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 №4

Вариант №32138.6

Группа: P3132

Выполнил: Русинов Дмитрий Станиславович

Преподаватель: Горбунов Михаил Витальевич

Санкт-Петербург

2021

Оглавление

Текст задания

Лабораторная работа #4

Доработать программу из лабораторной работы #3, обновив реализацию объектной модели в соответствии с новой версией описания предметной области.

Введите вариант: 32138.6

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

> - Я не понимаю,- очень робко, боясь опять рассердить Соню, начала Алиса,- как же они таскали оттуда мармелад? - Из обыкновенного колодца таскают воду,- сказал Шляпа,- а из мармеладного колодца всякий может, я надеюсь, таскать мармелад. Ты что - совсем дурочка? - Я говорю, как они могли таскать мармелад оттуда? Ведь они там жили- сказала Алиса,- решив оставить без ответа последние слова Шляпы. - Не только жили! - сказала Соня.- Они жили-были! И этот ответ настолько ошеломил бедную Алису, что она позволила Соне некоторое время продолжать рассказ без вынужденных остановок. Это было весьма кстати, так как рассказчица отчаянно зевала и усиленно терла глаза. - Так вот,- продолжала Соня,- этот самый мармадад они ели и пили - делали что хотели... Тут Алиса не выдержала.

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.

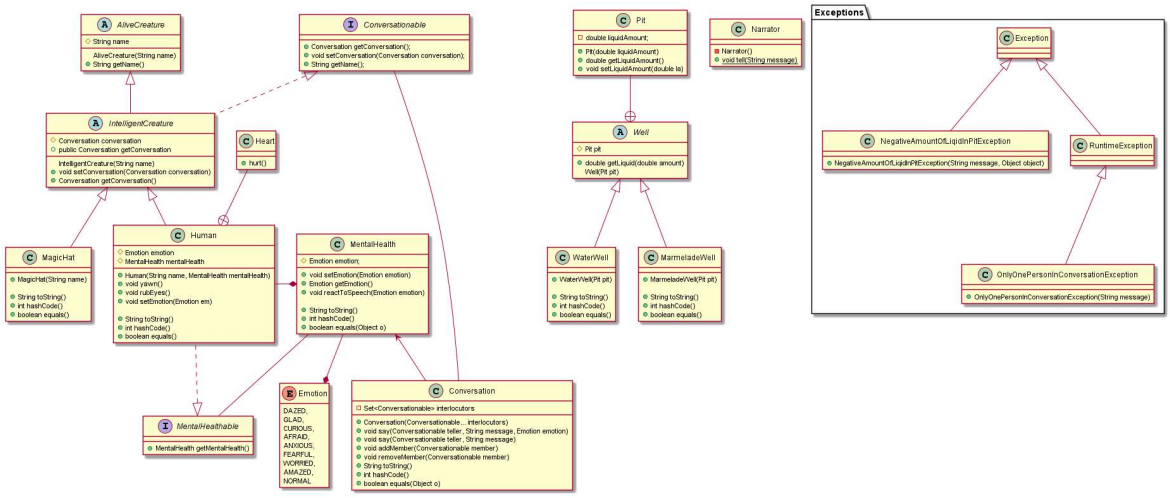
Отчёт по работе должен содержать:

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`.

UML диаграмма классов



Исходный код программы

Main

```
import abstractThings.Conversation;
import abstractThings.Emotion;
import abstractThings.MentalHealth;
import abstractThings.Narrator;
import creatures.Human;
import creatures.MagicHat;
import things.WaterWell;
import things.Well;

public class Main {
    public static void main(String[] args) {
        Well.Pit pit = new Well.Pit(100);
        WaterWell well = new WaterWell(pit);

        Human alice = new Human("Alice", new MentalHealth());
        Human sonya = new Human("Sonya", new MentalHealth());
        MagicHat hat = new MagicHat("Hat");

        Conversation conversation = new Conversation(alice, sonya, hat);

        conversation.say(alice, "Я не понимаю, как же они могли таскать
оттуда мармелад?", null);

        conversation.say(hat, "Из обыкновенного колодца таскают воду, а из
мармеладного колодца всякий может, " +
                "я надеюсь, таскать мармелад", null);

        conversation.say(alice, "Ты что, совсем дурочка? Я говорю, как они
могли таскать мармелад оттуда?" +
                " Ведь они там жили.", null);

        conversation.say(sonya,"Не только жили! Они жили-были!",
Emotion.DAZED);

        sonya.yawn();
        sonya.rubEyes();

        conversation.say(sonya,"Так вот, делали что хотели...",
Emotion.ANGRY);

        // выдуманная вещь далее:

        // Допустим, в тексте был бы эмоционально стабильный человек (и это
был бы единичный случай),
        // тогда его можно объект можно было бы объявить таким образом,
        // используя анонимный класс.
        Human tom = new Human("Thomas", new MentalHealth(){
            @Override
            public void reactToSpeech(Emotion emotion) {
                Narrator.tell("Thomas never cares about teller's intonation
so his mood was completely stable");
            };
        });
    }
}
```

