trong thiết kế phần mềm mà chúng ta thường gặp phải hàng ngày.

Design pattern được sử dụng để:

- Giúp sản phẩm của chúng ta linh hoạt, dễ dàng thay đổi và bảo trì hơn.
- Có một điều luôn xảy ra trong phát triển phần mềm, đó là sự thay đổi về yêu cầu. Lúc này hệ thống phình to, các tính năng mới được thêm vào trong khi performance cần được tối ưu hơn.
- Giúp cho các lập trình viên có thể hiểu code của người khác một cách nhanh chóng (có thể hiểu là các mối quan hệ giữa các module chẳng hạn). Mọi thành viên trong team có thể dễ dàng trao đổi với nhau để cùng xây dựng dự án mà không tốn nhiều thời gian.

1.2 Giới thiệu về tính năng của chương trình:

Trên thực tế, kinh doanh là một lĩnh vực được đánh giá rất cao trên thị trường hiện nay. Do đó mà không ít những ứng dụng quản lý công ty được ra đời để hỗ trợ.

Một công ty, doanh nghiệp có thể vận hành, quản trị thành công hay không phụ thuộc rất lớn vào chất lượng của ứng dụng quản lý họ áp dụng. Đó là lý do chúng em làm ra sản phẩm CompanyManagement.

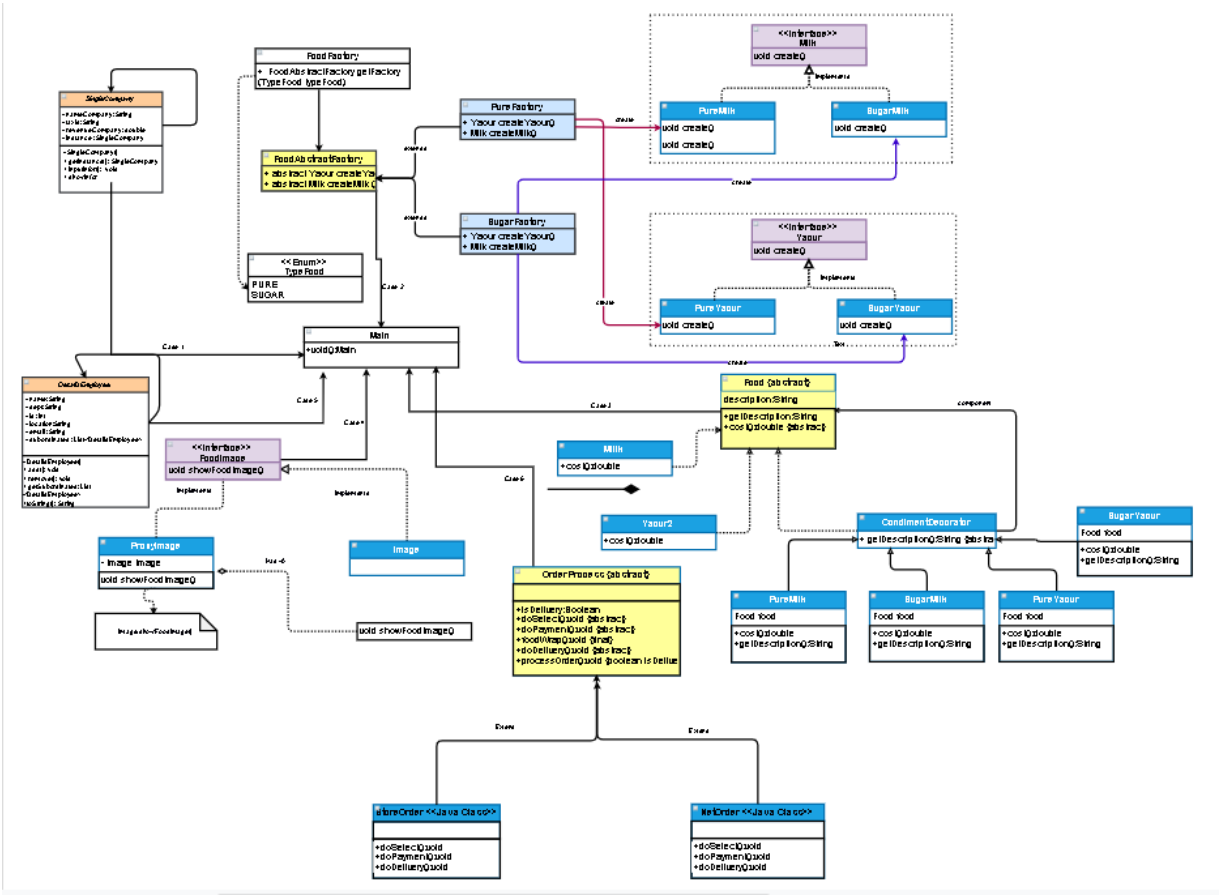
Ở đây chúng em demo CompanyManagment với quản lý việc bác thức ăn và quản lý nhân viên

Với CompanyManagement bạn có thể:

- 1. Show information about company

- 2. Creat Food
- 3. Show Menu of Food
- 4. Show information about employee
- 5. Order Food
- 6. Image of Food

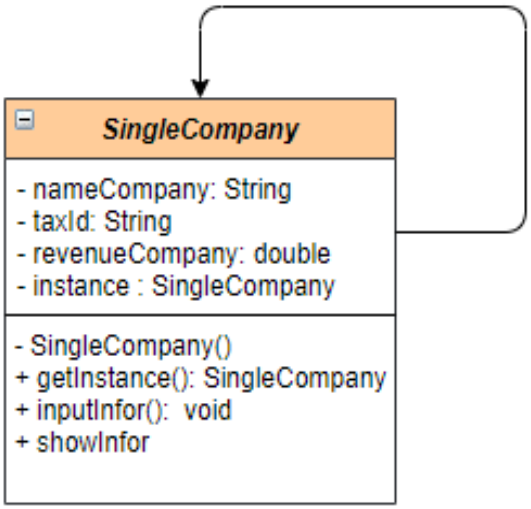
1.3 Tổng thể thiết kế:



CHƯƠNG 2
CÁC MẪU THIẾT KẾ

2.1 Mẫu thiết kế thứ 1: Singleton Pattern

2.1.1 Sơ đồ thiết kế:



Singleton pateern là một trong những mẫu thiết kế đơn giản nhất trong java. Mẫu thiết kế này cung cấp một trong những cách tốt nhất để tạo một đối tượng. Mẫu này liên quan đến một lớp duy nhất chịu trách nhiệm tạo một đối tượng đảm bảo rằng chỉ có một đối tượng duy nhất được tạo. Lớp này cung cấp một cách để truy cập đối tượng duy nhất của nó để được truy cập trực tiếp mà không cần khởi tạo đối tượng lớp.

