Университет ИТМО

Факультет ПИиКТ

Дисциплина: программирование

Лабораторная работа №3

Вариант 33207

Выполнил: Григорьев Александр Алексеевич,

группа P3130

Преподаватель: Блохина Елена Николаевна

г. Санкт-Петербург, 2021 год

Задание к лабораторной работе:

Описание предметной области, по которой должна быть построена объектная модель:

После бегства Ворчуна и Пилюлькина весь обслуживающий персонал больницы был занят лечением единственного больного -- Пульки, который, видя со стороны всех такое внимание к своей особе, совсем избаловался. То он требовал, чтобы ему на обед варили суп из конфет и кашу из мармелада; то заказывал котлеты из земляники с грибным соусом, хотя каждому известно, что таких котлет не бывает; то приказывал принести яблочное пюре, а когда приносили яблочное пюре, он говорил, что просил грушевого квасу; когда же приносили квас, он говорил, что квас воняет луком, или еще что-нибудь выдумывал. Все нянечки сбились с ног, исполняя его капризы. Они говорили, что у них спокон веку такого больного не было, что это сущее наказание, а не больной, и чтобы он выздоравливал уж поскорее, что ли.

Программа должна удовлетворять следующим требованиям:

1.Доработанная модель должна соответствовать принципам SOLID.

2.Программа должна содержать как минимум два интерфейса и один абстрактный класс (номенклатура должна быть согласована с преподавателем).

3.В разработанных классах должны быть переопределены методы equals(), toString() и hashCode().

4.Программа должна содержать как минимум один перечисляемый тип (enum).

Порядок выполнения работы:

1.Доработать объектную модель приложения.

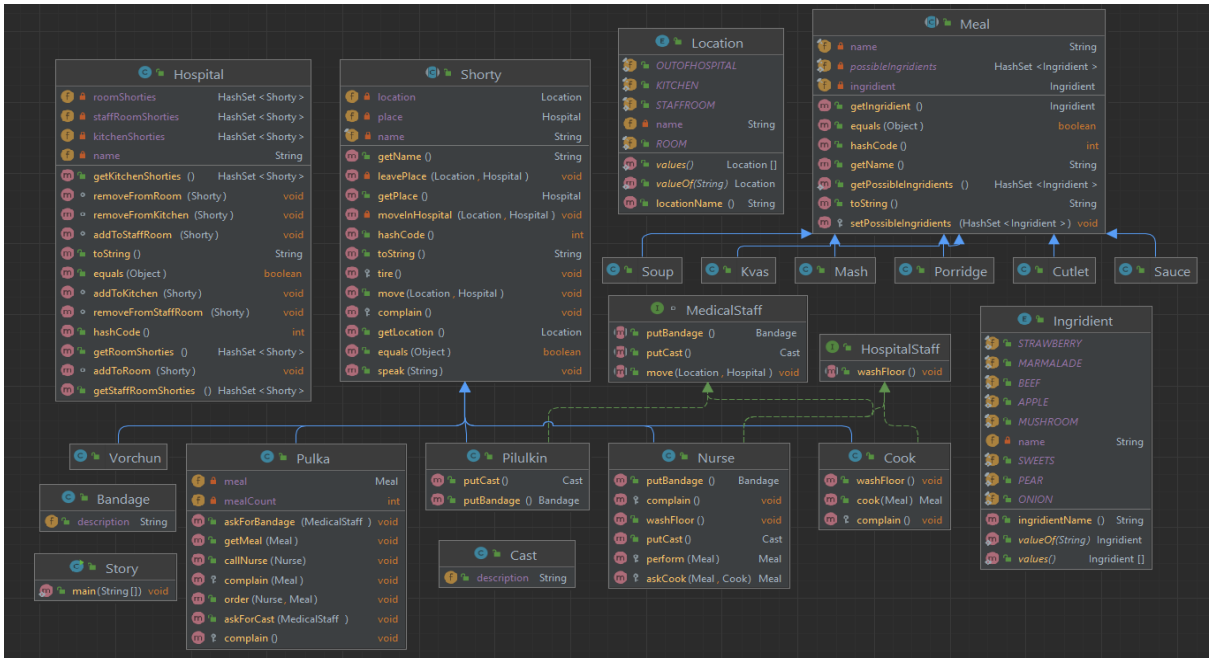
2.Перерисовать диаграмму классов в соответствии с внесёнными в модель изменениями.

3.Согласовать с преподавателем изменения, внесённые в модель.

4.Модифицировать программу в соответствии с внесёнными в модель изменениями.

