МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»

Протокол лабораторної роботи №7

з дисципліни об'єктно-орієнтоване програмування

на тему: «Принцип абстракції. Додаткові засади ООП. Фабричний метод. Перерахування»

Виконала студентка групи

ІПЗ-212

Ярош Є.В.

Прийняв

Рудніченко М.Д.

Одеса, 2022

ЗМІСТ

[ВСТУП](#_30j0zll) 3

[ХІД РОБОТИ](#_8fy8djbnwuy9) 4

[ВИСНОВОК](#_2et92p0) 9

# ВСТУП

Мета роботи - ознайомитись з механізмом абстракції в ООП, вивчити додаткові принципи ООП, розібратися з використанням патерну «Фабричний метод», вивчити та реалізувати перерахування.

# ХІД РОБОТИ

1. Створіть клас Школа. Помістіть у цей клас інформацію про кількість учнів, а також поточну пору року (використовуючи перерахування). Кожна пора року повинна мати назву англійською (використовуйте конструктор).

Код програми:

import java.util.Scanner;

class SchoolExample {

private int schoolNumber;

private int numberOfStudents;

public void setSchoolNumber(int schoolNumber) {

this.schoolNumber = schoolNumber;

};

public int getSchoolNumber() {

return schoolNumber;

};

public void setNumberOfStudents(int numberOfStudents) {

this.numberOfStudents = numberOfStudents;

};

public int getNumberOfStudents() {

return numberOfStudents;

};

enum Season {

*Winter*("Now is winter and we aren't studying"),

*Spring*("Now is spring and we are studying"),

*Summer*("Now is summer and we aren't studying"),

*Autumn*("Now is autumn and we are studying");

private String code;

Season(String code) {

this.code = code;

}

public String getCode() { return code;}

};

public void PrintResults() {

System.*out*.println("The school number " + getSchoolNumber() + " has " + getNumberOfStudents() + " students");

System.*out*.println(Season.*Winter*.getCode());

System.*out*.println(Season.*Spring*.getCode());

System.*out*.println(Season.*Summer*.getCode());

System.*out*.println(Season.*Autumn*.getCode());

};

}

public class Main {

public static void main(String [] args) {

SchoolExample mySchool = new SchoolExample();

Scanner in = new Scanner(System.*in*);

System.*out*.println("Enter the school number: ");

mySchool.setSchoolNumber(in.nextInt());

System.*out*.println("Enter the number of students: ");

mySchool.setNumberOfStudents(in.nextInt());

in.close();

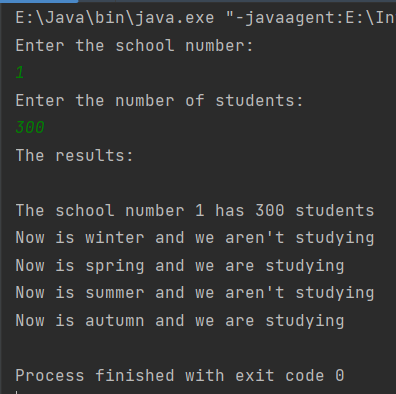
System.*out*.println("The results: \n");

mySchool.PrintResults();

};

}

Результат роботи програми наведено на рис.1.



Риcунок 1 – Результат роботи програми.

2. Попрацювати над відкриттям ще однієї кав'ярні. Вона може бути виконана в англійському або іспанському стилі. Або навіть у стилі космічного корабля. Додамо харчових барвників у каву, щоб блищало, і взагалі, кава буде просто космос!