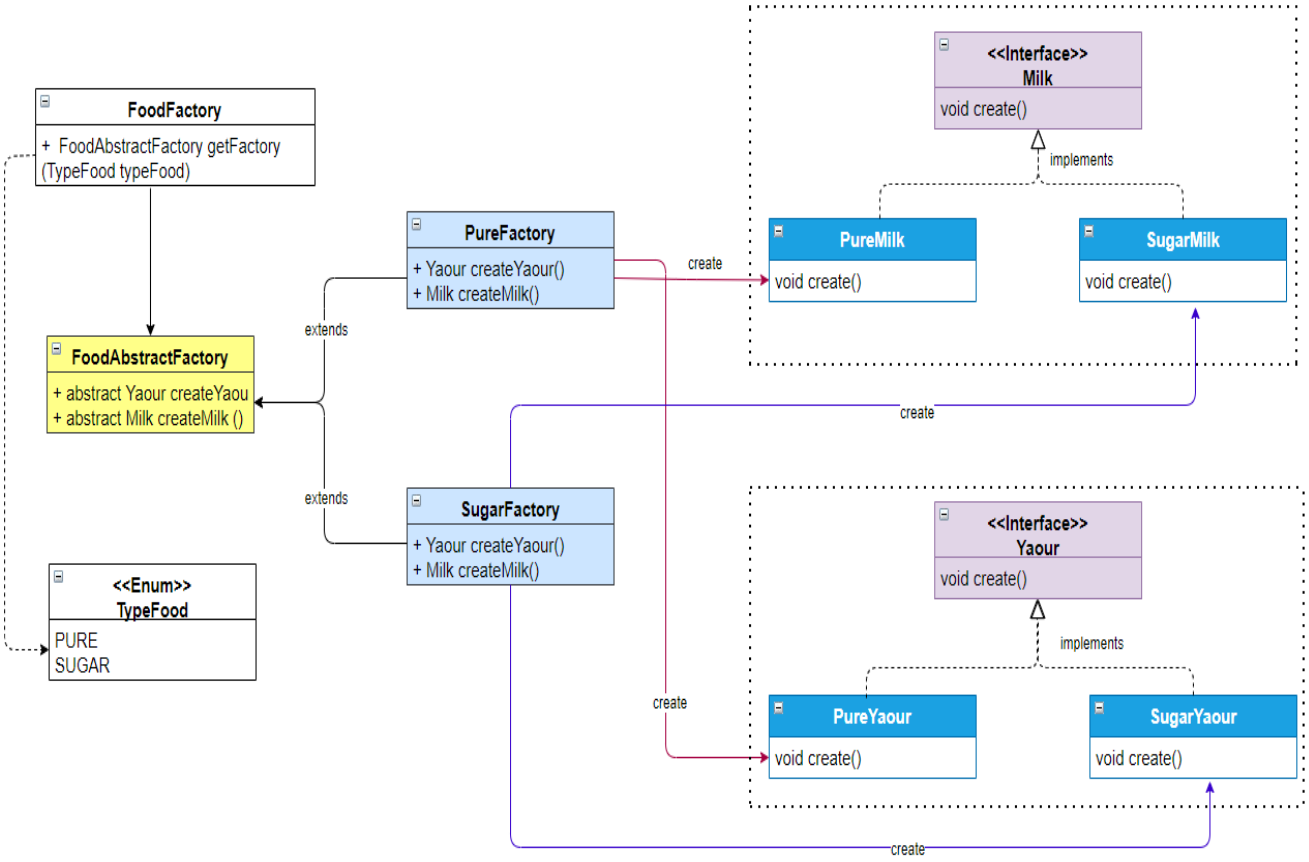
2.1.2 Code Demo:

Chúng ta sẽ tạo một lớp SingleCompany. Lớp này có hàm tạo của nó là private và nó tĩnh của chính nó. Lớp SingleCompany cung cấp phương thức tĩnh để thực hiện phương thức tĩnh. Ở đây chúng em cho phương thức khởi tạo Company với các nội dung nameCompany, taxId, revenueCompany.

```java
package Entity.Singleton;

import java.util.Scanner;

public class SingleCompany {
	private static String nameCompany;
	private static String taxId;
	private static double revenueCompany;

	private static SingleCompany instance = new SingleCompany(nameCompany,
nameCompany, revenueCompany);


	private SingleCompany(String nameCompany, String taxId, double revenueCompany)
{
		this.nameCompany = nameCompany;
		this.taxId = taxId;
		this.revenueCompany= revenueCompany;
	}

	public static SingleCompany getInstance() {
		return instance;
	}

	public void inputInfor(Scanner scanner) {
		System.out.println("Input name Company: ");
		this.nameCompany = scanner.nextLine();
		System.out.println("Input TaxId: ");
		this.taxId = scanner.nextLine();
		System.out.println("Input revenue Company: ");
		this.revenueCompany = Float.parseFloat(scanner.nextLine());
	}

	public void showInfor() {
		System.out.println("--ABOUT COMPANY--");
		System.out.println("\tName: " + this.nameCompany);
		System.out.println("\tId Tax: " + this.taxId);
		System.out.println("\tRevenue: " + this.revenueCompany);

	}
}
```

2.2 Mẫu thiết kế thứ 2: Abstract Factory

2.2.1 Sơ đồ thiết kế:



Abstract Factory là một trong những Creational pattern. Đây là phương pháp tạo ra Super-factory dùng để tạo ra các Factory khác (có thể gọi là Factory của các Factory). Trong pattern này một interface có nhiệm vụ tạo một Factory của các object có liên quan tới nhau mà không cần phải chỉ ra trực tiếp các class của object. Mỗi Factory được tạo ra có thể tạo ra các object bằng phương pháp giống như Factory pattern. Ta hiểu chúng nôm na là: Abstract Factory là một nhà máy lớn chứa nhiều nhà máy nhỏ, trong các nhày máy nhỏ đó có những phòng sản xuất, phòng đó tạo ra các sản phẩm khác nhau.

Một Abstract Factory bao gồm các thành phần cơ bản:

- Abstract Factory: Khai báo dạng interface hoặc abstract class chứa các phương thức để tạo đối tượng abstract.
- ConcreteFactory: Xây dựng, cài đặt các phương thức tạo các đối tượng cụ thể.
- AbstractProduct: Khai báo dạng interface hoặc abstract class để định nghĩa đối tượng abstract
- Product: Cài đặt các đối tượng củ thể, các phương thức quy định tại AbstractProduct.

Ở đây chúng em đã thành lập một công ty kinh doanh về thức ăn. Ban đầu công ty chỉ bán sữa gồm sữa nguyên chất (PureMil) và sữa có đường (SugarMilk). Với việc kinh doanh ngày càng thuận tiện nên quyết mở rộng thêm bán sữa chua (Yaour). Với việc công ty cũng kinh doanh Yaour theo loại có đường và nguyên chất. Tuy nhiên quá trình sản xuất Yaour và Milk theo từng TypeFood là khác nhau. Nên công ty tách ra nhà máy (Factory) : một dùng cho loại TypeFood là nguyên chất (PureFactory) và một cho có đường (SugarFactory) nhưng cả hai đều có thể tạo ra Milk và Yaour. Khi có nhu cầu chúng ta chỉ cần đến của hàng (FoodFactory) để đặt.

