«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет

По лабораторной работе №4

Вариант 290005.4

Студент:

Ильин Н. С.

P3110 поток 2.9

Преподаватель:

Наумова Н. А.

Санкт-Петербург, 2023 г.

Оглавление

Задание:

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

Когда Алиса с Грифоном прибежали. Король и Королева уже сидели на троне, а кругом собралась огромная толпа: пичужки, зверюшки всех пород, не говоря уже о картах всех мастей. Перед судейским троном стоял в цепях под охраной двух солдат - один справа, другой - слева - Червонный Валет.

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.

Но так как, судя по всему, до этого было еще далеко, она, чтобы скоротать время, стала рассматривать все окружающее. Хотя Алиса раньше никогда не бывала в суде, она устала про суд в книжках, и ей было очень приятно отметить, что она знает, как тут все - или почти все - называется.

"Вот это судья,- сказала она про себя.- Кто в большом парике, тот и судья".

Программа должна удовлетворять следующим требованиям:

1. В программе должны быть реализованы 2 собственных класса исключений (checked и unchecked), а также обработка исключений этих классов.
2. В программу необходимо добавить использование локальных, анонимных и вложенных классов (static и non-static).

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.

Вопросы к защите лабораторной работы:

1. Обработка исключительных ситуаций, три типа исключений.
2. Вложенные, локальные и анонимные классы.
3. Механизм рефлексии (reflection) в Java. Класс `Class`.

Выполнение работы:

Исходный код:

src/Main.java

```
  1 | import characters.*;
  2 | import characters.Character;
  3 | import characters.groups.*;
  4 | import characters.movement.*;
  5 | import exceptions.*;
  6 | import food.*;
  7 | import furniture.*;
  8 | import props.*;
  9 | import rooms.*;
 10 |
 11 |
 12 | public class Main {
 13 |     public static void main(String[] args) throws Exception {
 14 |         Queen queen = new Queen("Королева");
 15 |         King king = new King("Король");
 16 |         Alice alice = new Alice("Алиса");
 17 |         Gryphon gryphon = new Gryphon("Грифон");
 18 |         KnaveOfHearts knaveOfHearts = new KnaveOfHearts("Червонный
Валет");
 19 |         Furniture kingsThrone = new Throne("Трон");
 20 |         Furniture queensThrone = new Throne("Трон");
 21 |         king.sitOn(kingsThrone);
 22 |         queen.sitOn(queensThrone);
 23 |
 24 |         int retryCount = 3; // Количество попыток для прибытия
 25 |         // Checked exception
 26 |         for (Character character : new Character[] {alice,
gryphon}) {
 27 |             for (int i = 0; i < retryCount; i++) {
 28 |                 try {
 29 |                     character.arrive();
 30 |                     break;
 31 |                 } catch (ArrivalException e) {
 32 |                     // System.err.println(e.getMessage());
 33 |                     if (i < retryCount - 1) {
 34 |                         System.out.println(character.getName() + "
ещё раз пытается найти путь");
 35 |                     } else {
 36 |                         System.out.println(character.getName() + "
так и не смог прибежать");
 37 |                         System.exit(1);
 38 |
 39 |                     }
 40 |                 }
 41 |             }
 42 |         }
```

```
  43 |
  44 |
  45 |
  46 |           // Толпа
  47 |           Crowd crowd = new Crowd();
  48 |           // Unchecked exception
  49 |           try {
  50 |               // Добавление участников толпы
  51 |               crowd.addMember(new Crowd.Member("Пичужка 1"));
  52 |               crowd.addMember(new Crowd.Member("Пичужка 2"));
  53 |               crowd.addMember(new Crowd.Member("Зверушка 1"));
  54 |               crowd.addMember(new Crowd.Member("Зверушка 2"));
  55 |               crowd.addMember(new Crowd.Member("Карта Валет"));
  56 |               crowd.addMember(new Crowd.Member("Карта Туз"));
  57 |               // Сбор толпы
  58 |               crowd.gather();
  59 |           }
  60 |           catch (CantAttendException e) {
  61 |               System.err.println(e.getMessage());
  62 |               System.err.println("В толпу был добавлен запрещённый
участник. Выполнение программы завершено.");
  63 |               System.exit(1);
  64 |           }
  65 |           // Охрана
  66 |           KnaveOfHearts.Soldier guardLeft = knaveOfHearts.new
Soldier("Левый солдат");
  67 |           KnaveOfHearts.Soldier guardRight = knaveOfHearts.new
Soldier("Правый солдат");
  68 |
  69 |           knaveOfHearts.assignGuardLeft(guardLeft);
  70 |           knaveOfHearts.assignGuardRight(guardRight);
  71 |
  72 |           //=============================================================
================
  73 |           WhiteRabbit whiteRabbit = new WhiteRabbit("Белый Кролик");
  74 |           Courtroom courtroom = new Courtroom("Судебный зал");
  75 |           Table table = new Table("Стол");
  76 |           Dish dish = new Dish("Большое блюдо с пирожками", 10);
  77 |           dish.addFood(new Pie("Аппетитный пирожок", "Курица"));
  78 |           dish.addFood(new Pie("Аппетитный пирожок", "Рыба"));
  79 |           dish.addFood(new Pie("Аппетитный пирожок", "Картошка"));
  80 |
  81 |           whiteRabbit.standBy(king, Direction.BY_RIGHT_HAND);
  82 |           whiteRabbit.hold(new Trumpet("Труба"));
  83 |           whiteRabbit.hold(new Scroll("Пергаментный свиток"));
  84 |           courtroom.setCenterObject(table);
  85 |           table.setTopObject(dish);
  86 |           dish.lookLike("аппетитно");
  87 |           alice.salivateAt(dish);
  88 |           alice.think("Хорошо бы, суд уже кончился и позвали к
столу!");
  89 |           //=============================================================
================
  90 |
  91 |           // Алиса
  92 |           alice.observeScene(new String[]{"судья", "трон",
"зверюшки", "пичужки", "карты"});
  93 |           alice.think("Вот это судья. Кто в большом парике, тот и
судья");
```

```
        // Анонимный класс рассказчика
        Character narrator = new Character("Рассказчик") {
            @Override
            public void say(String speach) {
                System.out.println(speach);
            }
        };
        narrator.say("Хотя Алиса раньше никогда не бывала в суде, она устала про суд в книжках, " +
                "и ей было очень приятно отметить, что она знает, как тут все - или почти все - называется.");

    }

}
```

