«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет

По лабораторной работе №3

Вариант 290005

Студент:

Ильин Н. С.

P3110 поток 2.9

Преподаватель:

Наумова Н. А.

Санкт-Петербург, 2023 г.

Оглавление

Задание: 3

Выполнение работы: 3

Выводы: 13

Задание:

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

По правую руку от Короля находился Белый Кролик, с трубой в одной лапке и пергаментным свитком в другой. А в самом центре судебного зала стоял стол, а на нем красовалось большое блюдо с пирожками, н вид у них был такой аппетитный, что у Алисы прямо слюнки потекли. "Хорошо бы, суд уже кончился и позвали к столу!" - подумала она.

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.

Отчёт по работе должен содержать:

1. Текст задания.
2. Диаграмма классов объектной модели.
3. Исходный код программы.
4. Результат работы программы.
5. Выводы по работе.

Выполнение работы:

Исходный код:

src/Main.java

```
 1 | import characters.Alice;
 2 | import characters.King;
 3 | import characters.WhiteRabbit;
 4 | import characters.movement.Direction;
 5 | import food.Pie;
 6 | import props.*;
 7 | import furniture.Table;
 8 | import rooms.Courtroom;
 9 |
10 |
11 | public class Main {
12 |     public static void main(String[] args) {
13 |         System.out.println(new String("123") == new
String("123"));
```

```
14          System.out.println(new String("123") == "123");
15          System.out.println(("12"+"3").intern() == "123");
16          System.out.println(("12" + "3") == "123");
17          Alice alice = new Alice("Алиса");
18          King king = new King("Король");
19          WhiteRabbit whiteRabbit = new WhiteRabbit("Белый
Кролик");
20          Courtroom courtroom = new Courtroom("Судебный зал");
21          Table table = new Table("Стол");
22          Dish dish = new Dish("Большое блюдо с пирожками",
10);
23          dish.addFood(new Pie("Аппетитный пирожок",
"Курица"));
24          dish.addFood(new Pie("Аппетитный пирожок", "Рыба"));
25          dish.addFood(new Pie("Аппетитный пирожок",
"Картошка"));
26
27
28          whiteRabbit.standBy(king, Direction.BY_RIGHT_HAND);
29          whiteRabbit.hold(new Trumpet("Труба"));
30          whiteRabbit.hold(new Scroll("Пергаментный свиток"));
31          courtroom.setCenterObject(table);
32          table.setTopObject(dish);
33          dish.lookLike("аппетитно");
34          alice.salivateAt(dish);
35          alice.think("Хорошо бы, суд уже кончился и позвали к
столу!");
36          //alice.walkTo(Direction.CENTER);
37      }
38  }
```

src/rooms/Courtroom.java

```
1   package rooms;
2
3   import furniture.*;
4
5   public class Courtroom extends Room {
6       public Courtroom(String name) {
7           super(name);
8       }
9
10      public void setCenterObject(Table object){
11          System.out.println("В центре " + getName() + " стоит
" + object.getName());
12      }
13
14
15  }
```

src/rooms/Room.java

```
1   package rooms;
2
3   import interfaces.Nameable;
4
```

```java
public abstract class Room implements Nameable {
    private String name;
    public Room(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass())
return false;
        Room room = (Room) obj;
        return name.equals(room.name);
    }

    @Override
    public int hashCode() {
        return name.hashCode();
    }
    @Override
    public String toString() {
        return "Room{" +
                "name='" + name + '\'' +
                '}';
    }
}

```

src/props/Dish.java

```java
package props;


import food.Food;

import java.util.Arrays;

public class Dish extends Prop {
    private Food[] foodItems;
    private int itemCount;

    public Dish(String name, int maxItemCount) {
        super(name);
        this.foodItems = new Food[maxItemCount];
        this.itemCount = 0;
    }

    public void addFood(Food foodItem) {
        if (itemCount < foodItems.length) {
            foodItems[itemCount] = foodItem;
            itemCount++;
        } else {
```

```
            System.out.println("На блюде больше нет места.");
        }
    }

    public Food[] getFoodItems() {
        return foodItems;
    }
}
```

src/props/Prop.java

```
package props;

import food.*;
import interfaces.Nameable;

public abstract class Prop implements Nameable {
    protected String name;

    public Prop(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }
    public void lookLike(String description) {
        System.out.println(getName() + " выглядит " +
description);
    }
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass())
return false;
        Prop prop = (Prop) obj;
        return name.equals(prop.name);
    }

    public int hashCode() {
        return name.hashCode();
    }

    public String toString() {
        return "Prop{" +
                "name='" + name + '\'' +
                '}';
    }
}

```

src/props/Scroll.java

```
package props;

public class Scroll extends Prop {
    private String text = "...";
```

```
    public Scroll(String name){
        super(name);
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}