2.2.2 Code Demo:

- Đầu tiên ta cần tạo một Food Factory: nhằm để tạo ra các nhà máy sản xuất theo từng loại TypeFood

```
package Entity.AbstractFactory;

public class FoodFactory {
	private FoodFactory() {

	}

	public static FoodAbstractFactory getFactory(TypeFood typeFood) {
		switch(typeFood) {
			case SUGAR:
				return new SugarFactory();
			case PURE:
				return new PureFactory();
			default:
				throw new UnsupportedOperationException("This food isn't
supported ");
		}
	}
}
```

- Tiếp điến ta sẽ tạo FoodAbstractFactory ở đây ta sẽ chúng để có thể abstract các đối tượng ở đây là Milk hay Yaour

```
package Entity.AbstractFactory;

public abstract class FoodAbstractFactory {
	public abstract Yaour createYaour();
	public abstract Milk createMilk();
}
```

- Sau đó ta sẽ tạo nhà máy sản xuất thức ăn nguyên chất:

```
package Entity.AbstractFactory;

public class PureFactory extends FoodAbstractFactory{
	@Override
	public Yaour createYaour() {
		return new PureYaour();
	}

	@Override
	public Milk createMilk() {
		return new PureMilk();
	}
}
```

- Tạo nhà máy sản xuất thức ăn có đường:

```
package Entity.AbstractFactory;

public class SugarFactory extends FoodAbstractFactory{
	@Override
	public Yaour createYaour() {
		return new SugarYaour();
	}

	@Override
	public Milk createMilk() {
		return new SugarMilk();
	}
}
```

- Tiếp tục ta sẽ tạo các interface là Milk sau đó là các loại milk: PureMilk, SugarMilk

```
package Entity.AbstractFactory;

public interface Milk {
	void create();
}
```

```
package Entity.AbstractFactory;

public class PureMilk implements Milk{

	@Override
	public void create() {
		// TODO Auto-generated method stub
		System.out.println(" +++++++ Create PureMilk +++++++ ");
	}

}
```

```
package Entity.AbstractFactory;

public class SugarMilk implements Milk{

	@Override
	public void create() {
		// TODO Auto-generated method stub
		System.out.println(" +++++++  Create SugarMilk +++++++ ");
	}

}
```

- Tương tự: ta tạo interface là Yaour sau đó tạo các loại đối tượng PureYaour và SugarYaour

```
package Entity.AbstractFactory;

public interface Yaour {
	void create();
}
```

```
package Entity.AbstractFactory;

public class PureYaour implements Yaour{

	@Override
	public void create() {
		// TODO Auto-generated method stub
		System.out.println(" +++++++ Create PureYaour +++++++ ");
	}

}
```

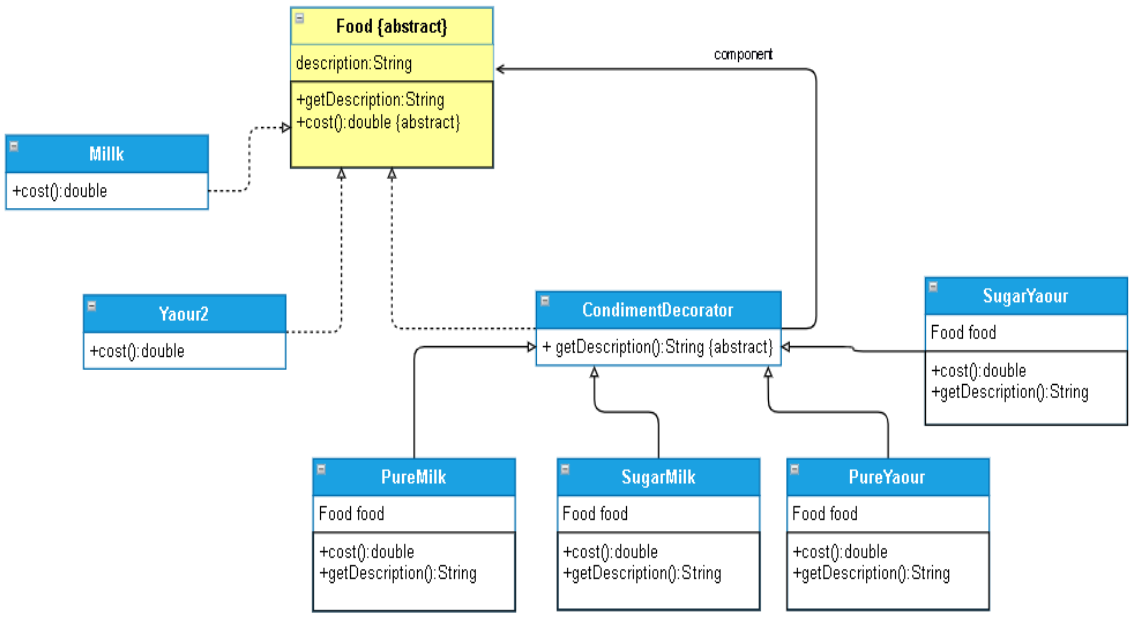
```
package Entity.AbstractFactory;

public class SugarYaour implements Yaour{

	@Override
	public void create() {
		// TODO Auto-generated method stub
		System.out.println("+++++++ Create SugarYaour +++++++ ");
	}

}
```

- Ta sẽ dùng kiểu enum để định nghĩa TypeFood:

```
package Entity.AbstractFactory;

public enum TypeFood {
	PURE, SUGAR
}
```

2.3 Mẫu thiết kế thứ 3: Decorator pattern

2.3.1 Sơ đồ thiết kế:



Decorator pattern là một trong những Pattern thuộc nhóm cấu trúc (Structural Pattern). Nó cho phép người dùng thêm chức năng mới vào đối tượng hiện tại mà không muốn ảnh hưởng đến các đối tượng khác. Kiểu thiết kế này có cấu trúc hoạt động như một lớp bao bọc (wrap) cho lớp hiện có. Mỗi khi cần thêm tính năng mới, đối tượng hiện có được wrap trong một đối tượng mới (decorator class).

Decorator pattern sử dụng composition thay vì inheritance (thừa kế) để mở rộng đối tượng. Decorator pattern còn được gọi là Wrapper hay Smart Proxy

Các thành phần trong mẫu thiết kế Decorator:

- **Component**: là một interface quy định các method chung cần phải có cho tất cả các thành phần tham gia vào mẫu này.
- **ConcreteComponent** : là lớp hiện thực (implements) các phương thức của Component.
- **Decorator** : là một abstract class dùng để duy trì một tham chiếu của đối tượng Component và đồng thời cài đặt các phương thức của Component interface.
- **ConcreteDecorator** : là lớp hiện thực (implements) các phương thức của Decorator, nó cài đặt thêm các tính năng mới cho Component.
- **Client** : đối tượng sử dụng Component

Ở đây công ty đang có 2 loại nguyên liệu chính là Milk và Yaour và các lớp đồ ăn được tạo ra từ nó la Pure Milk,Pure Yaour,Sugar Milk,Sugar Yaour.