src/rooms/Courtroom.java

```
package rooms;

import furniture.*;

public class Courtroom extends Room {
    public Courtroom(String name) {
        super(name);
    }

    public void setCenterObject(Table object){
        System.out.println("В центре " + getName() + " стоит " + object.getName());
    }


}
```

src/rooms/Room.java

```
package rooms;

import interfaces.Nameable;

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

```
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

```java
package interfaces;

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

src/interfaces/Thinkable.java

```java
package interfaces;

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

src/furniture/Furniture.java

```java
package furniture;

import interfaces.Nameable;

public abstract class Furniture implements Nameable {
    protected String name;

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

    public String getName() {
        return this.name;
    }

    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return
false;
        Furniture furniture = (Furniture) obj;
        return name.equals(furniture.name);
    }

    public int hashCode() {

        return name.hashCode();
    }

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

src/furniture/Table.java

```java
package furniture;

import props.*;
public class Table extends Furniture {
    public Table(String name){
        super(name);
```

```
    }
    private Prop topObject;
    private Prop bottomObject;

    public void setTopObject(Prop object){
        this.topObject = object;
        System.out.println("На " + getName() + " стоит " +
object.getName());
    }
    public void setBottomObject(Prop object){
        this.bottomObject = object;
        System.out.println("Под " + getName() + " стоит " +
object.getName());
    }
}
```

src/furniture/Throne.java

```
package furniture;

public class Throne extends Furniture {
    public Throne(String name) {
        super(name);
    }
}
```

src/food/Food.java

```
package food;

import interfaces.Nameable;
public abstract class Food implements Nameable {
    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;

```

```java
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/exceptions/ArrivalException.java

```java
package exceptions;

public class ArrivalException extends Exception {
    public ArrivalException(String message) {
        super(message);
    }
}

```

src/exceptions/CantAttendException.java

```java
package exceptions;

public class CantAttendException extends RuntimeException{
    public CantAttendException(String message) {
        super(message);
    }
}

```

src/characters/Alice.java

```java
package characters;

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

src/characters/Character.java

