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 №3

Вариант №32138.4

Группа: P3132

Выполнил: Русинов Дмитрий Станиславович

Преподаватель: Горбунов Михаил Витальевич

Санкт-Петербург

2021

Оглавление

Текст задания

Введите вариант: 32138.4

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

> - 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...

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.

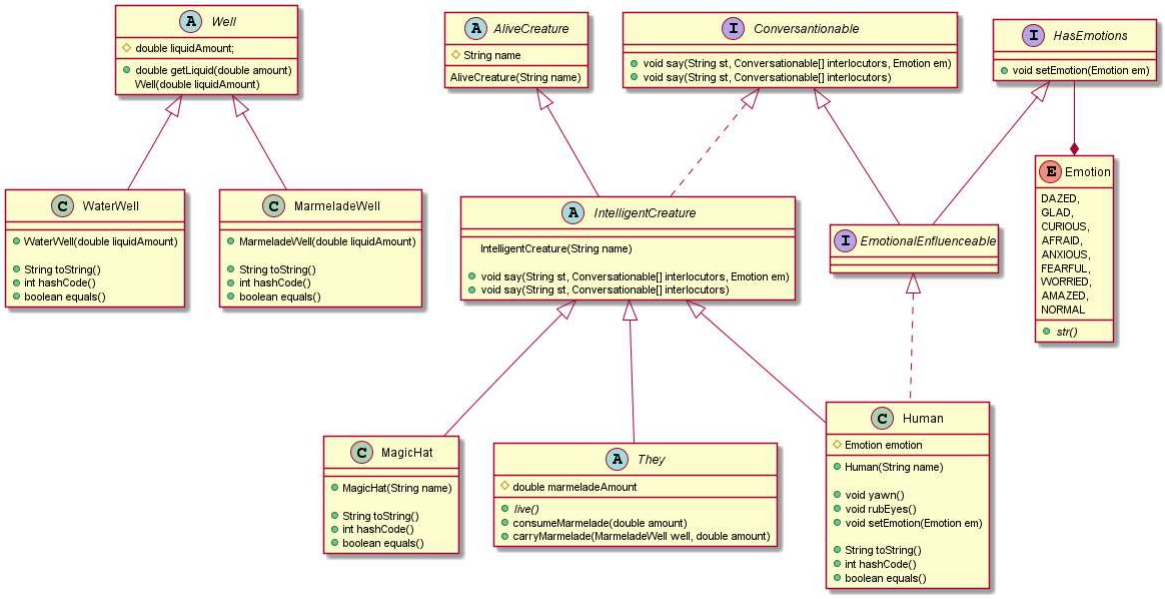
Отчёт по работе должен содержать:

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

Вопросы к защите лабораторной работы:

1. Принципы объектно-ориентированного программирования SOLID и STUPID.
2. Класс `Object`. Реализация его методов по умолчанию.
3. Особенности реализации наследования в Java. Простое и множественное наследование.
4. Понятие абстрактного класса. Модификатор `abstract`.
5. Понятие интерфейса. Реализация интерфейсов в Java, методы по умолчанию. Отличия от абстрактных классов.
6. Перечисляемый тип данных (enum) в Java. Особенности реализации и использования.
7. Методы и поля с модификаторами `static` и `final`.
8. Перегрузка и переопределение методов. Ковариванты возвращаемых типов данных.
9. Элементы функционального программирования в синтаксисе Java. Функциональные интерфейсы, лямбда-выражения. Ссылки на методы.

UML диаграмма классов



Исходный код программы

Main

```
import abstractThings.Emotion;
import creatures.Human;
import creatures.MagicHat;
import interfaces.Conversationable;

public class Main {
    public static void main(String[] args) {
        Human alice = new Human("Alice");
        Human sonya = new Human("Sonya");

        MagicHat hat = new MagicHat("The Hat");


        hat.say("Из обыкновенного колодца таскают воду," +
                " а из мармеладного колодца всякий может, я надеюсь, таскать
мармелад.", new Conversationable[]{alice, sonya});

        alice.say("Ты что - совсем дурочка? " +
                "- Я говорю, как они могли таскать мармелад оттуда? Ведь они
там жили", new Conversationable[]{hat, sonya});

        sonya.say("Не только жили! Они жили-были!", new
Conversationable[]{hat, alice}, Emotion.DAZED);

        System.out.println("\n");

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

        System.out.println("\n");

        sonya.say("Так вот, этот самый мармадад они ели и пили" +
                " - делали что хотели...", new Conversationable[]{hat,
alice});


    }
}
```

