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.

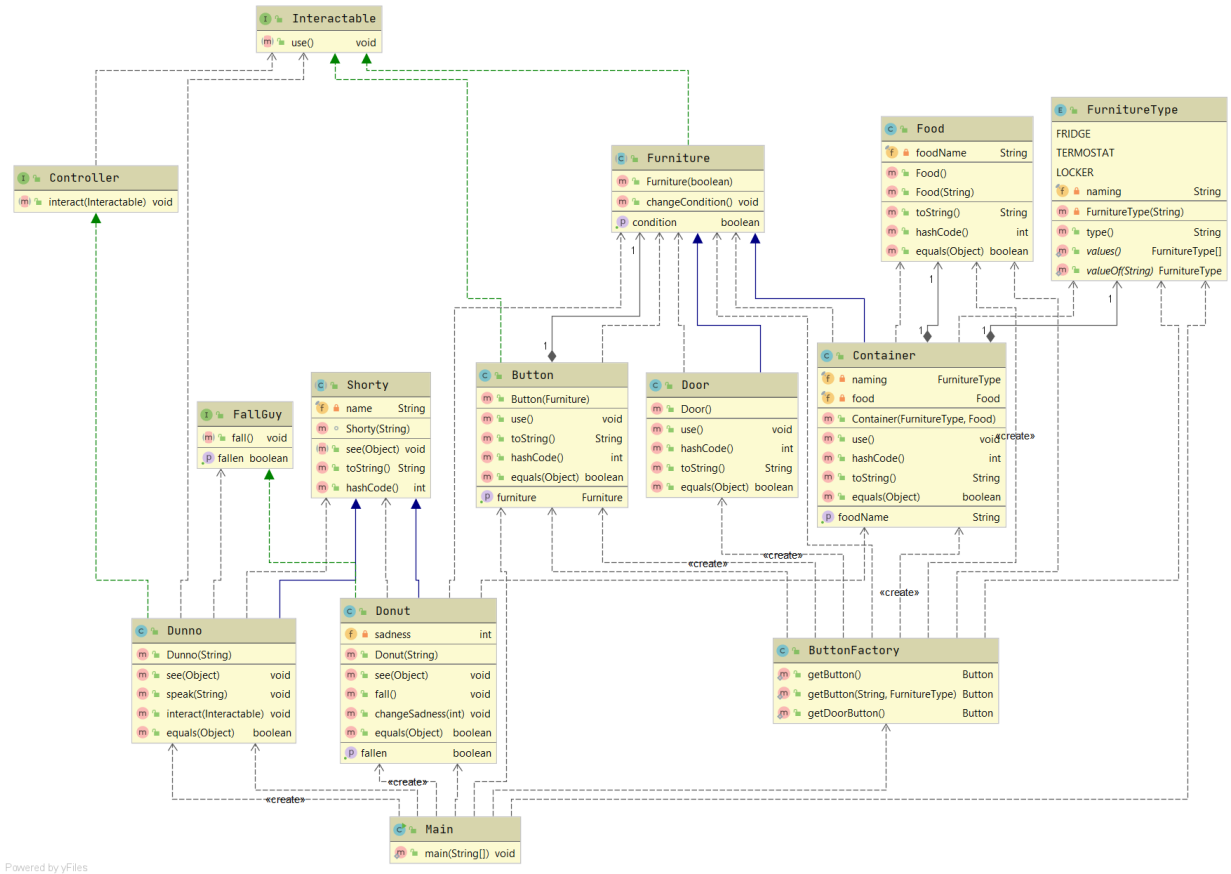
Программа должна удовлетворять следующим требованиям:

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){
        Dunno dunno = new Dunno("Незнайка");
        Donut donut = new Donut("Пончик");
        String naming = "пищевой продукт";
        Button doorButton = ButtonFactory.getDoorButton();
        Button[] buttons = {
                ButtonFactory.getButton(naming, FurnitureType.FRIDGE),
                ButtonFactory.getButton(naming, FurnitureType.LOCKER),
                ButtonFactory.getButton(naming, FurnitureType.TERMOSTAT)
        };
        donut.fall();
        dunno.see(donut);
        dunno.interact(doorButton);
        dunno.speak("");
        for (Button button : buttons){
            dunno.interact(button);
            donut.see(button.getFurniture());
        }
        donut.changeSadness(-3);
        donut.see(buttons[2].getFurniture());
    }
}
```

Shorty.java:

```
public abstract class Shorty{
    private final 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);
    }
}
```

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 final 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.toString().equals(this.toString()));
    }
}
```

FurnitureType.java:

```
public enum FurnitureType {
    FRIDGE("холодильник"),
    TERMOSTAT("термостат"),
    LOCKER("шкаф");
    private final String naming;
    FurnitureType(String naming){this.naming = naming;}
    public String type(){
        return this.naming;
    }
}
```