```java
package characters;

import characters.movement.Direction;
import furniture.Furniture;
import interfaces.Nameable;
import interfaces.Thinkable;
import props.Prop;
import exceptions.ArrivalException;

public abstract class Character implements Nameable, Thinkable {
    protected String name;
    private Prop objectInHand;
```

```
  13 |
  14 |       public Character(String name) {
  15 |           this.name = name;
  16 |       }
  17 |
  18 |       public String getName() {
  19 |           return this.name;
  20 |       }
  21 |
  22 |       public void standBy(Character character, Direction direction) {
  23 |           System.out.println(getName() + " находится " +
direction.getDirection() + " " + character.getName());
  24 |       }
  25 |
  26 |       // checked exception
  27 |       public void arrive() throws ArrivalException {
  28 |           if (Math.random() < 0.3) {
  29 |               throw new ArrivalException(getName() + " заблудился
пока бежал");
  30 |           }
  31 |           System.out.println(getName() + " прибежал");
  32 |       }
  33 |
  34 |       public void salivateAt(Prop prop) {
  35 |           System.out.println("У " + getName() + " текут слюнки от " +
prop.getName());
  36 |       }
  37 |
  38 |       public void think(String thought) {
  39 |           System.out.println("\"" + thought + "\" - думает " +
getName());
  40 |       }
  41 |
  42 |       public void say(String speach) {
  43 |           System.out.println(getName() + "говорит: " + speach);
  44 |       }
  45 |
  46 |       public void sitOn(Furniture furniture) {
  47 |           System.out.println(getName() + " сидит на " +
furniture.getName());
  48 |       }
  49 |
  50 |       public void observeScene(String[] sceneElements) {
  51 |           // Локальный класс для элемента сцены
  52 |           class SceneElement {
  53 |               private String name;
  54 |
  55 |               SceneElement(String name) {
  56 |                   this.name = name;
  57 |               }
  58 |
  59 |               public void describe() {
  60 |                   System.out.println(Character.this.name + "
рассматривает " + this.name);
  61 |               }
  62 |           }
  63 |
  64 |           for (String element : sceneElements) {
  65 |               // Использование локального класса
```

```
66 |                 new SceneElement(element).describe();
67 |             }
68 |
69 |         }
70 |
71 |         public void walkTo(Direction direction) {
72 |             System.out.println(getName() + " идет " +
direction.getDirection());
73 |         }
74 |
75 |         @Override
76 |         public boolean equals(Object obj) {
77 |             if (this == obj) return true;
78 |             if (obj == null || getClass() != obj.getClass()) return
false;
79 |             Character character = (Character) obj;
80 |             return name.equals(character.name);
81 |         }
82 |
83 |         @Override
84 |         public int hashCode() {
85 |             return name.hashCode();
86 |         }
87 |
88 |         @Override
89 |         public String toString() {
90 |             return "Character{" +
91 |                     "name='" + name + '\'' +
92 |                     '}';
93 |         }
94 |
95 |         public void hold(Prop object) {
96 |             this.objectInHand = object;
97 |             System.out.println(getName() + " держит " +
this.objectInHand.getName());
98 |         }
99 |  }
100 |
101 |
```

src/characters/Gryphon.java

```
1 | package characters;
2 |
3 | public class Gryphon extends Character {
4 |     public Gryphon(String name){
5 |         super(name);
6 |     }
7 | }
8 |
9 |
```

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){
```

```
        System.out.println(getName() + " приказывает: " + command);
    }
}
```

src/characters/KnaveOfHearts.java

```
package characters;

import characters.movement.Direction;

public class KnaveOfHearts extends Character {
    private Soldier guardLeft;
    private Soldier guardRight;

    public KnaveOfHearts(String name) {
        super(name);
    }

    // Вложенный нестатический класс Солдата
    public class Soldier extends Character {
        public Soldier(String name) {
            super(name);
        }

        public void guard(KnaveOfHearts knave) {
            System.out.println(getName() + " охраняет " +
knave.getName());
        }
    }

    // Методы для создания солдат-охранников
    public void assignGuardLeft(Soldier guard) {
        this.guardLeft = guard;
        guard.guard(this);
        guard.standBy(this, Direction.BY_LEFT_HAND);
    }

    public void assignGuardRight(Soldier guard) {
        this.guardRight = guard;
        guard.guard(this);
        guard.standBy(this, Direction.BY_RIGHT_HAND);
    }
}