Emotion

```java
package abstractThings;

public enum Emotion {
    AFRAID("afraid"),
    AMAZED("amazed"),
    ANGRY("angry"),
    ANXIOUS("anxious"),
    CURIOUS("curious"),
    DAZED("dazed"),
    FEARFUL("fearful"),
    GLAD("glad"),
    NORMAL("normal"),
    WORRIED("worried");

    Emotion(String value) {
        this.value = value;
    }

    private final String value;

    public String getValue() {
        return value;
    }
}
```

Conversation

```
package abstractThings;

import exceptions.OnlyOnePersonInConversationException;
import interfaces.Conversationable;
import interfaces.MentalHealthable;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Objects;
import java.util.Set;

public class Conversation {
    private final Set<Conversationable> interlocutors;

    public Conversation(Conversationable... interlocutors) {
        this.interlocutors = new HashSet<>(Arrays.asList(interlocutors));
    }

    public void say(Conversationable teller, String message, Emotion emotion)
{
        if (interlocutors.contains(teller)) {
            Set<Conversationable> listeners = new HashSet<>(interlocutors);
            listeners.remove(teller);

            // Нет никакого смысла в диалоге из одного человека
            if (listeners.isEmpty()) {
                throw new OnlyOnePersonInConversationException("Please add
more participants to the conversation");
            }

            String listenersNames = "";
            for (Conversationable i : listeners) {
                listenersNames += i.getName() + ", ";
            }
            listenersNames = listenersNames.substring(0,
listenersNames.length() - 2);

            Narrator.tell(teller.getName() + " told: \"" + message + "\" to "
+ listenersNames + ".");

            if (emotion != null) {
                // Каждая сущность из listeners, обладающая ментальным
здоровьем, отреагирует на слова teller'а
                for (Conversationable i : listeners) {
                    if (i instanceof MentalHealthable) {
                        System.out.print(i.getName() + ": ");
                        ((MentalHealthable)
i).getMentalHealth().reactToSpeech(emotion);
                    }
                }
            }
        } else {
            Narrator.tell(teller.getName() + "was not in the conversation so
nobody heard anything");
        }
    }
```

```
    public void addMember(Conversationable member) {
        interlocutors.add(member);
        member.setConversation(this);
    }

    public void removeMember(Conversationable member) {
        interlocutors.remove(member);
        member.setConversation(null);
    }

    @Override
    public String toString() {
        return interlocutors.toString();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Conversation that = (Conversation) o;
        return interlocutors.equals(that.interlocutors);
    }

    @Override
    public int hashCode() {
        return Objects.hash(interlocutors);
    }
}
```

MentalHealth

```
package abstractThings;

import java.util.Objects;

public class MentalHealth {
    protected Emotion emotion = Emotion.NORMAL;

    public void setEmotion(Emotion emotion) {
        this.emotion = emotion;
    }

    public Emotion getEmotion() {
        return emotion;
    }


    public void reactToSpeech(Emotion emotion) {
        if (emotion != Emotion.NORMAL) {
            setEmotion(emotion);
            Narrator.tell("is now " + emotion.getValue() + " because of these
words");
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        MentalHealth that = (MentalHealth) o;
        return emotion == that.emotion;
    }

    @Override
    public int hashCode() {
        return Objects.hash(emotion);
    }

    @Override
    public String toString() {
        return "MentalHealth{" +
                "emotion=" + emotion +
                '}';
    }
}
```

Narrator

```
package abstractThings;

public class Narrator {
    private Narrator(){}
    public static void tell(String message) {
        System.out.println(message);
    }
}
```

AliveCreature

```
package creatures;

public abstract class AliveCreature {
    final protected String name;
    AliveCreature(String name){
        this.name = name;
    }

    public String getName(){
        return name;
    }
}
```