Выполнение:

Объектная модель:



Исходный код:

Shorty.java

```
public class Story {
    public static void main(String[] args) {
        Hospital durka = new Hospital("durka");
        Vorchun vorchun = new Vorchun(Location.ROOM, durka);
        Pilulkin pilulkin = new Pilulkin(Location.STAFFROOM, durka);
        vorchun.move(Location.OUTOFHOSPITAL, durka);
        pilulkin.move(Location.OUTOFHOSPITAL,durka);
        Nurse nurse = new Nurse(Location.STAFFROOM, durka);
        Pulka pulka = new Pulka(Location.ROOM, durka);
        pulka.callNurse(nurse);
        Cook cook = new Cook(Location.KITCHEN, durka);
        pulka.order(nurse, new Soup(Ingridient.SWEETS));
        pulka.order(nurse, new Porridge(Ingridient.MARMALADE));
        pulka.order(nurse, new Cutlet(Ingridient.STRAWBERRY));
        pulka.order(nurse, new Mash(Ingridient.APPLE));
    }
}
```

Hospital.java

```
import java.util.HashSet;

import java.util.Objects;

public class Hospital {
    private String name;
    private HashSet<Shorty> kitchenShorties = new HashSet<>();
    private HashSet<Shorty> roomShorties = new HashSet<>();
    private HashSet<Shorty> staffRoomShorties = new HashSet<>();

    public Hospital(String hospName) {
        name = hospName;
    }

    void addToKitchen(Shorty shorty) {
        kitchenShorties.add(shorty);
    }

    void removeFromKitchen(Shorty shorty) {
        kitchenShorties.remove(shorty);
    }

    void addToStaffRoom(Shorty shorty) {
        staffRoomShorties.add(shorty);
    }

    void removeFromStaffRoom(Shorty shorty) {
        staffRoomShorties.remove(shorty);
    }

    void addToRoom(Shorty shorty) {
        roomShorties.add(shorty);
    }
```

```
    void removeFromRoom(Shorty shorty) {
        roomShorties.remove(shorty);
    }

    public HashSet<Shorty> getKitchenShorties() {
        return kitchenShorties;
    }

    public HashSet<Shorty> getRoomShorties() {
        return roomShorties;
    }

    public HashSet<Shorty> getStaffRoomShorties() {
        return staffRoomShorties;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Hospital)) return false;
        Hospital hospital = (Hospital) o;
        return Objects.equals(name, hospital.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }

    @Override
    public String toString() {
        return name;
    }
}
```

Shorty.java

```
import java.sql.SQLOutput;

import java.util.Objects;

public abstract class Shorty {
    private final String name;
    private Location location;
    private Hospital place;

    public Shorty(String n) {
        name = n;
        location = Location.OUTOFHOSPITAL;
    }

//    public Shorty(String n, Location notDefault) {
//        name = n;
//        LOCATION = notDefault;
//    }

    public Shorty(String n, Location notDefault, Hospital hosp) {
        name = n;
        location = notDefault;
```

```
        if (!(notDefault == Location.OUTOFHOSPITAL)) {
            place = hosp;
            moveInHospital(notDefault, hosp);
        }
    }


    public void move(Location newLocation, Hospital hosp) {
        if (newLocation != location) {
            if (newLocation == Location.OUTOFHOSPITAL) {
                System.out.println(getName() + " escapes from Hospital");
                if (place != null) {
                    leavePlace(newLocation, hosp);
                    place = null;
                }
            } else {
                place = hosp;
               // System.out.println(name + " moves from " + location.locationName() + " to " +
newLocation.locationName());
                leavePlace(newLocation, hosp);
                moveInHospital(newLocation, hosp);
            }
            location = newLocation;
        }
    }

    private void leavePlace(Location locat, Hospital hosp) {
        switch (locat) {
            case KITCHEN:
                hosp.removeFromKitchen(this);
            case ROOM:
                hosp.removeFromRoom(this);
            case STAFFROOM:
                hosp.removeFromStaffRoom(this);
        }
    }

    private void moveInHospital(Location locat, Hospital hosp) {
        switch (getLocation()) {
            case KITCHEN:
                hosp.addToKitchen(this);
            case ROOM:
                hosp.addToRoom(this);
            case STAFFROOM:
                hosp.addToStaffRoom(this);
        }
    }

    public void speak(String text) {
        System.out.println(text);
    }

    public String getName() {
        return name;
    }

    public Location getLocation() {
        return location;
```

```
    }

    public Hospital getPlace() {
        return place;
    }

    protected void complain() {
        System.out.print(this.getName() + ": I can't stand it no more. ");
    }

    protected void tire(){
        System.out.println(this.getName() + ": I'm incredibly tired!");
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Shorty)) return false;
        Shorty shorty = (Shorty) o;
        return Objects.equals(name, shorty.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }

    @Override
    public String toString() {
        return getName();
    }
}
```