```

src/characters/Queen.java

```
package characters;

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


```

src/characters/WhiteRabbit.java

```
package characters;

```

```
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 }
```

src/characters/groups/Crowd.java

```
1  package characters.groups;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import exceptions.CantAttendException;
6
7  public class Crowd {
8      private List<Member> members = new ArrayList<>();
9
10     // Статический вложенный класс Member
11     public static class Member {
12         private String name;
13         public Member(String name) {
```

```
14 |             this.name = name;
15 |             if (name == "Карта Король") {
16 |                 throw new CantAttendException("Короли не
присоединяются к толпе");
17 |             }
18 |         }
19 |
20 |         public void attend() {
21 |             System.out.println(this.name + " собирается в толпу");
22 |         }
23 |     }
24 |
25 |     // Метод для добавления нового участника в толпу
26 |     public void addMember(Member member) {
27 |         members.add(member);
28 |     }
29 |
30 |     // Метод для иллюстрации присутствия толпы
31 |     public void gather() {
32 |         for (Member member : members) {
33 |             member.attend();
34 |         }
35 |     }
36 | }
37 |
```

src/Main.java

```
 1 | import characters.*;
 2 | import characters.Character;
 3 | import characters.groups.*;
 4 | import characters.movement.*;
 5 | import exceptions.*;
 6 | import food.*;
 7 | import furniture.*;
 8 | import props.*;
 9 | import rooms.*;
10 |
11 |
12 | public class Main {
13 |     public static void main(String[] args) throws Exception {
14 |         Queen queen = new Queen("Королева");
15 |         King king = new King("Король");
16 |         Alice alice = new Alice("Алиса");
17 |         Gryphon gryphon = new Gryphon("Грифон");
18 |         KnaveOfHearts knaveOfHearts = new KnaveOfHearts("Червонный
Валет");
19 |         Furniture kingsThrone = new Throne("Трон");
20 |         Furniture queensThrone = new Throne("Трон");
21 |         king.sitOn(kingsThrone);
22 |         queen.sitOn(queensThrone);
23 |
24 |         int retryCount = 3; // Количество попыток для прибытия
25 |         // Checked exception
26 |         for (Character character : new Character[] {alice,
gryphon}) {
27 |             for (int i = 0; i < retryCount; i++) {
28 |                 try {
```

```
                    character.arrive();
                    break;
                } catch (ArrivalException e) {
                    // System.err.println(e.getMessage());
                    if (i < retryCount - 1) {
                        System.out.println(character.getName() + "
ещё раз пытается найти путь");
                    } else {
                        System.out.println(character.getName() + "
так и не смог прибежать");
                        System.exit(1);

                    }
                }
            }



            // Толпа
            Crowd crowd = new Crowd();
            // Unchecked exception
            try {
                // Добавление участников толпы
                crowd.addMember(new Crowd.Member("Пичужка 1"));
                crowd.addMember(new Crowd.Member("Пичужка 2"));
                crowd.addMember(new Crowd.Member("Зверушка 1"));
                crowd.addMember(new Crowd.Member("Зверушка 2"));
                crowd.addMember(new Crowd.Member("Карта Валет"));
                crowd.addMember(new Crowd.Member("Карта Туз"));
                // Сбор толпы
                crowd.gather();
            }
            catch (CantAttendException e) {
                System.err.println(e.getMessage());
                System.err.println("В толпу был добавлен запрещённый
участник. Выполнение программы завершено.");
                System.exit(1);
            }
            // Охрана
            KnaveOfHearts.Soldier guardLeft = knaveOfHearts.new
Soldier("Левый солдат");
            KnaveOfHearts.Soldier guardRight = knaveOfHearts.new
Soldier("Правый солдат");

            knaveOfHearts.assignGuardLeft(guardLeft);
            knaveOfHearts.assignGuardRight(guardRight);

            //=============================================================
================
            WhiteRabbit whiteRabbit = new WhiteRabbit("Белый Кролик");
            Courtroom courtroom = new Courtroom("Судебный зал");
            Table table = new Table("Стол");
            Dish dish = new Dish("Большое блюдо с пирожками", 10);
            dish.addFood(new Pie("Аппетитный пирожок", "Курица"));
            dish.addFood(new Pie("Аппетитный пирожок", "Рыба"));
            dish.addFood(new Pie("Аппетитный пирожок", "Картошка"));

            whiteRabbit.standBy(king, Direction.BY_RIGHT_HAND);
```

```
82          whiteRabbit.hold(new Trumpet("Труба"));
83          whiteRabbit.hold(new Scroll("Пергаментный свиток"));
84          courtroom.setCenterObject(table);
85          table.setTopObject(dish);
86          dish.lookLike("аппетитно");
87          alice.salivateAt(dish);
88          alice.think("Хорошо бы, суд уже кончился и позвали к
столу!");
89          //=============================================================
================
90
91          // Алиса
92          alice.observeScene(new String[]{"судья", "трон",
"зверюшки", "пичужки", "карты"});
93          alice.think("Вот это судья. Кто в большом парике, тот и
судья");
94          // Анонимный класс рассказчика
95          Character narrator = new Character("Рассказчик") {
96              @Override
97              public void say(String speach) {
98                  System.out.println(speach);
99              }
100         };
101         narrator.say("Хотя Алиса раньше никогда не бывала в суде,
она устала про суд в книжках, " +
102                 "и ей было очень приятно отметить, что она знает,
как тут все - или почти все - называется.");
103
104     }
105
106 }
```