2.3.2 Code Demo:

Đầu tiên tạo 1 abstract food với 2 method để mô tả tên sản phẩm với giá kèm chung.

```
package Entity.Decorator;

public abstract class Food {
	String description = "Unknown Food";

	public String getDescription() {
		return description;
	}

	public abstract double cost();

}
```

Tiếp tạo Decorator tiếp nhận mô tả các loại đồ ăn từ milk với yaour

```
package Entity.Decorator;

public abstract class CondimentDecorator extends Food {
	public abstract String getDescription();
}
```

Đây là 1 trong 2 lớp con kế thừa từ food để mô tả nguyên liệu chính là sữa kèm giá gốc.

```
package Entity.Decorator;

public class Millk extends Food {
	public Millk() {
		description = "Milk";

	}

	public double cost() {
		return 1.00;
	}
}
```

Đây là 1 trong 2 lớp con kế thừa từ food để mô tả nguyên liệu chính là yaour kèm giá gốc.

```
package Entity.Decorator;

public class Yaour2 extends Food {
	public Yaour2() {
		description = "Yaour";
	}

	public double cost() {
		return 2.00;
	}
}
```

Đây là lớp trang trí từ thành phẩm gốc là sữa or yaour

```
package Entity.Decorator;

public class PureMilk extends CondimentDecorator{
	Food food;

	public PureMilk (Food food) {
		this.food = food;
	}

	@Override
	public String getDescription() {
		// TODO Auto-generated method stub
		return food.getDescription() + ",Pure";
	}

	@Override
	public double cost() {
		// TODO Auto-generated method stub
		return .15 + food.cost();
	}

}
```

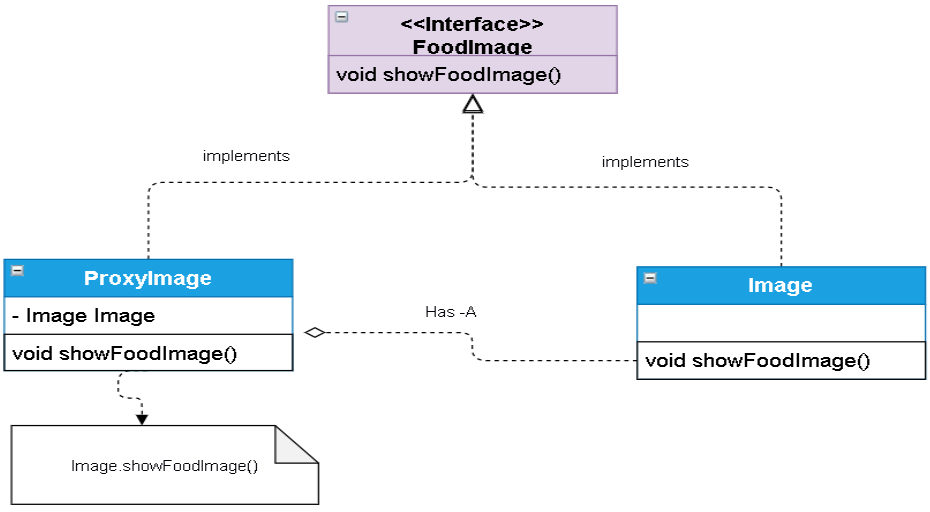
```java
package Entity.Decorator;

public class PureYaour extends CondimentDecorator{
	Food food;

	public PureYaour (Food food) {
		this.food = food;
	}

	@Override
	public String getDescription() {
		// TODO Auto-generated method stub
		return food.getDescription() + ",Pure";
	}

	@Override
	public double cost() {
		// TODO Auto-generated method stub
		return .15 + food.cost();
	}

}
```

```java
package Entity.Decorator;

public class SugarMilk extends CondimentDecorator{
	Food food;

	public SugarMilk (Food food) {
		this.food = food;
	}

	@Override
	public String getDescription() {
		// TODO Auto-generated method stub
		return food.getDescription() + ",Sugar";
	}

	@Override
	public double cost() {
		// TODO Auto-generated method stub
		return .19 + food.cost();
	}

}
```

```
package Entity.Decorator;

public class SugarYaour extends CondimentDecorator{
	Food food;

	public SugarYaour (Food food) {
		this.food = food;
	}

	@Override
	public String getDescription() {
		// TODO Auto-generated method stub
		return food.getDescription() + ",Sugar";
	}

	@Override
	public double cost() {
		// TODO Auto-generated method stub
		return .19 + food.cost();
	}

}
```

2.4 Mẫu thiết kế thứ 4: Proxy Pattern

2.4.1 Sơ đồ thiết kế:



Proxy Pattern là một trong những Pattern thuộc nhóm cấu trúc (Structural Pattern).

Proxy có nghĩa là "ủy quyền" hay "đại diện". Mục đích xây dựng Proxy pattern cũng chính vì muốn tạo ra một đối tượng sẽ ủy quyền, thay thế cho một đối tượng khác.

Proxy Pattern là mẫu thiết kế mà ở đó tất cả các truy cập trực tiếp đến một đối tượng nào đó sẽ được chuyển hướng vào một đối tượng trung gian (Proxy Class). Mẫu Proxy (người đại diện) đại diện cho một đối tượng khác thực thi các phương thức, phương thức đó có thể được định nghĩa lại cho phù hợp với múc đích sử dụng.

Các thành phần tham gia vào mẫu Proxy Pattern:

- **Subject** : là một interface định nghĩa các phương thực để giao tiếp với client. Đối tượng này xác định giao diện chung cho RealSubject và Proxy để Proxy có thể được sử dụng bất cứ nơi nào mà RealSubject mong đợi.
- **Proxy** : là một class sẽ thực hiện các bước kiểm tra và gọi tới đối tượng của class service thật để thực hiện các thao tác sau khi kiểm tra. Nó duy trì một tham chiếu đến RealSubject để Proxy có thể truy cập nó. Nó cũng thực hiện các giao diện tương tự như RealSubject để Proxy có thể được sử dụng thay cho RealSubject. Proxy cũng điều khiển truy cập vào RealSubject và có thể tạo hoặc xóa đối tượng này.
- **RealSubject** : là một class service sẽ thực hiện các thao tác thực sự. Đây là đối tượng chính mà proxy đại diện.
- **Client** : Đối tượng cần sử dụng RealSubject nhưng thông qua Proxy.

Ở đây ta có 1 interface FoodImage để thực thi chạy hình ảnh thông qua lấy hình ảnh từ Proxy và Image upload vào.

