Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

Синтез помехоустойчивого кода

Вариант 32856

Группа: P3132

Выполнил: Овчаренко Александр Андреевич

Проверил: к.т.н. преподаватель Белозубов А.В.

Санкт-Петербург

2021

Оглавление

Задание

Написать программу на основе предложенного текста.

Программа должна удовлетворять следующим требованиям:

- Доработанная модель должна соответствовать принципам SOLID.
- Программа должна содержать как минимум два интерфейса и один абстрактный класс (номенклатура должна быть согласована с преподавателем).
- В разработанных классах должны быть переопределены методы equals(), toString() и hashCode().
- 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 работы

```
package lab_3.kids;

public class Korotishi {
    private final String name;
    private final Gender gender;
    public static Korotishi malishi = new Korotishi("Malishi", Gender.Malishi);
    public static Korotishi malishki = new Korotishi("Malishki",
Gender.Malishki);

    private Korotishi(String name, Gender gender) {
        this.name = name;
        this.gender = gender;
    }

    public String getGender() {
        return gender.getGender();
    }

    @Override
    public int hashCode() {
        return name.hashCode() + gender.hashCode();
    }

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

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

    public static Korotishi createMalish(String name) {
        if (name.equals("Malishi")) throw new IllegalArgumentException(name + " -
wrong name");
        else {return new Korotishi(name, Gender.Malishi);}
    }

    public static Korotishi createMalishka(String name) {
        if (name.equals("Malishki")) throw new IllegalArgumentException(name + "
- wrong name");
        else {return new Korotishi(name, Gender.Malishki);}
    }
}
```

```java
package lab_3.kids;

public enum Gender {
    Malishi("Malish"),
    Malishki("Malishki");

    private String gender;

    Gender(String gender) {
        this.gender = gender;
    }

    public String getGender() {
        return gender;
    }
}
```

```java
package lab_3.subjects;


public abstract class ThisObject {
    private final String object;
    private String description;

    public ThisObject (String object, String description) {
        this.object = object;
        this.description = description;
    }

    public String getObject() {
        return object;
    }

    public void setDesc(String description) {
        this.description = description;
    }

    public void printDesc() {
        System.out.printf("%s's description: %p", object, description);
    }

    public boolean equals(ThisObject obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        return object == obj.object;
    }

    @Override
    public String toString() {
        return object;
```

```
    }
}
```

```
package subjects;

public class Sok extends ThisObject{
    protected final String flower;

    public Sok(String flower) {
        super("sok", null);
        this.flower = flower;
    }

    public Sok(String flower, String description) {
        super("sok", description);
        this.flower = flower;
    }

    public String getFlower() {
        return flower;
    }
}
```

```
package actions;
import kids.Korotishi;
import subjects.*;

public class CollectAction {
    protected final String name = "collect";
    protected final ThisObject object;
    protected final String reason;
    protected String container;
    protected Korotishi[] people;

    public CollectAction(String reason, String container, Korotishi[] people,
ThisObject object) {
        this.reason = reason;
        this.container = container;
        this.people = people;
        this.object = object;
    }

    public void setContainer(String container) {
        this.container = container;
    }

    public void changePeople(Korotishi[] people) {
        this.people = people;
    }
```

```
    public void printResult() {
        System.out.print(people[0]);
        for (int i = 1; i < people.length; i++) {System.out.print(", " +
people[i]);}
        System.out.println(" " + name + " " + object + " " + container + " " +
reason + ".");
    }
}
```

```
package actions;

import subjects.ThisObject;

public interface DescribeCollectAction {
    void describeProcess(ThisObject thisObject);
}
```

```
package actions;

import subjects.Sok;
import subjects.ThisObject;

public class ProcessCollectSok implements DescribeCollectAction{

    @Override
    public void describeProcess(ThisObject thisObject) {
        try {
            Sok sok = (Sok) thisObject;
            System.out.println("1. Go to " + sok.getFlower() + ".");
            System.out.println("2. Make an incision.");
                System.out.println("3. Collect sok.");
            if (sok.getFlower().equals("flowers as ficus")) {
                for(int i = 0; i < 3; i++) {
                    System.out.printf("Wait %s hours\n", (3 - i));
                }
                System.out.println("The rubber is ready!");
            }
        }
        catch(Exception e) {
            System.out.printf("Exception: %s is not sok", e, thisObject);
        }
    }
}
```

```
package actions;
import kids.Korotishi;

public class GivecmdAction {
    protected final String name = "give";
    protected final Korotishi subject;
    protected final Korotishi object;
    protected final String what_do;

    public GivecmdAction(Korotishi subject, Korotishi object, String what_do) {
        this.subject = subject;
        this.object = object;
        this.what_do = what_do;
    }

    public void print() {
        System.out.println(subject + " " + name + " " + object + " " + what_do +
".");
    }
}
```

```
package actions;
import kids.*;
import subjects.*;

public interface GetThisObjectAction {
    void getObjectAction (String reason, String container, Korotishi[] pearson,
ThisObject thisObject);
}
```

```
package actions;
import kids.Korotishi;
import subjects.*;

public class GetSokAction implements GetThisObjectAction{

    @Override
    public void getObjectAction(String reason, String container, Korotishi[]
pearson, ThisObject thisObject) {
        try {
            Sok sok = (Sok) thisObject;
            CollectAction collSok = new CollectAction(reason, container, pearson,
sok);
            collSok.printResult();
            System.out.println("<< -- start collect sok -- >>");
            DescribeCollectAction decribe = new ProcessCollectSok();
            decribe.describeProcess(sok);
            System.out.println("<< -- finish collect sok -- >>");
```

```
        }
        catch(Exception e) {
            System.out.printf("Exception: %s is not sok", thisObject);
        }
    }
}
```

```
package actions;
import kids.*;

public class MeetAction {
    protected final Korotishi subject;
    protected final Korotishi object;
    protected final String name = "meet";

    public MeetAction(Korotishi subject, Korotishi object) {
        this.subject = subject;
        this.object = object;
    }

    public void print() {
        System.out.println(subject + " " + name + " " + object + ".");
    }
}
```

```
package actions;
import kids.Korotishi;

public class GoAction {
    protected final String name = "go";
    protected final Korotishi subject;
    protected String reason;

    public GoAction(Korotishi subject, String reason) {
        this.subject = subject;
        this.reason = reason;
    }

    public void changeReason(String reason) {
        this.reason = reason;
    }

    public void print() {
        System.out.println(subject + " " + name + " " + reason + ".");
    }
}
```

```
package actions;

import kids.Korotishi;

public class PlayAction {
```

```
    protected final String name = "play";
    protected final Korotishi subject;
    protected final Korotishi other;
    protected String game;

    public PlayAction(Korotishi subject, Korotishi other, String game) {
        this.subject = subject;
        this.game = game;
        this.other = other;
    }

    public void changeGame(String game) {
        this.game = game;
    }

    public void print() {
        System.out.println(subject + " " + name + " with " + other + " " + game +
".");
    }
}
```

