Университет ИТМО

Факультет систем управления и робототехники

Программирование

Лабораторная работа №3

Вариант №31455

Преподаватель:

Сорокин Роман Борисович

Выполнил:

Богданов Денис Андреевич

Группа:

R3142

Текст задания

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

Увидев, что падение не причинило Пончику никакого вреда, Незнайка затворил дверь и сказал с веселой улыбкой: Незнайка начал нажимать разные кнопки и открывать дверцы стенных шкафов, термостатов и холодильников, на полках которых хранились самые разнообразные пищевые продукты. Пончик, однако, был так сильно расстроен, что даже вид продуктов его не радовал.

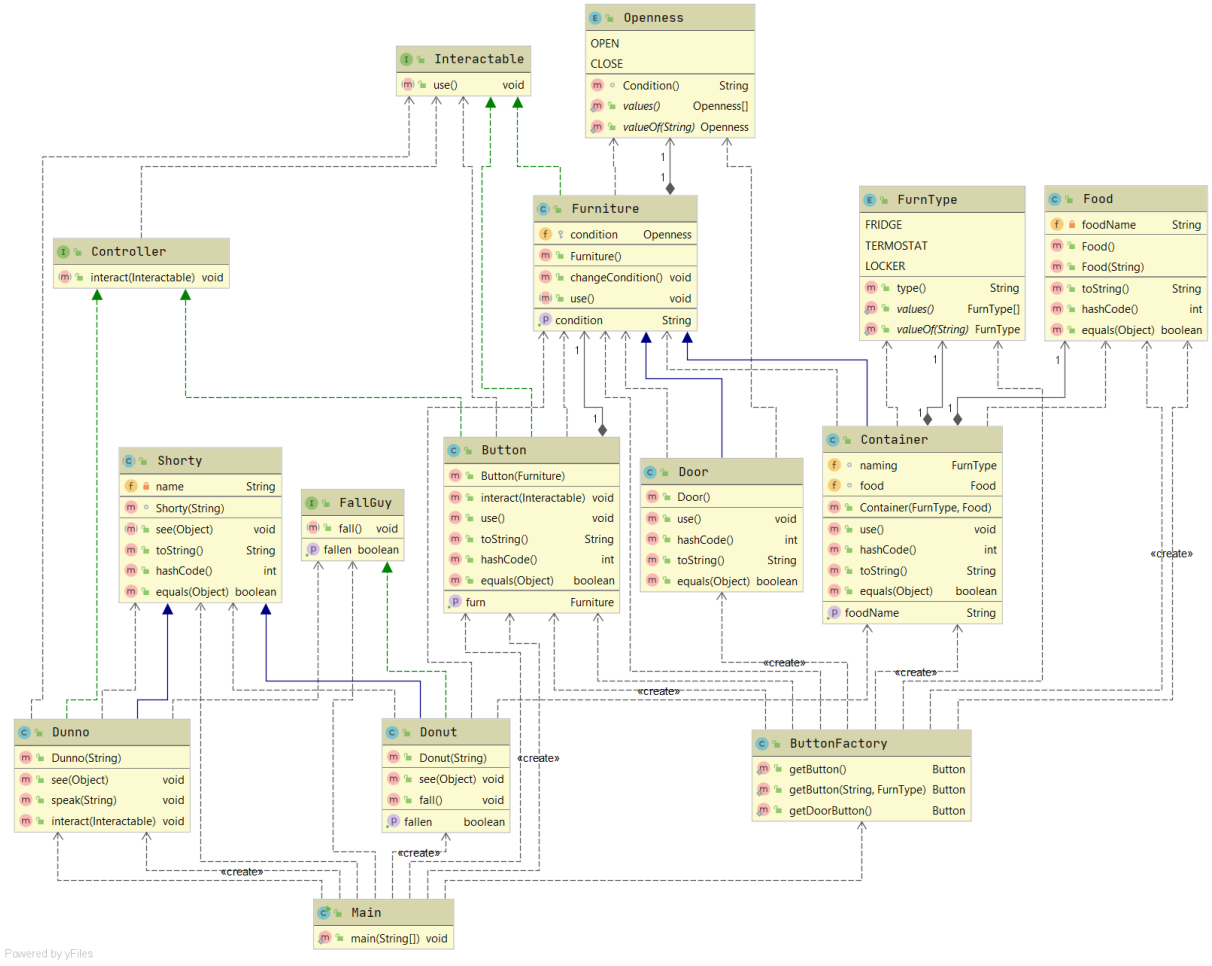
Программа должна удовлетворять следующим требованиям:

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.