Furniture.java:

```
public abstract class Furniture implements Interactable{
    private boolean openess;
    public Furniture(boolean condition){
        openess = condition;
    }
    public void changeCondition() {
        openess = !openess;
    }
    public boolean getCondition(){
        return openess;
    }
}
```

Door.java:

```
public class Door extends Furniture{
    public Door(){
        super(true);
    }
    @Override
    public void use(){
        this.changeCondition();
        System.out.println(this.toString() + " " + (this.getCondition() ? "открыта" :
"закрыта"));
    }
    @Override
    public int hashCode(){
        return (this.getCondition() ? 1 : 0);
    }
```

```
    @Override
    public String toString() {
        return "Дверь";
    }
    @Override
    public boolean equals(Object object) {
        if (object instanceof Door)
            return (((Door)object).getCondition() == this.getCondition());
        return false;
    }
}
```

Container.java:

```
public class Container extends Furniture{
    private final FurnitureType naming;
    private final Food food;
    public Container(FurnitureType naming, Food food){
        super(false);
        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() ? "открыт":
"закрыт"));
    }
    @Override
    public int hashCode(){
        int i = 0;
        switch(naming){
            case LOCKER:
                i += 1;
                break;
            case TERMOSTAT:
                i += 2;
                break;
            case FRIDGE:
                i += 3;
                break;
            default:
                i += 0;
        }
        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){
        if (obj instanceof Container)
            return (((Container)obj).getFoodName().equals(this.getFoodName())) &&
(((Container)obj).getCondition() == this.getCondition());
        return false;
    }
}
```

Button.java:

```
public class Button implements Interactable{
    private final Furniture furniture;
    public Button(Furniture furniture){
        this.furniture = furniture;
    }
    public void use(){
        System.out.print("Кнопку нажали: ");
        this.furniture.use();
    }
    public Furniture getFurniture() {
        return furniture;
    }
    @Override
    public String toString(){
        return "Кнопка от " + furniture.toString();
    }
    @Override
    public int hashCode(){
        return (furniture.hashCode() * 2);
    }
    @Override
    public boolean equals(Object obj){
        if (obj instanceof Button)
            return (((Button)obj).getFurniture().equals(this.getFurniture()));
        return false;
    }
}
```

ButtonFactory.java:

```
public class ButtonFactory {
    public static Button getButton(){
        return ButtonFactory.getButton("пищевой продукт", FurnitureType.FRIDGE);
    }
    public static Button getButton(String foodName, FurnitureType 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 object) {
        System.out.print(this.toString() + " видит: " + object.toString());
        if (object instanceof FallGuy)
            System.out.println(((FallGuy) object).isFallen() ? " упал" : " стоит");
    }
    public void speak(String message){
        System.out.println(this.toString() + " говорит с широкой улыбкой: " + message);
    }
    @Override
    public void interact(Interactable object){
        System.out.println(this.toString() + " использует " + object.toString());
        object.use();
```

```
    }
    @Override
    public boolean equals(Object object){
        if (object instanceof Dunno)
            return (object.toString().equals(this.toString()));
        return false;
    }
}
```

Donut.java:

```
public class Donut extends Shorty implements FallGuy{
    private boolean condition = false;
    private int sadness;
    public Donut(String name){
        super(name);
        sadness = 3;
    }
    @Override
    public void see(Object object){
        System.out.print(this.toString() + " видит ");
        if (object instanceof Container)
            if (((Furniture)object).getCondition())
                System.out.println(((Container)object).getFoodName() + ", " +
(this.sadness > 0 ? "но даже это его не " : "и это его ") + "радует");
        else
            System.out.println(object.toString());
    }
    @Override
    public void fall(){
        condition = true;
        System.out.println(this.toString() + " падает");
    }
    public void changeSadness(int change){
        sadness += change;
    }
    @Override
    public boolean isFallen() {
        return condition;
    }
    @Override
    public boolean equals(Object object){
        if (object instanceof Donut)
            return (object.toString().equals(this.toString())) &&
(((Donut)object).isFallen() == this.isFallen());
        return false;
    }
}
```

Пример работы программы

Пончик падает

Незнайка видит: Пончик упал

Незнайка использует Кнопка от Дверь

Кнопку нажали: Дверь закрыта

Незнайка говорит с широкой улыбкой:

Незнайка использует Кнопка от холодильник с пищевой продукт

Кнопку нажали: холодильник открыт

Пончик видит пищевой продукт, но даже это его не радует

Незнайка использует Кнопка от шкаф с пищевой продукт

Кнопку нажали: шкаф открыт

Пончик видит пищевой продукт, но даже это его не радует

Незнайка использует Кнопка от термостат с пищевой продукт

Кнопку нажали: термостат 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.