2.4.2 Code Demo:

Đầu tiên ta tạo 1 interface FoodImage để hiển thị hình ảnh thức ăn

```
package Entity.Proxy;

public interface FoodImage {
	void showFoodImage();
}
```

Sau đó tạo 1 class Image để thực thi hiển thị hình ảnh

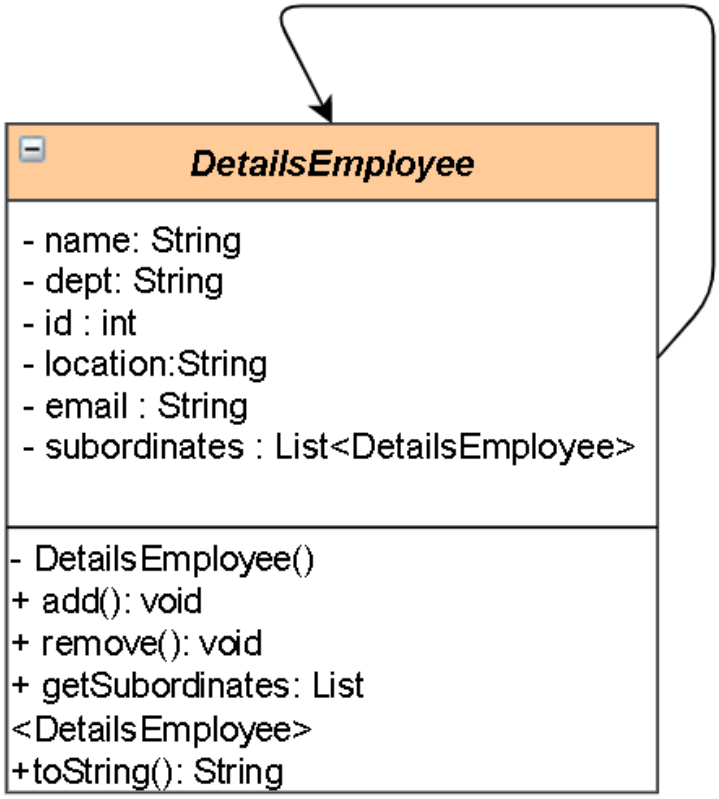
```
package Entity.Proxy;

public class Image implements FoodImage {
	private String url;

    public Image(String url) {
        this.url = url;
        System.out.println("Image loaded: " + this.url);
    }

    @Override
    public void showFoodImage() {
        System.out.println("FoodImage showed: " + this.url);
    }
}
```

Sau đó tạo 1 proxy để kiểm tra xem hình ảnh đầu vào có trùng nếu trùng thì sẽ trả ra đã tồn tại.

```
package Entity.Proxy;

public class ProxyImage implements FoodImage {
    private Image Image;
    private String url;

    public ProxyImage(String url) {
        this.url = url;
        System.out.println("Image unloaded: " + this.url);
    }

    @Override
    public void showFoodImage() {
        if (Image == null) {
            Image = new Image(this.url);
        } else {
            System.out.println("Image already existed: " + this.url);
        }
        Image.showFoodImage();
    }
}
```

2.5 Mẫu thiết kế thứ 5: Composite Pattern

2.5.1 Sơ đồ thiết kế:



Composite là một mẫu thiết kế thuộc nhóm cấu trúc (Structural Pattern). Composite Pattern là một sự tổng hợp những thành phần có quan hệ với nhau để tạo ra thành phần lớn hơn. Nó cho phép thực hiện các tương tác với tất cả đối tượng trong mẫu tương tự nhau.

Composite Pattern được sử dụng khi chúng ta cần xử lý một nhóm đối tượng tương tự theo cách xử lý 1 object. Composite pattern sắp xếp các object theo cấu trúc cây để diễn giải 1 phần cũng như toàn bộ hệ thống phân cấp. Pattern này tạo một lớp chứa nhóm đối tượng của riêng nó. Lớp này cung cấp các cách để sửa đổi nhóm của cùng 1

object. Pattern này cho phép Client có thể viết code giống nhau để tương tác với composite object này, bất kể đó là một đối tượng riêng lẻ hay tập hợp các đối tượng.

Một Composite Pattern bao gồm các thành phần cơ bản sau:

- **Base Component** : là một interface hoặc abstract class quy định các method chung cần phải có cho tất cả các thành phần tham gia vào mẫu này.
- **Leaf** : là lớp hiện thực (implements) các phương thức của Component. Nó là các object không có con.
- **Composite :** lưu trữ tập hợp các Leaf và cài đặt các phương thức của Base Component. Composite cài đặt các phương thức được định nghĩa trong interface Component bằng cách ủy nhiệm cho các thành phần con xử lý.
- **Client**: sử dụng Base Component để làm việc với các đối tượng trong Composite.

2.5.2 Code Demo:

```
package Entity.Composite;
import java.util.ArrayList;
import java.util.List;

public class DetailsEmployee {
   private String name;
   private String dept;
   private int id;
   private String location;
   private String email;
   private List<DetailsEmployee> subordinates;

   // constructor

   public DetailsEmployee(String name, String dept, int id, String location, String
email) {
       super();
       this.name = name;
       this.dept = dept;
       this.id = id;
       this.location = location;
       this.email = email;
       subordinates = new ArrayList<DetailsEmployee>();
}

public void add(DetailsEmployee e) {
      subordinates.add(e);
   }

   public void remove(DetailsEmployee e) {
      subordinates.remove(e);
   }

   public List<DetailsEmployee> getSubordinates(){
     return subordinates;
   }

   public String toString(){
      return ("Employee :[ Name : " + name + ", Dept : " + dept + ", Id :" + id + ",
Location " + location + ",Email" + email  + " ]");
   }
}
```

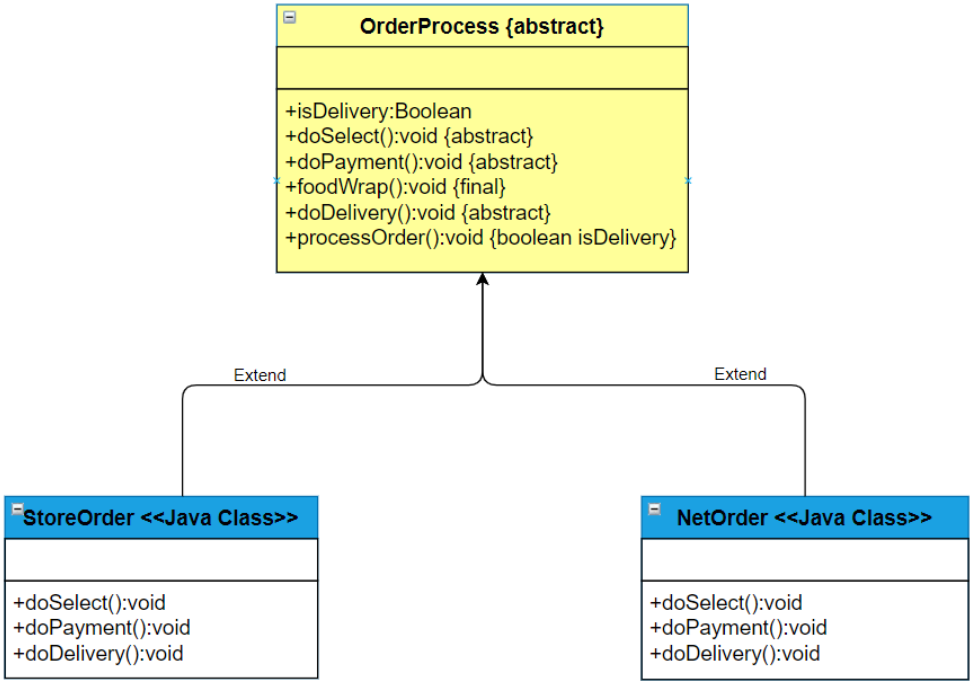
2.6 Mẫu thiết kế thứ 6: Template Method Pattern

- Template Method Pattern là một trong những Pattern thuộc nhóm hành vi (Behavior Pattern). Pattern này nói rằng "Định nghĩa một bộ khung của một thuật toán trong một chức năng, chuyển giao việc thực hiện nó cho các lớp con. Mẫu Template Method cho phép lớp con định nghĩa lại cách thực hiện của một thuật toán, mà không phải thay đổi cấu trúc thuật toán".