Код програми:

abstract class Coffee {

public void makeCoffee() {

}

public void pourIntoCup() {

}

public void addSomeSpecial() {

}

}

class Americano extends Coffee {}

class Cappuccino extends Coffee {}

class CaffeLatte extends Coffee {}

class Espresso extends Coffee {}

class ItalianStyleAmericano extends Coffee {}

class ItalianStyleCappuccino extends Coffee {}

class ItalianStyleCaffeLatte extends Coffee {}

class ItalianStyleEspresso extends Coffee {}

class AmericanStyleAmericano extends Coffee {

@Override

public void addSomeSpecial() {

System.*out*.println("The use of our secret secret originated in South America…");

}

}

class AmericanStyleCappuccino extends Coffee {}

class AmericanStyleCaffeLatte extends Coffee {}

class AmericanStyleEspresso extends Coffee {}

class LunarStyleAmericano extends Coffee {

@Override

public void addSomeSpecial() {

System.*out*.println("With the addition of moon dust...");

}

}

class LunarStyleCappuccino extends Coffee {

@Override

public void addSomeSpecial() {

System.*out*.println("Adding a drop of sunlight...");

}

}

class LunarStyleCaffeLatte extends Coffee {

@Override

public void addSomeSpecial() {

System.*out*.println("Adding some stars...");

}

}

class LunarStyleEspresso extends Coffee {

@Override

public void addSomeSpecial() {

System.*out*.println("Adding our secret ingredient...");

}

}

enum CoffeeType {

*ESPRESSO*,

*AMERICANO*,

*CAFFE\_LATTE*,

*CAPPUCCINO*

}

class SimpleCoffeeFactory {

public Coffee createCoffee (CoffeeType type) {

Coffee coffee = null;

switch (type) {

case *AMERICANO*:

coffee = new Americano();

break;

case *ESPRESSO*:

coffee = new Espresso();

break;

case *CAPPUCCINO*:

coffee = new Cappuccino();

break;

case *CAFFE\_LATTE*:

coffee = new CaffeLatte();

break;

}

return coffee;

}

}

abstract class CoffeeShop {

public Coffee orderCoffee(CoffeeType type) {

Coffee coffee = createCoffee(type);

coffee.makeCoffee();

coffee.pourIntoCup();

coffee.addSomeSpecial();

System.*out*.println("Bon appetit!");

return coffee;

}

protected abstract Coffee createCoffee(CoffeeType type);

}

class ItalianCoffeeShop extends CoffeeShop {

@Override

public Coffee createCoffee (CoffeeType type) {

Coffee coffee = null;

switch (type) {

case *AMERICANO*:

coffee = new ItalianStyleAmericano();

break;

case *ESPRESSO*:

coffee = new ItalianStyleEspresso();

break;

case *CAPPUCCINO*:

coffee = new ItalianStyleCappuccino();

break;

case *CAFFE\_LATTE*:

coffee = new ItalianStyleCaffeLatte();

break;

}

return coffee;

}

}

class AmericanCoffeeShop extends CoffeeShop {

@Override

public Coffee createCoffee (CoffeeType type) {

Coffee coffee = null;

switch (type) {

case *AMERICANO*:

coffee = new AmericanStyleAmericano();

break;

case *ESPRESSO*:

coffee = new AmericanStyleEspresso();

break;

case *CAPPUCCINO*:

coffee = new AmericanStyleCappuccino();

break;

case *CAFFE\_LATTE*:

coffee = new AmericanStyleCaffeLatte();

break;

}

return coffee;

}

}

class LunarCoffeeShop extends CoffeeShop {

@Override

public Coffee createCoffee (CoffeeType type) {

Coffee coffee = null;

switch (type) {

case *AMERICANO*:

coffee = new LunarStyleAmericano();

break;

case *ESPRESSO*:

coffee = new LunarStyleEspresso();

break;

case *CAPPUCCINO*:

coffee = new LunarStyleCappuccino();

break;

case *CAFFE\_LATTE*:

coffee = new LunarStyleCaffeLatte();

break;

}

return coffee;

}

}

public class Main {

public static void main(String[] args) {

System.*out*.println("Welcome to <<Lunar ship>>, my own open-space CoffeeShop!");

CoffeeShop lunarCoffeeShop = new LunarCoffeeShop();

CoffeeShop americanCoffeeShop = new AmericanCoffeeShop();

lunarCoffeeShop.orderCoffee(CoffeeType.*AMERICANO*);