```

src/props/Trumpet.java

```
package props;

public class Trumpet extends Prop {
    public Trumpet(String name){
        super(name);
    }

    public void makeSound(){
        System.out.println(getName() + " издаёт звук");
    }
}

```

src/interfaces/Nameable.java

```
package interfaces;

public interface Nameable {
    String getName();
}
```

src/interfaces/Thinkable.java

```
package interfaces;

public interface Thinkable {
    void think(String thought);
}
```

src/furniture/Furniture.java

```
package furniture;

import interfaces.Nameable;

public abstract class Furniture implements Nameable {
    protected String name;

    public Furniture(String name) {
        this.name = name;
    }
```

```
11 |
12 |     public String getName() {
13 |         return this.name;
14 |     }
15 |
16 |     public boolean equals(Object obj) {
17 |         if (this == obj) return true;
18 |         if (obj == null || getClass() != obj.getClass())
return false;
19 |         Furniture furniture = (Furniture) obj;
20 |         return name.equals(furniture.name);
21 |     }
22 |
23 |     public int hashCode() {
24 |
25 |         return name.hashCode();
26 |     }
27 |
28 |     public String toString() {
29 |         return "Furniture{" +
30 |                 "name='" + name + '\'' +
31 |                 '}';
32 |     }
33 | }
```

src/furniture/Table.java

```
1  | package furniture;
2  |
3  | import props.*;
4  | public class Table extends Furniture {
5  |     public Table(String name){
6  |         super(name);
7  |     }
8  |     private Prop topObject;
9  |     private Prop bottomObject;
10 |
11 |     public void setTopObject(Prop object){
12 |         this.topObject = object;
13 |         System.out.println("На " + getName() + " стоит " +
object.getName());
14 |     }
15 |     public void setBottomObject(Prop object){
16 |         this.bottomObject = object;
17 |         System.out.println("Под " + getName() + " стоит " +
object.getName());
18 |     }
19 | }
```

src/food/Food.java

```
1  | package food;
2  |
3  | import interfaces.Nameable;
4  | public abstract class Food implements Nameable {
5  |     protected String name;
```

```
    public Food(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
    public int hashCode() {
        return name.hashCode();
    }
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass())
return false;
        Food food = (Food) obj;
        return name.equals(food.name);
    }
    public String toString() {
        return "Food{" +
                "name='" + name + '\'' +
                '}';
    }
}

```

src/food/Pie.java

```
package food;

import interfaces.Nameable;

import javax.swing.*;

public class Pie extends Food {
    private String filling;
    public Pie(String name, String filling){
        super(name);
        this.filling = filling;
    }

    public String getFilling() {
        return this.filling;
    }
}
```

src/characters/Alice.java

```
package characters;

public class Alice extends Character {
    public Alice(String name){
        super(name);
    }
}
```

src/characters/Character.java

```
  1 | package characters;
  2 |
  3 | import characters.movement.Direction;
  4 | import interfaces.Nameable;
  5 | import interfaces.Thinkable;
  6 | import props.Prop;
  7 |
  8 | public abstract class Character implements Nameable,
Thinkable {
  9 |     protected String name;
 10 |     private Prop objectInHand;
 11 |
 12 |     public Character(String name) {
 13 |         this.name = name;
 14 |     }
 15 |
 16 |     public String getName() {
 17 |         return this.name;
 18 |     }
 19 |
 20 |     public void standBy(Character character, Direction
direction) {
 21 |         System.out.println(getName() + " находится " +
direction.getDirection() + " " + character.getName());
 22 |     }
 23 |
 24 |     public void salivateAt(Prop prop) {
 25 |         System.out.println("У " + getName() + " текут слюнки
от " + prop.getName());
 26 |     }
 27 |
 28 |     public void think(String thought) {
 29 |         System.out.println("\"" + thought + "\" - думает " +
getName());
 30 |     }
 31 |
 32 |     public void walkTo(Direction direction) {
 33 |         System.out.println(getName() + " идет " +
direction.getDirection());
 34 |     }
 35 |
 36 |     @Override
 37 |     public boolean equals(Object obj) {
 38 |         if (this == obj) return true;
 39 |         if (obj == null || getClass() != obj.getClass())
return false;
 40 |         Character character = (Character) obj;
 41 |         return name.equals(character.name);
 42 |     }
 43 |
 44 |     @Override
 45 |     public int hashCode() {
```

```
46          return name.hashCode();
47      }
48
49      @Override
50      public String toString() {
51          return "Character{" +
52                  "name='" + name + '\'' +
53                  '}';
54      }
55
56      public void hold(Prop object) {
57          this.objectInHand = object;
58          System.out.println(getName() + " держит " +
this.objectInHand.getName());
59      }
60  }
61
62
```