Human

```java
package creatures;

import abstractThings.Emotion;
import abstractThings.MentalHealth;
import abstractThings.Narrator;
import interfaces.MentalHealthable;

import java.util.Objects;


public class Human extends IntelligentCreature implements MentalHealthable {

    protected final MentalHealth mentalHealth;
    protected final Heart heart;

    public Human(String name, MentalHealth mentalHealth) {
        super(name);
        this.mentalHealth = mentalHealth;
        this.heart = new Heart();
    }

    protected class Heart {
        public void hurt() { getMentalHealth().setEmotion(Emotion.WORRIED); }
    }

    public void yawn() {
        Narrator.tell(getName() + " yawns");
    }

    public void rubEyes() {
        Narrator.tell(getName() + " rubs eyes");
    }


    @Override
    public MentalHealth getMentalHealth() {
        return this.mentalHealth;
    }

    @Override
    public String toString() {
        return "Human{" +
                "mentalHealth=" + mentalHealth + ", name=" + name +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Human human = (Human) o;
        return mentalHealth.equals(human.mentalHealth);
    }

    @Override
    public int hashCode() {
        return Objects.hash(mentalHealth, name);
    }
}
```

IntelligentCreature

```java
package creatures;

import abstractThings.Conversation;
import interfaces.Conversationable;

public abstract class IntelligentCreature extends AliveCreature implements
Conversationable {

    protected Conversation conversation;

    IntelligentCreature(String name){
        super(name);
    }

    @Override
    public Conversation getConversation(){
        return this.conversation;
    }

    @Override
    public void setConversation(Conversation conversation) {
        this.conversation = conversation;
    }
}
```

MagicHat

```java
package creatures;

import java.util.Objects;

public class MagicHat extends IntelligentCreature{
    public MagicHat(String name){
        super(name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(getName());
    }

    @Override
    public String toString() {
        return "MagicHat{" +
                "name='" + getName() + '\'' +
                '}';

    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        MagicHat human = (MagicHat) o;
        return getName().equals(human.getName());
    }
}
```

NegativeAmountOfLiquidInPitException

```
package exceptions;

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

OnlyOnePersonInConversationException

```
package exceptions;

public class OnlyOnePersonInConversationException extends RuntimeException{

    public OnlyOnePersonInConversationException(String message){
        super(message);
    }

}
```

Well

```
package things;

import abstractThings.Narrator;
import exceptions.NegativeAmountOfLiquidInPitException;

public abstract class Well {
    protected Pit pit;

    Well(Pit pit) {
        this.pit = pit;
    }

    public static class Pit{
        public Pit(double liquidAmount){
            this.liquidAmount = liquidAmount;
        }

        private double liquidAmount;
        public double getLiquidAmount(){
            return liquidAmount;
        }
        public void setLiquidAmount (double liquidAmount) throws
NegativeAmountOfLiquidInPitException {
            if (liquidAmount < 0)
                throw new NegativeAmountOfLiquidInPitException("You can't set
a negative number to liquidAmount");
            this.liquidAmount = liquidAmount;
        }
    }

    public void getLiquid(double amount) {
        try{
            pit.setLiquidAmount(pit.getLiquidAmount() - amount);
            Narrator.tell("The Well now has less liquid, got it
successfully.");
        }
        catch (NegativeAmountOfLiquidInPitException e){
            Narrator.tell("Well, well did not have enough liquid.");
        }
    }
}
```

MarmeladeWell

```
package things;

public class MarmeladeWell extends Well {
    public MarmeladeWell(Pit pit) {
        super(pit);
    }
}
```

WaterWell

```
package things;

public class WaterWell extends Well {
    public WaterWell(Pit pit) {
        super(pit);
    }
}
```

Conversationable

```
package interfaces;

import abstractThings.Conversation;
import abstractThings.Emotion;

public interface Conversationable {
    public Conversation getConversation();
    public void setConversation(Conversation conversation);
    public String getName();
}
```

MentalHealthable

```
package interfaces;

import abstractThings.MentalHealth;

public interface MentalHealthable {
    public MentalHealth getMentalHealth();
}
```

Выходные данные программы

```
Alice told: "Я не понимаю, как же они могли таскать оттуда мармелад?" to Sonya, Hat.
Hat told: "Из обыкновенного колодца таскают воду, а из мармеладного колодца всякий может, я надеюсь, таскать мармелад" to Alice, Sonya.
Alice told: "Ты что, совсем дурочка? Я говорю, как они могли таскать мармелад оттуда? Ведь они там жили." to Sonya, Hat.
Sonya told: "Не только жили! Они жили-были!" to Alice, Hat.
Alice: is now dazed because of these words
Sonya yawns
Sonya rubs eyes
Sonya told: "Так вот, делали что хотели..." to Alice, Hat.
Alice: is now angry because of these words
```

Заключение

Для выполнения этой лабораторной работы было необходимо изучить различные типы классов: анонимные, вложенные (static&non-static), а также узнать о типах исключений в Java, о том, как их обрабатывать и о том, как создавать собственные.