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
5  public abstract class Room implements Nameable {
6      private String name;
7      public Room(String name){
8          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

```
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

```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

```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

```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

```java
package interfaces;

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

src/interfaces/Thinkable.java

```java
package interfaces;

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

src/furniture/Furniture.java

```java
package furniture;

import interfaces.Nameable;

public abstract class Furniture implements Nameable {
    protected String name;

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

    public String getName() {
        return this.name;
    }

    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return
false;
        Furniture furniture = (Furniture) obj;
        return name.equals(furniture.name);
    }

    public int hashCode() {

        return name.hashCode();
    }

```

```
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/furniture/Throne.java

```
1  package furniture;
2
3  public class Throne extends Furniture {
4      public Throne(String name) {
5          super(name);
6      }
7  }
```

src/food/Food.java

```
 1  package food;
 2
 3  import interfaces.Nameable;
 4  public abstract class Food implements Nameable {
 5      protected String name;
 6      public Food(String name) {
 7          this.name = name;
 8      }
 9      public String getName() {
10          return this.name;
11      }
12      public int hashCode() {
13          return name.hashCode();
14      }
15      public boolean equals(Object obj) {
16          if (this == obj) return true;
17          if (obj == null || getClass() != obj.getClass()) return
false;
```

```
18          Food food = (Food) obj;
19          return name.equals(food.name);
20      }
21      public String toString() {
22          return "Food{" +
23                  "name='" + name + '\'' +
24                  '}';
25      }
26  }
27
```

src/food/Pie.java

```
 1  package food;
 2
 3  import interfaces.Nameable;
 4
 5  import javax.swing.*;
 6
 7  public class Pie extends Food {
 8      private String filling;
 9      public Pie(String name, String filling){
10          super(name);
11          this.filling = filling;
12      }
13
14      public String getFilling() {
15          return this.filling;
16      }
17  }
```

src/exceptions/ArrivalException.java

```
1  package exceptions;
2
3  public class ArrivalException extends Exception {
4      public ArrivalException(String message) {
5          super(message);
6      }
7  }
8
```

src/exceptions/CantAttendException.java

```
1  package exceptions;
2
3  public class CantAttendException extends RuntimeException{
4      public CantAttendException(String message) {
5          super(message);
6      }
7  }
8
```

src/characters/Alice.java

```
1  package characters;
2
3  public class Alice extends Character {
4      public Alice(String name){
5          super(name);
6      }
7  }
```

src/characters/Character.java

```java
package characters;

import characters.movement.Direction;
import furniture.Furniture;
import interfaces.Nameable;
import interfaces.Thinkable;
import props.Prop;
import exceptions.ArrivalException;

public abstract class Character implements Nameable, Thinkable {
    protected String name;
    private Prop objectInHand;