src/characters/King.java

```
1   package characters;
2
3   public class King extends Character {
4       public King(String name){
5           super(name);
6       }
7       public void command(String command){
8           System.out.println(getName() + " приказывает: " +
command);
9       }
10  }
```

src/characters/WhiteRabbit.java

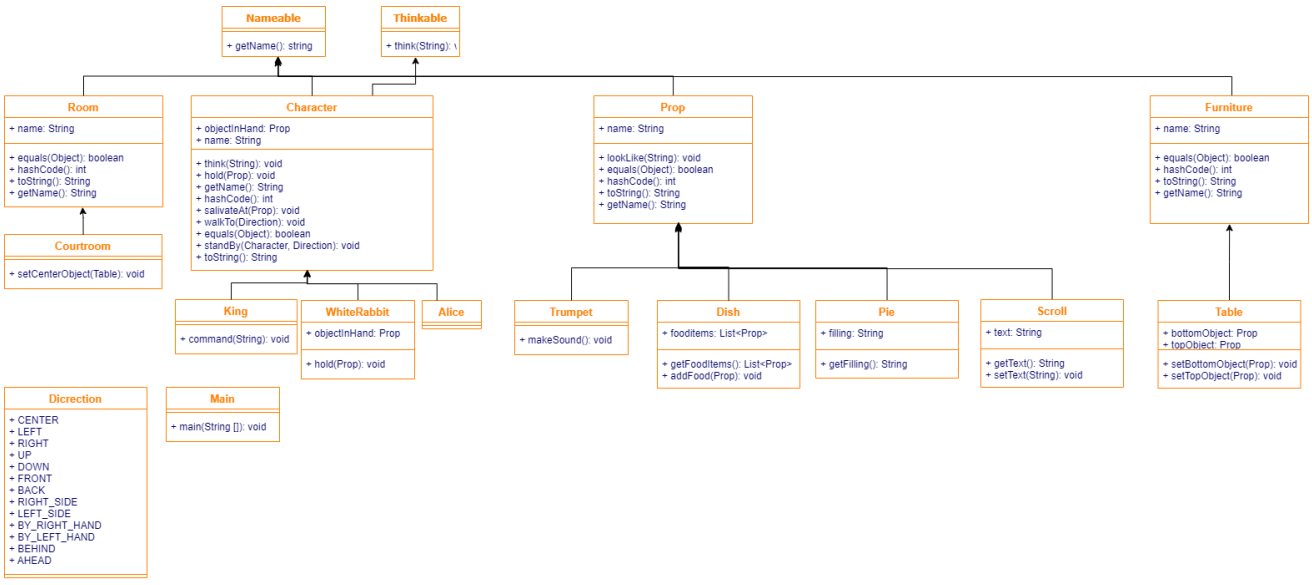
```
1   package characters;
2
3   import props.Prop;
4
5   public class WhiteRabbit extends Character {
6       private Prop objectInHand;
7       public WhiteRabbit(String name){
8           super(name);
9       }
10      @Override
11      public void hold(Prop object){
12          this.objectInHand = object;
13          System.out.println(getName() + " держит в лапке " +
this.objectInHand.getName());
14      }
15  }
```

src/characters/movement/Direction.java

```
1   package characters.movement;
2
3   public enum Direction {
```

```
4       CENTER("в центр"),
5       LEFT("налево"),
6       RIGHT("направо"),
7       UP("вверх"),
8       DOWN("вниз"),
9       FRONT("вперед"),
10      BACK("назад"),
11      RIGHT_SIDE("с правой стороны от"),
12      LEFT_SIDE("с левой стороны от"),
13      BY_RIGHT_HAND("по правую руку от"),
14      BY_LEFT_HAND("по левую руку от"),
15      BEHIND("сзади"),
16      AHEAD("спереди");
17
18      private String direction;
19
20      Direction(String direction) {
21          this.direction = direction;
22      }
23
24      public String getDirection() {
25          return direction;
26      }
27
28  }
```

Диаграмма классов:



Вывод программы:

```
Белый Кролик находится по правую руку от Король
Белый Кролик держит в лапке Труба
Белый Кролик держит в лапке Пергаментный свиток
В центре Судебный зал стоит Стол
На Стол стоит Большое блюдо с пирожками
Большое блюдо с пирожками выглядит аппетитно
У Алиса текут слюнки от Большое блюдо с пирожками
"Хорошо бы, суд уже кончился и позвали к столу!" - думает Алиса
```

Выводы:

В результате проделанной работы, я попрактиковался с применением принципов ООП, самостоятельно спроектировал объектную модель приложения, вручную изобразил диаграмму классов.