Emotion

```
package abstractThings;

public enum Emotion {
    DAZED{
        @Override
        public String str() {
            return "dazed";
        }
    },
    GLAD{
        @Override
        public String str() {
            return "glad";
        }
    },
    CURIOUS{
        @Override
        public String str() {
            return "curious";
        }
    },
    AFRAID{
        @Override
        public String str() {
            return "afraid";
        }
    },
    ANXIOUS{
        @Override
        public String str() {
            return "anxious";
        }
    },
    FEARFUL{
        @Override
        public String str() {
            return "fearful";
        }
    },
    WORRIED{
        @Override
        public String str() {
            return "worried";
        }
    },
    AMAZED{
        @Override
        public String str() {
            return "amazed";
        }
    },
    NORMAL{
        @Override
        public String str() { return "normal"; }
    };

    public abstract String str();

}
```

AliveCreature

```java
package creatures;

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

Human

```java
package creatures;

import abstractThings.Emotion;
import interfaces.EmotionalEnfluenceable;

import java.util.Objects;

public class Human extends IntelligentCreature implements
EmotionalEnfluenceable {

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

    protected Emotion emotion = Emotion.NORMAL;

    public void yawn() {
        System.out.println(this.name + " yawns");
    }

    public void rubEyes() {
        System.out.println(this.name + " rubs eyes");
    }

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

    public Emotion getEmotion() {
        return this.emotion;
    }

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

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

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

IntelligentCreature

```
package creatures;

import abstractThings.Emotion;
import interfaces.Conversationable;
import interfaces.EmotionalEnfluenceable;

public abstract class IntelligentCreature extends AliveCreature implements
Conversationable {

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

    @Override
    public void say(String st, Conversationable[] interlocutors, Emotion em){
        System.out.println(this.name + " said\": " + st + "\".");

        for (Conversationable interlocutor: interlocutors){
            if (interlocutor instanceof EmotionalEnfluenceable){

                ((EmotionalEnfluenceable) interlocutor).setEmotion(em);
                if(interlocutor instanceof AliveCreature){
                    System.out.println(((AliveCreature) interlocutor).name +
" is now " + em.str() + " because of these words.");
                }

                else{
                    System.out.println("The thing is now" + em.str() +
"because of these words.");
                }
            }
        }

    }

    @Override
    public void say(String st, Conversationable[] interlocutors) {
        System.out.println(this.name + " said\": " + st + "\".");
    }


}
```

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(name);
    }

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

    }

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

Well

```
package things;

public abstract class Well {
    protected double liquidAmount;

    Well(double liquidAmount) {
        this.liquidAmount = liquidAmount;
    }

    public double getLiquid(double amount) {
        if (liquidAmount - amount < 0) {
            System.out.println("Can't use the Well because you want too much
liquid.");
            return 0;
        } else {
            liquidAmount -= amount;
            System.out.println("The Well now has less liquid, got it
successfully.");
            return amount;
        }
    }
}
```

MarmeladeWell

```
package things;

public class MarmeladeWell extends Well {
    public MarmeladeWell(double liquidAmount) {
        super(liquidAmount);
    }
}
```

WaterWell

```
package things;

public class WaterWell extends Well {
    public WaterWell(double liquidAmount) {
        super(liquidAmount);
    }
}
```

Conversationable

```
package interfaces;

import abstractThings.Emotion;

public interface Conversationable {
    public void say(String st, Conversationable[] interlocutors, Emotion em);
    public void say(String st, Conversationable[] interlocutors);
}
```