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

    public String getName() {
        return this.name;
    }

    public void standBy(Character character, Direction direction) {
        System.out.println(getName() + " находится " +
direction.getDirection() + " " + character.getName());
    }

    // checked exception
    public void arrive() throws ArrivalException {
        if (Math.random() < 0.3) {
            throw new ArrivalException(getName() + " заблудился
пока бежал");
        }
        System.out.println(getName() + " прибежал");
    }

    public void salivateAt(Prop prop) {
        System.out.println("У " + getName() + " текут слюнки от " +
prop.getName());
    }

    public void think(String thought) {
        System.out.println("\"" + thought + "\" - думает " +
getName());
    }

    public void say(String speach) {
        System.out.println(getName() + "говорит: " + speach);
    }

    public void sitOn(Furniture furniture) {
        System.out.println(getName() + " сидит на " +
furniture.getName());
    }

    public void observeScene(String[] sceneElements) {
        // Локальный класс для элемента сцены
        class SceneElement {
```

```
53 |             private String name;
54 |
55 |             SceneElement(String name) {
56 |                 this.name = name;
57 |             }
58 |
59 |             public void describe() {
60 |                 System.out.println(Character.this.name + "
рассматривает " + this.name);
61 |             }
62 |         }
63 |
64 |         for (String element : sceneElements) {
65 |             // Использование локального класса
66 |             new SceneElement(element).describe();
67 |         }
68 |
69 |     }
70 |
71 |     public void walkTo(Direction direction) {
72 |         System.out.println(getName() + " идет " +
direction.getDirection());
73 |     }
74 |
75 |     @Override
76 |     public boolean equals(Object obj) {
77 |         if (this == obj) return true;
78 |         if (obj == null || getClass() != obj.getClass()) return
false;
79 |         Character character = (Character) obj;
80 |         return name.equals(character.name);
81 |     }
82 |
83 |     @Override
84 |     public int hashCode() {
85 |         return name.hashCode();
86 |     }
87 |
88 |     @Override
89 |     public String toString() {
90 |         return "Character{" +
91 |                 "name='" + name + '\'' +
92 |                 '}';
93 |     }
94 |
95 |     public void hold(Prop object) {
96 |         this.objectInHand = object;
97 |         System.out.println(getName() + " держит " +
this.objectInHand.getName());
98 |     }
99 | }
100 |
101 |
```

src/characters/Gryphon.java

```
1 | package characters;
2 |
3 | public class Gryphon extends Character {
4 |     public Gryphon(String name){
```

```
        super(name);
    }
}

```

src/characters/King.java

```
package characters;

public class King extends Character {
    public King(String name){
        super(name);
    }
    public void command(String command){
        System.out.println(getName() + " приказывает: " + command);
    }
}
```

src/characters/KnaveOfHearts.java

```
package characters;

import characters.movement.Direction;

public class KnaveOfHearts extends Character {
    private Soldier guardLeft;
    private Soldier guardRight;

    public KnaveOfHearts(String name) {
        super(name);
    }

    // Вложенный нестатический класс Солдата
    public class Soldier extends Character {
        public Soldier(String name) {
            super(name);
        }

        public void guard(KnaveOfHearts knave) {
            System.out.println(getName() + " охраняет " +
knave.getName());
        }
    }

    // Методы для создания солдат-охранников
    public void assignGuardLeft(Soldier guard) {
        this.guardLeft = guard;
        guard.guard(this);
        guard.standBy(this, Direction.BY_LEFT_HAND);
    }

    public void assignGuardRight(Soldier guard) {
        this.guardRight = guard;
        guard.guard(this);
        guard.standBy(this, Direction.BY_RIGHT_HAND);
    }
}

```

src/characters/Queen.java

```java
package characters;

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


```

src/characters/WhiteRabbit.java