- Điều này có nghĩa là Template method giúp cho chúng ta tạo nên một bộ khung (template) cho một vấn đề đang cần giải quyết. Trong đó các đối tượng cụ thể sẽ có cùng các bước thực hiện, nhưng trong mỗi bước thực hiện đó có thể khác nhau. Điều này sẽ tạo nên một cách thức truy cập giống nhau nhưng có hành động và kết quả khác nhau.

- Template Method Pattern được sử dụng khá nhiều trong mô hình Abstract – Concrete Class. Khi chúng ta muốn các Concrete class tự thực thi xử lí theo cách của nó, nhưng đồng thời vẫn đảm bảo thuận theo những ràng buộc nhất định từ Abstract class. Ví dụ như ràng buộc về thứ tự các bước thực hiện, hay ràng buộc về dữ liệu đầu vào, đầu ra, …

2.6.1 Sơ đồ thiết kế:



2.6.2 Code Demo:

OrderProcess.java

```
package Entity.Template;

public abstract class OrderProcess {
       public boolean isDelivery;

    public abstract void doSelect();

    public abstract void doPayment();

    public final void foodWrap()
    {
        try
        {
            System.out.println("Food wrap successful");
        }
        catch (Exception e)
        {
            System.out.println("Food wrap unsuccessful");
        }
    }

    public abstract void doDelivery();
```

```
    public final void processOrder(boolean isDelivery)
    {
        doSelect();
        doPayment();
        if (isDelivery) {
            foodWrap();
        }
        doDelivery();
    }
}
```

NetOrder.java

```
package Entity.Template;

public class NetOrder extends OrderProcess{

	@Override
	public void doSelect() {
		// TODO Auto-generated method stub
		System.out.println("Item added to online shopping cart");
	  System.out.println("Get food wrap preference");
	  System.out.println("Get delivery address.");
	}

	@Override
	public void doPayment() {
		// TODO Auto-generated method stub
		System.out.println("Online Payment through Netbanking, card or
Paytm");
	}

	@Override
	public void doDelivery() {
		// TODO Auto-generated method stub
		System.out.println("Ship the item through post to delivery address");
	}

}
```

StoreOrder.java

```
package Entity.Template;

public class StoreOrder extends OrderProcess{

	@Override
	public void doSelect() {
		// TODO Auto-generated method stub
```

```
			System.out.println("Customer chooses the item from shelf.");
		}

		@Override
		public void doPayment() {
			// TODO Auto-generated method stub
			System.out.println("Pays at counter through cash/POS");
		}

		@Override
		public void doDelivery() {
			// TODO Auto-generated method stub
			System.out.println("Item delivered to in delivery counter.");
		}

}
```