HasEmotions

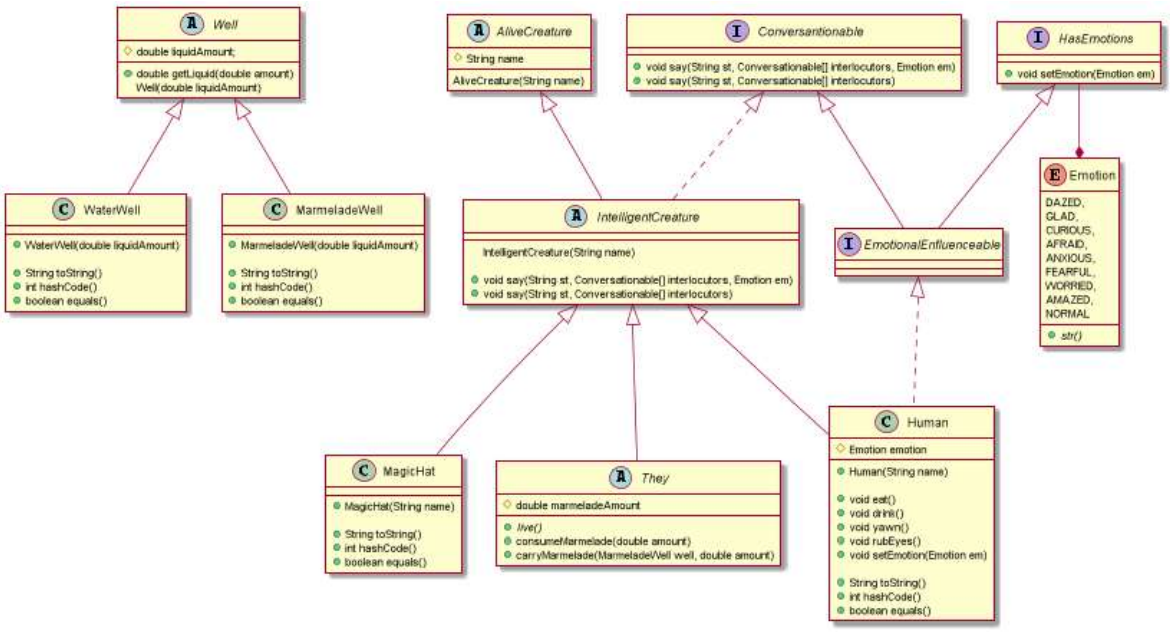
```
package interfaces;

import abstractThings.Emotion;

public interface HasEmotions {
    void setEmotion(Emotion em);
}
```

EmotionalEnfluenceable

```
package interfaces;

public interface EmotionalEnfluenceable extends  Conversationable,
HasEmotions {
}
```

They

```
package creatures;

import things.MarmeladeWell;

public abstract class They extends IntelligentCreature{
    protected double marmeladeAmount = 0;

    public They(){
        super("they");
    }

    public abstract void live();

    public void carryMarmelade(MarmeladeWell well, double amount){
        double got = well.getLiquid(amount);
        this.marmeladeAmount += got;
    }

    public void consumeMarmelade(double amount){
        if (marmeladeAmount - amount < 0){
            System.out.println(name + " wanted to consume more marmelade than
they have");
        } else {
            System.out.println(name + "successfully consumed " + amount + "
of Marmelade");
        }
    }
}
```


Well
double liquidAmount;
double getLiquid(double amount)
Well(double liquidAmount)
AliveCreature
String name
AliveCreature(String name)
Conversationable
void say(String st, Conversationable[] interlocutors, Emotion em)
void say(String st, Conversationable[] interlocutors)
HasEmotions
void setEmotion(Emotion em)
Emotion
DAZED,
GLAD,
CURIOUS,
AFRAID,
ANXIOUS,
FEARFUL,
WORRIED,
AMAZED,
NORMAL
str()
WaterWell
WaterWell(double liquidAmount)
String toString()
int hashCode()
boolean equals()
MarmeladeWell
MarmeladeWell(double liquidAmount)
String toString()
int hashCode()
boolean equals()
IntelligentCreature
IntelligentCreature(String name)
void say(String st, Conversationable[] interlocutors, Emotion em)
void say(String st, Conversationable[] interlocutors)
EmotionalEnfluenceable
MagicHat
MagicHat(String name)
String toString()
int hashCode()
boolean equals()
They
double marmeladeAmount
live()
consumeMarmelade(double amount)
carryMarmelade(MarmeladeWell well, double amount)
Human
Emotion emotion
Human(String name)
void eat()
void drink()
void yawn()
void rubEyes()
void setEmotion(Emotion em)
String toString()
int hashCode()
boolean equals()

Выходные данные программы

```
The Hat said": Из обыкновенного колодца таскают воду, а из мармеладного колодца всякий может, я надеюсь, таскать мармелад.".
Alice said": Ты что - совсем дурочка? - Я говорю, как они могли таскать мармелад оттуда? Ведь они там жили".
Sonya said": Не только жили! Они жили-были!".
Alice is now dazed because of these words.


Sonya yawns
Sonya rubs eyes


Sonya said": Так вот, этот самый мармадад они ели и пили - делали что хотели...".

Process finished with exit code 0
```

Заключение

Для выполнения этой лабораторной работы необходимо было изучить принципы SOLID, научиться работать с интерфейсами и enum’ами.