Pulka.java

```
public class Pulka extends Shorty {
    private Meal meal;
    private int mealCount = 0;

    public Pulka() {
        super("Pulka");
    }

    public Pulka(Location notDefault, Hospital hosp) {
        super("Pulka", notDefault, hosp);
    }

    public void callNurse(Nurse nurse) {
        nurse.move(Location.ROOM, this.getPlace());
    }

    public void order(Nurse nurse, Meal meal) {
        System.out.println("Pulka: Cook me " + meal);
        getMeal(nurse.perform(meal));
        if (mealCount == 5) {
            nurse.complain();
            nurse.tire();
        }
```

```
        if (mealCount == 4) {
            order(nurse, new Kvas(Ingridient.PEAR));
        }
        //complain(this.meal);
    }

    public void getMeal(Meal meal) {
        System.out.println("Pulka gets " + meal);
        this.meal = meal;
        mealCount++;
        if (mealCount == 4 || mealCount == 5) {
            complain(meal);
        }
    }

    public void askForBandage(MedicalStaff medic) {
        medic.move(Location.ROOM, this.getPlace());
        medic.putBandage();
    }

    public void askForCast(MedicalStaff medic) {
        medic.move(Location.ROOM, this.getPlace());
        medic.putCast();
    }

    @Override
    protected void complain() {
        super.complain();
        System.out.println("This hospital is awful.");
    }

    protected void complain(Meal meal) {
        super.complain();
        if (mealCount == 5) {
            System.out.println("This " + meal.getName() + " stinks of onion");
        }
        if (mealCount == 4) {
            System.out.println("I asked for Pear Kvas");
        }
    }
}
```

Nurse.java

```
public class Nurse extends Shorty implements HospitalStaff, MedicalStaff {


    public Nurse() {
        super("Nurse");
    }

    public Nurse(Location notDefault, Hospital hosp) {

        super("Nurse", notDefault, hosp);
    }

    protected Meal perform(Meal meal) {
        boolean f = false;
```

```
        move(Location.KITCHEN, this.getPlace());
        for (Shorty shorty : getPlace().getKitchenShorties()) {
            if (shorty instanceof Cook) {
                f = true;
                askCook(meal, (Cook) shorty);
            }
        }
        move(Location.ROOM, this.getPlace());
        if (!f) {
            move(Location.ROOM, this.getPlace());
            //System.out.println(this.getName() + ": We have no cook!");
            return null;
        } else {
            return meal;
        }
    }

    protected Meal askCook(Meal meal, Cook c) {
       // System.out.println("Nurse: Cook "+ meal.getIngridient().ingridientName() + " " + meal.getName());
        return c.cook(meal);
    }

    @Override
    public void washFloor() {
        move(Location.STAFFROOM, getPlace());
        System.out.println(this.getName() + " washes the floor in StaffRoom");
    }

    @Override
    public Bandage putBandage() {
        return new Bandage();
    }

    @Override
    public Cast putCast() {
        return new Cast();
    }

    @Override
    protected void complain(){
        super.complain();
        System.out.println("What a terrible patient! He'd rather get well soon");
    }
}
```

Cook.java

```
public class Cook extends Shorty implements HospitalStaff{
    public Cook() {
        super("Cook");
    }

    public Cook(Location notDefault, Hospital hosp) {
        super("Cook", notDefault, hosp);
    }

    public Meal cook(Meal meal){
        if (!(meal.getPossibleIngridients().contains(meal.getIngridient()))){
```

```
            this.complain();
        }
        return meal;
    }

    @Override
    protected void complain() {
        super.complain();
        System.out.println("This meal doesn't exist");
    }

    @Override
    public void washFloor(){
        move(Location.KITCHEN, getPlace());
        System.out.println(this.getName() + " washes the floor in Kitchen");
    }
}
```

Pilulkin.java

```
public class Pilulkin extends Shorty implements MedicalStaff{
    public Pilulkin() {
        super("Pilulkin");
    }

    public Pilulkin(Location notDefault, Hospital hosp) {

        super("Pilulkin", notDefault, hosp);
    }

    @Override
    public Bandage putBandage(){
        return new Bandage();
    }

    @Override
    public Cast putCast(){
        return new Cast();
    }
}
```

Vorchun.java

```
public class Vorchun extends Shorty {
    public Vorchun() {
        super("Vorchun");
    }

    public Vorchun(Location notDefault, Hospital hosp) {

        super("Vorchun", notDefault, hosp);
    }
}
```

Location.java

```
public enum Location {
    KITCHEN("Kitchen"), ROOM("Room"), STAFFROOM("StaffRoom"), OUTOFHOSPITAL("OutOfHospital");
    private String name;

    Location(String locationName) {
        this.name = locationName;
```

```
    }

    public String locationName() {
        return name;
    }
}
```