```java
package characters;

import props.Prop;

public class WhiteRabbit extends Character {
    private Prop objectInHand;
    public WhiteRabbit(String name){
        super(name);
    }
    @Override
    public void hold(Prop object){
        this.objectInHand = object;
        System.out.println(getName() + " держит в лапке " +
this.objectInHand.getName());
    }
}
```

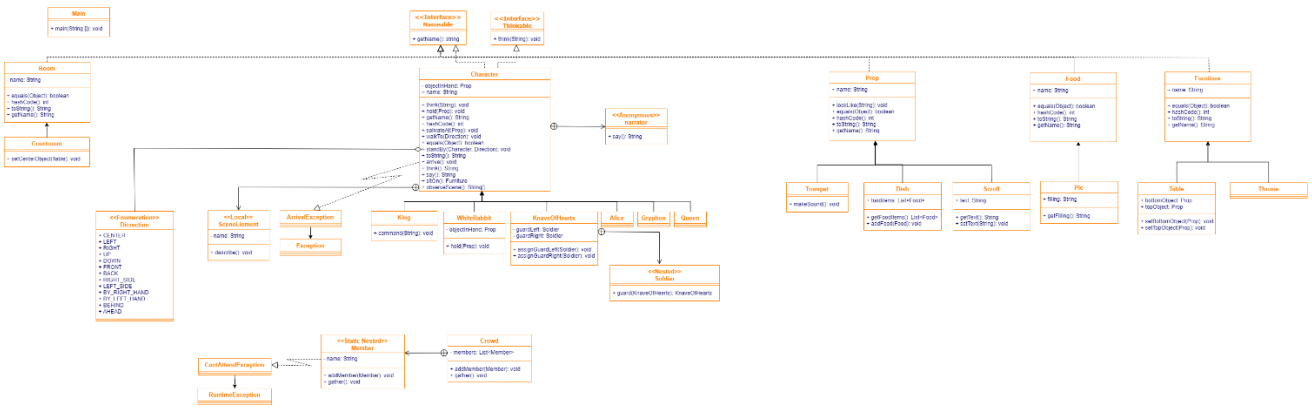
src/characters/movement/Direction.java

```java
package characters.movement;

public enum Direction {
    CENTER("в центр"),
    LEFT("налево"),
    RIGHT("направо"),
    UP("вверх"),
    DOWN("вниз"),
    FRONT("вперед"),
    BACK("назад"),
    RIGHT_SIDE("с правой стороны от"),
    LEFT_SIDE("с левой стороны от"),
    BY_RIGHT_HAND("по правую руку от"),
    BY_LEFT_HAND("по левую руку от"),
    BEHIND("сзади"),
    AHEAD("спереди");

    private String direction;

    Direction(String direction) {
        this.direction = direction;
    }

    public String getDirection() {
        return direction;
    }

}
```

src/characters/groups/Crowd.java

```java
package characters.groups;

import java.util.ArrayList;
import java.util.List;
import exceptions.CantAttendException;

public class Crowd {
    private List<Member> members = new ArrayList<>();

    // Статический вложенный класс Member
    public static class Member {
        private String name;
        public Member(String name) {
            this.name = name;
            if (name == "Карта Король") {
                throw new CantAttendException("Короли не присоединяются к толпе");
            }
        }

        public void attend() {
            System.out.println(this.name + " собирается в толпу");
        }
    }

    // Метод для добавления нового участника в толпу
    public void addMember(Member member) {
        members.add(member);
    }

    // Метод для иллюстрации присутствия толпы
    public void gather() {
        for (Member member : members) {
            member.attend();
        }
    }
}

```

Диаграмма классов:



Вывод программы:

```
Король сидит на Трон
Королева сидит на Трон
Алиса прибежал
Грифон ещё раз пытается найти путь
Грифон прибежал
Пичужка 1 собирается в толпу
Пичужка 2 собирается в толпу
Зверушка 1 собирается в толпу
Зверушка 2 собирается в толпу
Карта Валет собирается в толпу
Карта Туз собирается в толпу
Левый солдат охраняет Червонный Валет
Левый солдат находится по левую руку от Червонный Валет
Правый солдат охраняет Червонный Валет
Правый солдат находится по правую руку от Червонный Валет
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
Алиса рассматривает судья
Алиса рассматривает трон
Алиса рассматривает зверюшки
Алиса рассматривает пичужки
Алиса рассматривает карты
"Вот это судья. Кто в большом парике, тот и судья" - думает Алиса
Хотя Алиса раньше никогда не бывала в суде, она устала про суд в
книжках, и ей было очень приятно отметить, что она знает, как тут все -
или почти все - называется.
```

Выводы:

В результате проделанной работы, я изучил виды исключений и научился их обрабатывать. Кроме того, я узнал о локальных, анонимных и вложенных классах и поработал с ними на практике.