Диаграмма классов



Исходный код программы

Main.java:

```
public class Main {
    public static void main(String[] args){
        Shorty dun = new Dunno("Незнайка");
        Shorty don = new Donut("Пончик");
        Button doorButton = ButtonFactory.getDoorButton();
        Button[] buttons = new Button[5];
        for (int i = 0; i < buttons.length; i++){
            buttons[i] = ButtonFactory.getButton();
        }
        ((FallGuy)don).fall();
        dun.see(don);
        ((Dunno)dun).interact(doorButton);
        ((Dunno)dun).speak("");
        for (int i = 0; i < buttons.length; i++){
            ((Dunno)dun).interact(buttons[i]);
            don.see(buttons[i].getFurn());
        }
    }
}
```

Shorty.java:

```
public abstract class Shorty{
    private String name;
    Shorty(String name){
        this.name = name;
    }
    public abstract void see(Object A);
    @Override
    public String toString(){
        return name;
    }
    @Override
    public int hashCode(){
        return (name.hashCode() * 2);
    }
    @Override
    public boolean equals(Object obj){
        return (obj instanceof Shorty) && (obj.hashCode() == this.hashCode());
    }
}
```

FallGuy.java:

```
public interface FallGuy {
    void fall();
    boolean isFallen();
}
```

Controller.java:

```
public interface Controller {
    void interact(Interactable A);
}
```

Interactable.java:

```
public interface Interactable {
    void use();
}
```

Food.java:

```
public class Food {
    private String foodName;
    public Food(){
        foodName = "пищевой продукт";
    }
    public Food(String foodName){
        this.foodName = foodName;
    }
    @Override
    public String toString(){
        return foodName;
    }
    @Override
    public int hashCode(){
        return foodName.hashCode();
    }

    @Override
    public boolean equals(Object A) {
        return (A instanceof Food) && (A.hashCode() == this.hashCode());
    }
}
```

FurnType.java:

```
public enum FurnType {
    FRIDGE{
        @Override
        public String type(){
            return "холодильник";
        }
    },
    TERMOSTAT{
        @Override
        public String type(){
            return "термостат";
        }
    },
    LOCKER{
        @Override
        public String type(){
            return "шкаф";
        }
    };
    public String type(){
        return "фурнитура";
    }
}
```

Openness.java:

```
public enum Openness {
    OPEN{
        @Override
        String Condition() {
            return "открыт";
        }
    },
    CLOSE{
        @Override
        String Condition() {
            return "закрыт";
        }
    };
    String Condition(){
        return "неопределен";
```

```
    }
}
```

Furniture.java:

```
public abstract class Furniture implements Interactable{
    protected Openness condition;
    public Furniture(){
        condition = Openness.CLOSE;
    }
    public void changeCondition() {
        switch (this.condition){
            case OPEN:
                this.condition = Openness.CLOSE;
                break;
            case CLOSE:
                this.condition = Openness.OPEN;
                break;
        }
    }
    public String getCondition(){
        return condition.Condition();
    }
    @Override
    public abstract void use();
}
```

Door.java:

```
public class Door extends Furniture{
    public Door(){
        condition = Openness.OPEN;
    }
    @Override
    public void use(){
        this.changeCondition();
        System.out.println(this.toString() + " " + this.getCondition() + "a");
    }
    @Override
    public int hashCode(){
        int result = 0;
        if (this.condition == Openness.OPEN)
            result = 1;
        return result;
    }
    @Override
    public String toString() {
        return "Дверь";
    }
    @Override
    public boolean equals(Object obj) {
        return (obj instanceof Door) && (obj.hashCode() == this.hashCode());
    }
}
```