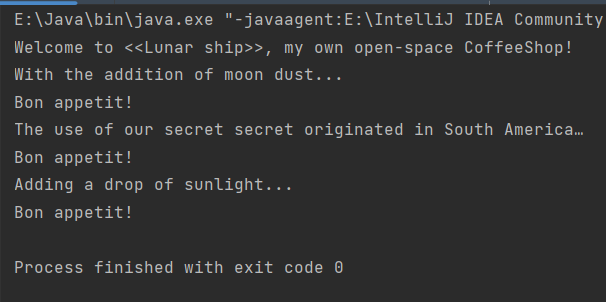
americanCoffeeShop.orderCoffee(CoffeeType.*AMERICANO*);

lunarCoffeeShop.orderCoffee(CoffeeType.*CAPPUCCINO*);

}

}

Результат роботи програми наведено на рис.2.



Риcунок 2 – Результат роботи програми.

3. На минулій лекції у вас було завдання створити віртуальний суші-бар або віртуальну піцерію. Твоя задача — не стоять на месте. Сьогодні ви дізналися, як за допомогою шаблону фабричний метод можна досягти успіху. Пора скористатися цими знаннями і розширити власний бізнес

Код програми:

abstract class Sushi {

public void prepareRiceOrNori() {

}

public void prepareFilling() {

}

public void serving() {

}

}

enum SushiType {

*MAKI\_ROLL*,

*TEMAKI*,

*NIGIRI*,

*SPRING\_ROLL*,

*URAMAKI*

}

class MakiRoll extends Sushi {}

class Temaki extends Sushi {}

class Nigiri extends Sushi {}

class SpringRoll extends Sushi {}

class Uramaki extends Sushi {}

class ChineseStyleMakiRoll extends Sushi {

@Override

public void serving() {

System.*out*.println("Adding vasabi...");

}

}

class ChineseStyleTemaki extends Sushi {

@Override

public void serving() {

System.*out*.println("Adding vasabi...");

}

}

class ChineseStyleNigiri extends Sushi {

@Override

public void serving() {

System.*out*.println("Adding vasabi...");

}

}

class ChineseStyleSpringRoll extends Sushi {

@Override

public void serving() {

System.*out*.println("Adding vasabi...");

}

}

class ChineseStyleUramaki extends Sushi {

@Override

public void serving() {

System.*out*.println("Adding vasabi...");

}

}

class SimpleSushiFactory {

public Sushi createSushi (SushiType type) {

Sushi sushi = null;

switch (type) {

case *MAKI\_ROLL*:

sushi = new MakiRoll();

break;

case *TEMAKI*:

sushi = new Temaki();

break;

case *NIGIRI*:

sushi = new Nigiri();

break;

case *SPRING\_ROLL*:

sushi = new SpringRoll();

break;

case *URAMAKI*:

sushi = new Uramaki();

}

return sushi;

}

}

abstract class SushiShop {

public Sushi orderSushi(SushiType type) {

Sushi sushi = createSushi(type);

sushi.prepareRiceOrNori();

sushi.prepareFilling();

sushi.serving();

System.*out*.println("Bon appetit!");

return sushi;

}

protected abstract Sushi createSushi(SushiType type);

}

class TraditionalSushiShop extends SushiShop {

@Override

public Sushi createSushi (SushiType type) {

Sushi sushi = null;

switch (type) {

case *MAKI\_ROLL*:

sushi = new MakiRoll();

break;

case *TEMAKI*:

sushi = new Temaki();

break;

case *NIGIRI*:

sushi = new Nigiri();

break;

case *SPRING\_ROLL*:\_ROLL:

sushi = new SpringRoll();

break;

case *URAMAKI*:

sushi = new Uramaki();

break;

}

return sushi;

}

}

class ChineseSushiShop extends SushiShop {

@Override

public Sushi createSushi (SushiType type) {

Sushi sushi = null;

switch (type) {

case *MAKI\_ROLL*:

sushi = new ChineseStyleMakiRoll();

break;

case *TEMAKI*:

sushi = new ChineseStyleTemaki();

break;

case *NIGIRI*:

sushi = new ChineseStyleNigiri();

break;

case *SPRING\_ROLL*:

sushi = new ChineseStyleSpringRoll();

break;

case *URAMAKI*:

sushi = new ChineseStyleUramaki();

break;