2.7 Main:

2.7.1 Code Demo:

```
package Main;

import java.util.Scanner;

import Entity.AbstractFactory.FoodAbstractFactory;
import Entity.AbstractFactory.FoodFactory;
import Entity.AbstractFactory.Milk;
import Entity.AbstractFactory.TypeFood;
import Entity.AbstractFactory.Yaour;
import Entity.Composite.DetailsEmployee;
import Entity.Decorator.Food;
import Entity.Decorator.Millk;
import Entity.Decorator.PureMilk;
import Entity.Decorator.PureYaour;
import Entity.Decorator.SugarMilk;
import Entity.Decorator.SugarYaour;
import Entity.Decorator.Yaour2;
import Entity.Proxy.ProxyImage;
import Entity.Singleton.SingleCompany;
import Entity.Template.NetOrder;
import Entity.Template.OrderProcess;
import Entity.Template.StoreOrder;

public class Main {
	private static FoodAbstractFactory factory;

	public static void main(String[] args) {
		System.out.println("Welcome to Company Management");
		Scanner scanner = new Scanner(System.in);

		SingleCompany company = SingleCompany.getInstance();
		boolean flag=true;
		company.inputInfor(scanner);
		System.out.println("--------------------------------------------------
");

		do {
			int n;
			System.out.println("");
			showMenu();
			System.out.println("Enter your option: ");
			n = Integer.parseInt(scanner.nextLine());
			switch (n) {
			case 1:
				company.showInfor();
				break;
			case 2:
				boolean flag1 = true;
				do {
					System.out.println("");
					showTypeFood();
					System.out.println("Choose type food: ");
					int i = Integer.parseInt(scanner.nextLine());
					switch(i) {
```

```
                                case 1:
                                        FoodAbstractFactory factory =
FoodFactory.getFactory(TypeFood.PURE);
                                        boolean flag2 = true;
                                        do {
                                                System.out.println("");
                                                showFood();
                                                System.out.println("Choose food: ");
                                                int k =
Integer.parseInt(scanner.nextLine());
                                                switch(k) {
                                                case 1:
                                                        Milk milk = factory.createMilk();
                                                        milk.create();
                                                        break;
                                                case 2:
                                                        Yaour yaour =
factory.createYaour();
                                                        yaour.create();
                                                        break;
                                                case 0:
                                                        System.out.println("Back to
TypeFood");
                                                        flag2 = false;
                                                        break;
                                                default:
                                                        System.out.println("Invalid
input!!!!");
                                                        break;
                                                }
                                        } while(flag2);
                                        break;
                                case 2:
                                        FoodAbstractFactory factory1 =
FoodFactory.getFactory(TypeFood.SUGAR);
                                        boolean flag3 = true;
                                        do {
                                                System.out.println("");
                                                showFood();
                                                System.out.println("Choose food: ");
                                                int k =
Integer.parseInt(scanner.nextLine());
                                                switch(k) {
                                                case 1:
                                                        Milk milk = factory1.createMilk();
                                                        milk.create();
                                                        break;
                                                case 2:
                                                        Yaour yaour =
factory1.createYaour();
                                                        yaour.create();
                                                        break;
                                                case 0:
                                                        System.out.println("Back to
TypeFood");
                                                        flag3 = false;
                                                        break;
                                                default:
```

```
                                                System.out.println("Invalid
input!!!!");
                                                break;
                                          }
                                    } while(flag3);
                                    break;

                              case 0:
                                    System.out.println("Back to Menu");
                                    flag1 = false;
                                    break;
                              default:
                                    System.out.println("Not TypeFood!!!!");
                                    break;
                              }
                        }while(flag1);
                        break;

                  case 3:
                        Food food = new Millk();
                        food = new PureMilk(food);
                        System.out.println(food.getDescription()
                                    +"$" + food.cost());

                        Food food2 = new Millk();
                        food2 = new SugarMilk(food2);
                        System.out.println(food2.getDescription()
                                    +"$" + food2.cost());

                        Food food3 = new Yaour2();
                        food3 = new PureYaour(food3);
                        System.out.println(food3.getDescription()
                                    +"$" + food3.cost());

                        Food food4 = new Yaour2();
                        food4 = new SugarYaour(food4);
                        System.out.println(food4.getDescription()
                                    +"$" + food4.cost());

                        break;
                  case 4:


                        boolean flag7 = true;
                        do{
                        System.out.println("");
                        showImageFood();
                        System.out.println("Choose type food images: ");
                        int u = Integer.parseInt(scanner.nextLine());
                        switch(u) {
                        case 1:
                              System.out.println("----PureMilk---- ");
                          ProxyImage proxyImage = new ProxyImage("https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcTqNOggwZJXZRl4Y5ABJ8uKa9Q67WLoHJmDcyDYMXhbvgHP4YAr
aDX8J3NiOrURXB6ojfSYDN0&usqp=CAc");
                          proxyImage.showFoodImage();
                              break;
                        case 2:
```

```
                              System.out.println("----SugarMilk---- ");
                            ProxyImage proxyImage2 = new
ProxyImage("https://suachobeyeu.vn/upload/images/sua-tuoi-tiet-trung-vinamilk-co-
duong-bich-220-ml-1.jpg");
                            proxyImage2.showFoodImage();
                                  break;
                           case 3:
                                  System.out.println("----PureYaour---- ");
                            ProxyImage proxyImage3 = new
ProxyImage("https://www.purnatur.eu/images/producten/natuuryoghurt/624.jpg");
                            proxyImage3.showFoodImage();
                            break;
                           case 4:
                                  System.out.println("----SugarYaour---- ");
                            ProxyImage proxyImage4 = new
ProxyImage("https://www.purnatur.eu/images/producten/natuuryoghurt/624.jpg");
                            proxyImage4.showFoodImage();
                                  break;
                           case 0:
                                  System.out.println("Back to Menu");
                                  flag7 = false;
                                  break;
                           default:
                                  System.out.println("Invalid input!!!!");
                                  break;
                           }
                           } while(flag7);
                           break;

                    case 5:
                           boolean flag4 = true;
                           do{
                           System.out.println("");
                           showOrder();
                           System.out.println("Choose type food: ");
                           int i = Integer.parseInt(scanner.nextLine());
                           switch(i) {
                           case 1:

                                  boolean flag5 = true;
                                  do {
                                         System.out.println("");
                                         showProcessOrder();
                                         System.out.println("Choose food: ");
                                         int k = Integer.parseInt(scanner.nextLine());
                                         switch(k) {
                                                      case 1:
                                                            OrderProcess netOrder = new
NetOrder();
                                                       netOrder.processOrder(true);
                                                       System.out.println();
                                                            break;
                                                      case 2:
                                                            OrderProcess storeOrder =
new StoreOrder();
                                                       storeOrder.processOrder(true);
                                                            break;
                                                      case 0:
```

```
                                                        System.out.println("Back to
TypeFood");
                                                        flag5 = false;
                                                        break;
                                                  default:
                                                        System.out.println("Invalid
input!!!!");
                                                        break;
                                                  }
                                            } while(flag5);
                                            break;
                              case 2:
                                    boolean flag6 = true;
                                    do {
                                          System.out.println("");
                                          showProcessOrder();
                                          System.out.println("Choose food: ");
                                          int k = Integer.parseInt(scanner.nextLine());
                                          switch(k) {
                                                case 1:
                                                      OrderProcess netOrder = new
NetOrder();
                                                  netOrder.processOrder(true);
                                                  System.out.println();
                                                      break;
                                                case 2:
                                                      OrderProcess storeOrder =
new StoreOrder();
                                                  storeOrder.processOrder(true);
                                                      break;
                                                case 0:
                                                      System.out.println("Back to
TypeFood");
                                                      flag6 = false;
                                                      break;
                                                default:
                                                      System.out.println("Invalid
input!!!!");
                                                      break;
                                                }
                                            } while(flag6);
                                            break;
                              case 0:
                                    System.out.println("Back to Menu");
                                    flag4 = false;
                                    break;
                              default:
                                    System.out.println("Invalid input!!!!");
                                    break;
                              }
                              } while(flag4);
                              break;


                        case 6:
                                    DetailsEmployee e1 = new
DetailsEmployee("Long","CEO",1,"LeHoangLong@gmail.com","FoodCompany");
                              DetailsEmployee e2 = new
```

```
DetailsEmployee("Son","HeadSales",2,"TranNgocThaiSon@gmail.com","FoodCompany");
                        DetailsEmployee e3 = new
DetailsEmployee("Tai","HeadMarketing",3,"NguyenHuuTai@gmail.com","FoodCompany");

                        e1.add(e2);
                        e1.add(e3);

                        System.out.println(e1);

                        for (DetailsEmployee headEmployee : e1.getSubordinates()) {
                           System.out.println(headEmployee);

                           for (DetailsEmployee employee :
headEmployee.getSubordinates()) {
                              System.out.println(employee);
                           }
                        }
                        break;




                    case 0:
                           System.out.println("See you again!!!!");
                           flag1 = false;
                           break;
                    default:
                           System.out.println("Invalid input!!!!");
                           break;
                    }
             }while (flag);
      }


      static void showMenu() {
             int spaceNum = 20;
             System.out.println(
                           String.format("%" + spaceNum + "s", "") + "Company
Management" + String.format("%" + spaceNum + "s", ""));
             System.out.println("Menu:");
             System.out.println("\t1. Show information about company");
             System.out.println("\t2. Create Food");
             System.out.println("\t3. Show Menu of Food");
             System.out.println("\t4. Image of Food");
             System.out.println("\t5. Order Food");
             System.out.println("\t6. Show information about employee");
             System.out.println("\t0. Exit");
      }

      static void showTypeFood() {
             int spaceNum = 25;
             System.out.println(
                           String.format("%" + spaceNum + "s", "") + "Type Food" +
String.format("%" + spaceNum + "s", ""));
             System.out.println("\t\t1. Pure Food");
             System.out.println("\t\t2. Sugar Food");
             System.out.println("\t\t0. Back to Menu");
      }
```

```
	static void showFood() {
		int spaceNum = 25;
		System.out.println(
				String.format("%" + spaceNum + "s", "") + "Food" +
String.format("%" + spaceNum + "s", ""));
		System.out.println("\t\t\t1. Milk");
		System.out.println("\t\t\t2. Yaour");
		System.out.println("\t\t\t0. Back to Type Food");
	}

	static void showOrder() {
		int spaceNum = 25;
		System.out.println(
				String.format("%" + spaceNum + "s", "") + "Type Food" +
String.format("%" + spaceNum + "s", ""));
		System.out.println("\t\t\t1. Order Milk");
		System.out.println("\t\t\t2. Order Yaour");
		System.out.println("\t\t\t0. Back to Menu");
	}

	static void showProcessOrder() {
		int spaceNum = 30;
		System.out.println(
				String.format("%" + spaceNum + "s", "") + "Type Order" +
String.format("%" + spaceNum + "s", ""));
		System.out.println("\t\t\t1. Internet Order");
		System.out.println("\t\t\t2. Store Order");
		System.out.println("\t\t\t0. Back to Type Food");
	}

	static void showImageFood() {
		int spaceNum = 40;
		System.out.println(
				String.format("%" + spaceNum + "s", "") + "FoodImage" +
String.format("%" + spaceNum + "s", ""));
		System.out.println("\t\t1. Pure Milk");
		System.out.println("\t\t2. Sugar Milk");
		System.out.println("\t\t3. Pure Yaour");
		System.out.println("\t\t4. Sugar Yaour");
		System.out.println("\t\t0. Back to Menu");
	}


	static void BacktoMenu() {
		int spaceNum = 30;
		System.out.println(
				String.format("%" + spaceNum + "s", "") + "Food" +
String.format("%" + spaceNum + "s", ""));
		System.out.println("\t\t\t0. Back to Menu");
	}
}
```