Container.java:

```
public class Container extends Furniture{
    FurnType naming;
    Food food;
    public Container(FurnType naming, Food food){
        super();
        this.naming = naming;
        this.food = food;
    }
    public String getFoodName() {
        return food.toString();
```

```
    }
    @Override
    public void use(){
        this.changeCondition();
        System.out.println(this.naming.type() + " " + this.getCondition());
    }
    @Override
    public int hashCode(){
        int i = 0;
        switch(naming){
            case LOCKER:
                i += 1;
                break;
            case TERMOSTAT:
                i += 2;
                break;
            case FRIDGE:
                i += 3;
                break;
        }
        if (food.toString().equals("пищевой продукт"))
            i += 1;
        return i;
    }
    @Override
    public String toString(){
        return (naming.type() + " с " + food.toString());
    }
    @Override
    public boolean equals(Object obj){
        return (obj instanceof Container) && (obj.hashCode() == this.hashCode());
    }
}
```

Button.java:

```
public class Button implements Controller, Interactable{
    private Furniture furn;
    public Button(Furniture furn){
        this.furn = furn;
    }
    @Override
    public void interact(Interactable A){
        A.use();
    }
    public void use(){
        System.out.print("Кнопку нажали: ");
        interact(furn);
    }
    public Furniture getFurn() {
        return furn;
    }
    @Override
    public String toString(){
        return "Кнопка от " + furn.toString();
    }
    @Override
    public int hashCode(){
        return (furn.hashCode() * 2);
    }
    @Override
    public boolean equals(Object obj){
        return (obj instanceof Button) && (obj.hashCode() == this.hashCode());
    }
}
```

ButtonFactory.java:

```
public class ButtonFactory {
    public static Button getButton(){
        return ButtonFactory.getButton("пищевой продукт", FurnType.FRIDGE);
    }
    public static Button getButton(String foodName, FurnType ftype){
        Food food = new Food(foodName);
        Furniture furniture = new Container(ftype, food);
        Button button = new Button(furniture);
        return button;
    }
    public static Button getDoorButton(){
        Furniture door = new Door();
        Button button = new Button(door);
        return button;
    }
}
```

Dunno.java:

```
public class Dunno extends Shorty implements Controller{
    public Dunno(String name){
        super(name);
    }
    @Override
    public void see(Object A) {
        System.out.print(this.toString() + " видит: ");
        if (A instanceof FallGuy)
            if (((FallGuy)A).isFallen())
                System.out.println(A.toString() + " упал");
            else
                System.out.println(A.toString() + " стоит");
        else
            System.out.println(A.toString());
    }
    public void speak(String message){
        System.out.println(this.toString() + " говорит с широкой улыбкой: " + message);
    }
    @Override
    public void interact(Interactable A){
        A.use();
    }
}
```

Donut.java:

```
public class Donut extends Shorty implements FallGuy{
    boolean condition = false;
    public Donut(String name){
        super(name);
    }
    @Override
    public void see(Object A){
        System.out.print(this.toString() + " видит ");
        if ((A instanceof Container) && (((Furniture)A).getCondition().equals("открыт")))
            System.out.println(((Container)A).getFoodName() + ", но даже это его не
радует");
        else
            System.out.println(A.toString());
    }
    @Override
    public void fall(){
        condition = true;
        System.out.println(this.toString() + " падает");
    }
```

```
    @Override
    public boolean isFallen() {
        return condition;
    }
}
```

Пример работы программы

Пончик падает

Незнайка видит: Пончик упал

Кнопку нажали: Дверь закрыта

Незнайка говорит с широкой улыбкой:

Кнопку нажали: холодильник открыт

Пончик видит пищевой продукт, но даже это его не радует

Кнопку нажали: холодильник открыт

Пончик видит пищевой продукт, но даже это его не радует

Кнопку нажали: холодильник открыт

Пончик видит пищевой продукт, но даже это его не радует

Кнопку нажали: холодильник открыт

Пончик видит пищевой продукт, но даже это его не радует

Кнопку нажали: холодильник открыт

Пончик видит пищевой продукт, но даже это его не радует

Выводы

В процессе выполнения лабораторной работы я ознакомился с принципами SOLID, ознакомился с абстрактными классами, интерфейсами и перечислениями.