}

return sushi;

}

}

public class Main {

public static void main(String[] args) {

System.*out*.println("Welcome to the SushiBar!");

SushiShop traditionalSushiShop = new TraditionalSushiShop();

SushiShop chineseSushiShop = new ChineseSushiShop();

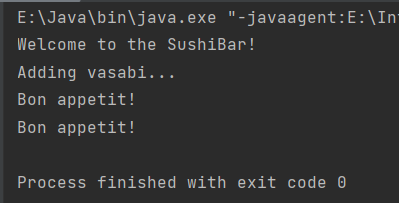
chineseSushiShop.orderSushi(SushiType.*TEMAKI*);

traditionalSushiShop.orderSushi(SushiType.*NIGIRI*);

}

}

Результат роботи програми наведено на рис.3.



Риcунок 3 – Результат роботи програми.

**Теоретичні питання:**

1. Що таке принцип абстракції? Наведіть приклад використання цього принципу під час проектування та розробки ГО-додатку.

Абстракція – спрощене опис чи виклад системи, у якому одні властивості і деталі виділяються, інші опускаються. Прикладом застосування цього принципу є створення абстрактного класу, який описує найбільш загальні властивості групи об'єктів, наприклад абстрактний клас трикутник може містити три поля, що вказують на координати вершин.

1. Що таке принцип модульності? Наведіть аргументи "за" використання цього принципу.

Модульність - властивість системи, яка була розкладена на внутрішньо зв'язкові, але слабко зв'язкові між собою модулі. Модульність програмного коду дозволяє значно зменшити час перекомпіляції при змінах, що вносяться лише в невелику кількість вихідних файлів, та спрощує групову розробку. Також це можливість заміни окремих компонентів (таких як jar-файли, so або dll бібліотеки) кінцевого програмного продукту без необхідності перескладання всього проекту (наприклад, розробка плагінів до вже готової програми).

1. Що таке принцип ієрархії в ОВП? Наведіть приклад використання ієрархій під час виконання лабораторних робіт.

Ієрархія – ранжована чи впорядкована система абстракцій. Принцип ієрархічності передбачає використання ієрархій розробки програмних систем. Прикладом може бути ієрархія класів у завданні з відкриттям кав'ярні, де кав'ярні у різних стилях є окремим випадком кав'ярні взагалі.

1. Що таке типізація в ОВП? Наведіть приклад типізації.

Типізація - це обмеження, що накладається на властивості об'єктів і перешкоджає взаємозамінності абстракцій різних типів. У лабораторній роботі ми визначили тип AmericanStyleAmericano, який перевизначає метод addSomeSpecial (), перевизначення, яке діє для цього типу, не діє для всіх інших типів.

1. Що таке принцип сталості? Чи створювали ви в попередніх лабораторних роботах об'єкти, що зберігаються?

Стійкість – властивість абстракції існувати в часі (незалежно від процесу, що породив даний програмний об'єкт) та у просторі (переміщуючись з адресного простору, в якому він був створений). У попередніх лабораторних роботах ми не створювали об'єкти, що зберігаються.

1. Що таке патерн "Фабричний метод". До якого принципу ОВП можна віднести цей патерн?

Паттерн фабричний метод визначає інтерфейс створення об'єкта, але дозволяє субкласам вибрати клас екземпляра, що створюється. Таким чином, Фабричний метод делегує операцію створення екземпляра субкласів. Цей патерн можна зарахувати до принципу абстракції.

1. Що таке перерахування? Навіщо потрібно використовувати перелік? До якого принципу ОВП можна віднести цей патерн?

Перерахування – це тип, посилальна змінна якого може приймати одне з кількох наперед визначених значень. Переліки є заміною набору констант. Перелік необхідно використовувати для того, щоб убезпечити типізацію та зберегти інкапсуляцію. Цей патерн можна зарахувати до принципу типізації.

# ВИСНОВОК

Протягом виконання лабораторної роботи, я ознайомилася з механізмом абстракції в ООП, вивчила додаткові принципи ООП, розібралася з використанням патерну «Фабричний метод», вивчила та реалізувала перерахування.