2.8 Chạy Demo

```
Welcome to Company Management
Input name Company:
Tai
Input TaxId:
2
Input revenue Company:
2
--------------------------------------------------

                    Company Management
Menu:
        1. Show information about company
        2. Create Food
        3. Show Menu of Food
        4. Image of Food
        5. Order Food
        6. Show information about employee
        0. Exit
Enter your option:
```

```
Enter your option:
1
--ABOUT COMPANY--
        Name: Tai
        Id Tax: 2
        Revenue: 2.0

                    Company Management
Menu:
        1. Show information about company
        2. Create Food
        3. Show Menu of Food
        4. Image of Food
        5. Order Food
        6. Show information about employee
        0. Exit
```

```
                    Company Management
Menu:
        1. Show information about company
        2. Create Food
        3. Show Menu of Food
        4. Image of Food
        5. Order Food
        6. Show information about employee
        0. Exit
Enter your option:
2

                          Type Food
                1. Pure Food
                2. Sugar Food
                0. Back to Menu
Choose type food:
1


                         Food
                        1. Milk
                        2. Yaour
                        0. Back to Type Food
Choose food:
1
 +++++++ Create PureMilk +++++++

                         Food
                        1. Milk
                        2. Yaour
                        0. Back to Type Food
```

```
                          0. Back to Type Food
Choose food:
2
 +++++++ Create PureYaour +++++++

                           Food
                          1. Milk
                          2. Yaour
                          0. Back to Type Food
Choose food:
0
Back to TypeFood

                           Type Food
                  1. Pure Food
                  2. Sugar Food
                  0. Back to Menu
Choose type food:
2


                           Food
                          1. Milk
                          2. Yaour
                          0. Back to Type Food
Choose food:
1
 +++++++  Create SugarMilk +++++++

                           Food
                          1. Milk
                          2. Yaour
                          0. Back to Type Food
Choose food:
2
+++++++ Create SugarYaour +++++++

                           Food
                          1. Milk
```

```
                    2. Yaour
                    0. Back to Type Food
Choose food:
0
Back to TypeFood

                     Type Food
               1. Pure Food
               2. Sugar Food
               0. Back to Menu
Choose type food:
0
Back to Menu


                  Company Management
Menu:
        1. Show information about company
        2. Create Food
        3. Show Menu of Food
        4. Image of Food
        5. Order Food
        6. Show information about employee
        0. Exit
Enter your option:
3
Milk,Pure$1.15
Milk,Sugar$1.19
Yaour,Pure$2.15
Yaour,Sugar$2.19
```

```
                    Company Management
Menu:
        1. Show information about company
        2. Create Food
        3. Show Menu of Food
        4. Image of Food
        5. Order Food
        6. Show information about employee
        0. Exit
Enter your option:
4

                                FoodImage
            1. Pure Milk
            2. Sugar Milk
            3. Pure Yaour
            4. Sugar Yaour
            0. Back to Menu
Choose type food images:
1
----PureMilk----
Image unloaded: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTqNOggwZJXZRl4Y5ABJ8uKa9Q67WLoHJmDcyDYMXhbvgHP4YAraDX8J3NiOrURXB6ojfSYDN0&usqp=CAc
Image loaded: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTqNOggwZJXZRl4Y5ABJ8uKa9Q67WLoHJmDcyDYMXhbvgHP4YAraDX8J3NiOrURXB6ojfSYDN0&usqp=CAc
FoodImage showed: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTqNOggwZJXZRl4Y5ABJ8uKa9Q67WLoHJmDcyDYMXhbvgHP4YAraDX8J3NiOrURXB6ojfSYDN0&usqp=CAc
```

```
                    Company Management
Menu:
        1. Show information about company
        2. Create Food
        3. Show Menu of Food
        4. Image of Food
        5. Order Food
        6. Show information about employee
        0. Exit
Enter your option:
5

                         Type Food
                       1. Order Milk
                       2. Order Yaour
                       0. Back to Menu
Choose type food:
1
```

```
                    Type Order
                1. Internet Order
                2. Store Order
                0. Back to Type Food
Choose food:
1
Item added to online shopping cart
Get food wrap preference
Get delivery address.
Online Payment through Netbanking, card or Paytm
Food wrap successful
Ship the item through post to delivery address


                    Type Order
                1. Internet Order
                2. Store Order
                0. Back to Type Food
Choose food:
2
Customer chooses the item from shelf.
Pays at counter through cash/POS
Food wrap successful
Item delivered to in delivery counter.

                    Type Order
                1. Internet Order
                2. Store Order
                0. Back to Type Food
Choose food:
0
Back to TypeFood
```

```
                Type Food
              1. Order Milk
              2. Order Yaour
              0. Back to Menu
Choose type food:
0
Back to Menu
```

```
                    Company Management
Menu:
        1. Show information about company
        2. Create Food
        3. Show Menu of Food
        4. Image of Food
        5. Order Food
        6. Show information about employee
        0. Exit
Enter your option:
6
Employee :[ Name : Long, Dept : CEO, Id :1, Location LeHoangLong@gmail.com,EmailFoodCompany ]
Employee :[ Name : Son, Dept : HeadSales, Id :2, Location TranNgocThaiSon@gmail.com,EmailFoodCompany ]
Employee :[ Name : Tai, Dept : HeadMarketing, Id :3, Location NguyenHuuTai@gmail.com,EmailFoodCompany ]
```

```
                    Company Management
Menu:
        1. Show information about company
        2. Create Food
        3. Show Menu of Food
        4. Image of Food
        5. Order Food
        6. Show information about employee
        0. Exit
Enter your option:
0
See you again!!!!
```

TÀI LIỆU THAM KHẢO

https://en.wikipedia.org/wiki/Composite_pattern

https://www.tutorialspoint.com/design_pattern/composite_pattern.htm

https://en.wikipedia.org/wiki/Decorator_pattern

https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm

https://refactoring.guru/design-patterns/proxy

https://en.wikipedia.org/wiki/Proxy_pattern

https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm

https://gpcoder.com/4365-huong-dan-java-design-pattern-abstract-factory/

https://toihocdesignpattern.com/chuong-8-template-method-pattern.html

https://freetuts.net/template-pattern-trong-java-2878.html