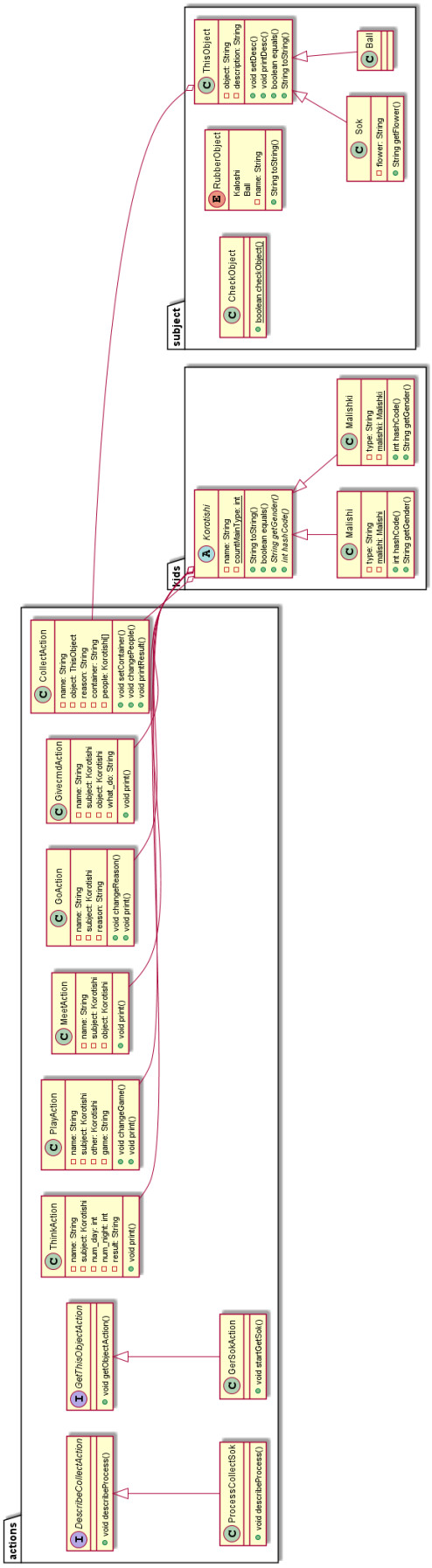
```
package actions;
import kids.Korotishi;

public class ThinkAction {
    protected final String name = "think";
    protected final Korotishi subject;
    protected final int num_day;
    protected final int num_night;
    protected final String result;

    public ThinkAction(Korotishi subject, int day, int night, String result) {
        this.subject = subject;
        num_day = day;
        num_night = night;
        this.result = result;
    }

    public void print() {
        System.out.println(subject + " " + name + ". He has been thinking for " +
num_day +
        " days and " + num_night + " night. Result - " + result + ".");
    }
}
```