Ingridient.java

```
public enum Ingridient {
    STRAWBERRY("Strawberry"), SWEETS("Sweets"), MARMALADE("Marmalade"),
    MUSHROOM("Mushroom"), APPLE("Apple"),
    PEAR("Pear"), ONION("Onion"), BEEF("Beef");
    private String name;

    Ingridient(String ingridientName) {
        this.name = ingridientName;
    }

    public String ingridientName() {
        return name;
    }
}
```

Cast.java

```
public class Cast {
    public String description = "гипс";
    public Cast(){
        description = "Гипс";
    }
}
```

Bandage.java

```
public class Bandage {
    public String description = "бинт";

    public Bandage() {
        description = "Бинт";
    }
}
```

Meal.java

```
import java.util.HashSet;
import java.util.Objects;

public abstract class Meal {
    private final Ingridient ingridient;
    private final String name;
    private static final HashSet<Ingridient> possibleIngridients = new HashSet<>();

    public Meal(String n, Ingridient ing) {
        name = n;
        ingridient = ing;
    }

    public String getName() {
        return name;
    }

    public Ingridient getIngridient() {
```

```
        return ingridient;
    }

    protected void setPossibleIngridients(HashSet<Ingridient> ingridients){
        for (Ingridient ing: ingridients){
            possibleIngridients.add(ing);
        }
    }

    public static HashSet<Ingridient> getPossibleIngridients() {
        return possibleIngridients;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Meal)) return false;
        Meal meal = (Meal) o;
        return ingridient == meal.ingridient && Objects.equals(name, meal.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }

    @Override
    public String toString() {
        return getIngridient().ingridientName() + " " +getName();
    }
}
```

Cutlet.java

```
import java.util.HashSet;

public class Cutlet extends Meal {
    public Cutlet(Ingridient ing) {
        super("Cutlet", ing);
        HashSet<Ingridient> ingridients = new HashSet<>();
        ingridients.add(Ingridient.BEEF);
        setPossibleIngridients(ingridients);
    }
}
```

Soup.java

```
import java.util.HashSet;

public class Soup extends Meal {
    public Soup(Ingridient ing) {
        super("Soup", ing);
        HashSet<Ingridient> ingridients = new HashSet<>();
        ingridients.add(Ingridient.SWEETS);
        setPossibleIngridients(ingridients);
    }
}
```

Porridge.java

```
import java.util.HashSet;
```

```
public class Porridge extends Meal {
    public Porridge(Ingridient ing) {
        super("Porridge", ing);
        HashSet<Ingridient> ingridients = new HashSet<>();
        ingridients.add(Ingridient.MARMALADE);
        setPossibleIngridients(ingridients);
    }
}
```

Mash.java

```
import java.util.HashSet;

public class Mash extends Meal {
    public Mash(Ingridient ing) {
        super("Mash", ing);
        HashSet<Ingridient> ingridients = new HashSet<>();
        ingridients.add(Ingridient.APPLE);
        ingridients.add(Ingridient.PEAR);
        setPossibleIngridients(ingridients);
    }
}
```

MedicalStaff.java

```
interface MedicalStaff {
    public Bandage putBandage();

    public Cast putCast();

    public void move(Location newLocation, Hospital hosp);
}
```

HospitalStaff.java

```
public interface HospitalStaff {
    public void washFloor();
}
```

Результат выполнения программы:

```
Vorchun escapes from Hospital
Pilulkin escapes from Hospital
Pulka: Cook me Sweets Soup
Pulka gets Sweets Soup
Pulka: Cook me Marmalade Porridge
Pulka gets Marmalade Porridge
Pulka: Cook me Strawberry Cutlet
Cook: I can't stand it no more. This meal doesn't exist
Pulka gets Strawberry Cutlet
Pulka: Cook me Apple Mash
Pulka gets Apple Mash
Pulka: I can't stand it no more. I asked for Pear Kvas
Pulka: Cook me Pear Kvas
Pulka gets Pear Kvas
Pulka: I can't stand it no more. This Kvas stinks of onion
Nurse: I can't stand it no more. What a terrible patient! He'd rather get well soon
Nurse: I'm incredibly tired!
```

Вывод: Во время выполнения данной лабораторной работы я научился применять принципы SOLID на практике, изучил понятия абстрактного класса, интерфейса и перечисления, а так же столкнулся с основными плюсом и минусом SOLID: очень сложно начать проект без предварительной разработки архитектуры и представления необходимых зависимостей, однако гораздо легче расширять и дополнять готовый проект, в котором используются принципы SOLID.