Uml-диаграмма



Исполняемы код:

```
package lab_3;
import lab_3.actions.*;
import lab_3.kids.*;
import lab_3.subjects.*;

public class Main {
    public static void main(String[] args) {
        Korotishi znayka = Korotishi.createMalish("Znayka");
        Korotishi neznayka = Korotishi.createMalish("Neznayka");
        Korotishi gynka = Korotishi.createMalish("Gynka");
        Korotishi simka = Korotishi.createMalish("Simka");
        Korotishi simka2 = Korotishi.createMalish("Simka");
        Korotishi shpula = Korotishi.createMalish("Shpula");
        ThinkAction znthoughts = new ThinkAction(znayka, 3, 3, "to make rubber
ball");
        znthoughts.print();
        GivecmdAction collSokCmd = new GivecmdAction(znayka, Korotishi.malishi,
"start to collect sok");
        collSokCmd.print();
        Sok sokForBall = new Sok("flowers as ficus", "for big ball");
        GetThisObjectAction getSok = new GetSokAction();
        getSok.getObjectAction("for big ball", "in big bottle", new
Korotishi[]{Korotishi.malishi, znayka},
        sokForBall);
        GoAction goNez = new GoAction(neznayka, "to collect sok");
        goNez.print();
        MeetAction meetNez = new MeetAction(neznayka, gynka);
        meetNez.print();
        PlayAction gameGyn = new PlayAction(gynka, Korotishi.malishki, "hight -
seek");
        gameGyn.print();
        System.out.println(simka.hashCode());
        System.out.println(simka2.hashCode());
        System.out.println(simka.equals(simka2));
        System.out.println(simka2.equals(shpula));
    }
}
```

Результат вывода программы:

Znayka think. He has been thinking for 3 days and 3 night. Result - to make rubber ball.

Znayka give Malishi command start to collect sok.

Malishi, Znayka collect sok in big bottle for big ball.

<< -- start collect sok -- >>

1. Go to flowers as ficus.

2. Make an incision.

3. Collect sok.

Wait 3 hours

Wait 2 hours

Wait 1 hours

The rubber is ready!

<< -- finish collect sok -- >>

Neznayka go to collect sok.

Neznayka meet Gynka.

Gynka play with Malishki hight - seek.

1559898701

1559898701

true

false

Итоги

В результате выполнения исследовательской работы были изучены принципы объектно-ориентированного программирования SOLID, понятие интерфейса: что это такое и зачем они нужны? Также был рассмотрен класс Object и его методыю. В процессе выполнения работы пришлось пересмотреть подход к написанию программ: лучше сначала придумать реализацию, а после начинать писать код.