НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет систем управления и робототехники

Программирование

Лабораторная работа № 3

Выполнил студент

Репин Арсений Владиславович

Группа № R3135

Преподаватель: Мартин Райла

г. Санкт-Петербург

2020

Вариант 91329

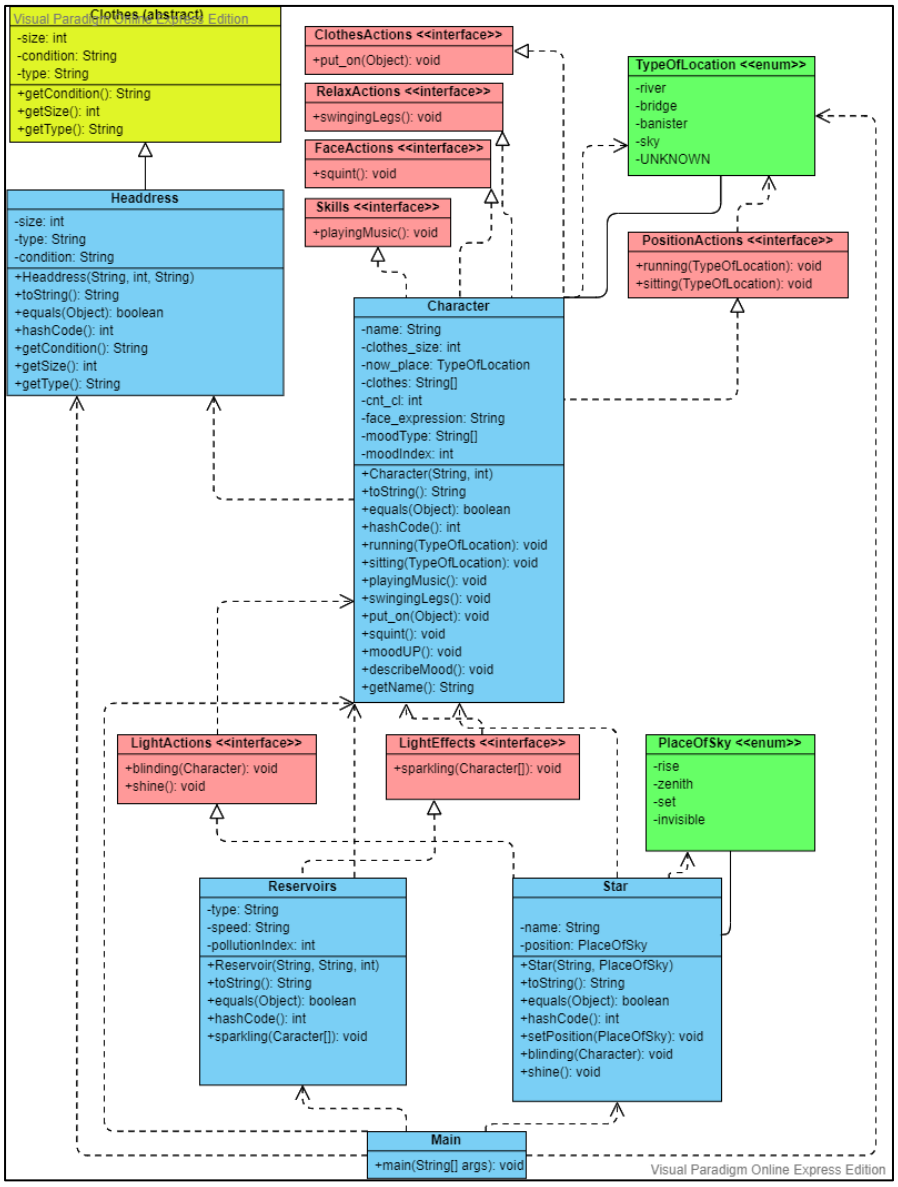
1. Задание.
   Описание предметной области, по которой должна быть построена объектная модель:

   “Муми-тролль побежал прямо на музыку и внизу у реки увидел Снусмумрика. Тот сидел на перилах моста, нахлобучив на лоб свою старую шляпу, и болтал над водой ногами. Солнце только что поднялось над верхушками деревьев и светило им прямо в лицо. А они жмурились от его лучей, болтали ногами над бегущей сверкающей водой, и на сердце у них было привольно и беззаботно.”

   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()*, *hashCode()*.
   4. Программа должна содержать как минимум один перечисляемый тип (*enum*).

2. Диаграмма классов.



3. Исходный код программы.

Main.java

```
package fairytale;

public class Main {

    public static void main(String[] args) {
        Character mummytroll = new Character("Mummy-troll",35);
        Character snufkin = new Character("Snufkin",40);
        snufkin.playingMusic();
        mummytroll.running(TypeOfLocation.river);
        snufkin.sitting(TypeOfLocation.banister);
        snufkin.put_on(new Headdress("hat",50,"old"));
        snufkin.swingingLegs();
        Star sun = new Star("Sun", PlaceOfSky.invisible);
        sun.setPosition(PlaceOfSky.rise);
        sun.shining();
        sun.blinding(mummytroll);
        sun.blinding(snufkin);
        mummytroll.sitting(TypeOfLocation.bridge);
        mummytroll.swingingLegs();
        snufkin.sitting(TypeOfLocation.bridge);
        snufkin.swingingLegs();
        Reservoirs river = new Reservoirs("river", "running", 0);
        river.sparkling(new Character[]{mummytroll,snufkin});
        mummytroll.describeMood();
        snufkin.describeMood();
    }
}
```

Character.java

```
package fairytale;
import java.util.Objects;

public class Character implements Skills, RelaxActions, PositionActions,
ClothesActions, FaceActions  {
    private final String name;
    private final int clothes_size;
    private TypeOfLocation now_place = TypeOfLocation.UNKNOWN;
    private String[] clothes = new String[10];
    private int cnt_cl = 0;
    private String face_expression = "normal";
    private final String[] moodType = new
String[]{"terrible","depressed","bad","normal","good","great","free","carefree"};
    private int moodIndex = 4;

    Character (String name, int clothes_size){
        this.name = name;
        this.clothes_size = clothes_size;
    }

    @Override
    public String toString() {
        return "Character {" + "name = " + name + ", " + "clothes size = " +
clothes_size + ", " + "location = " + now_place + ", " + "mood = " +
moodType[moodIndex] + "}";
    }

    @Override
    public boolean equals(Object obj) {
        if(this == obj) return true;
        if(!(obj instanceof Character)) return false;
```

```
        Character eq_char = (Character)obj;
        return Objects.equals(name, eq_char.name) && Objects.equals(clothes_size,
eq_char.clothes_size) && Objects.equals(now_place, eq_char.now_place);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, clothes_size, cnt_cl, moodType[moodIndex]);
    }

    @Override
    public void running(TypeOfLocation finish_place) {
        if(finish_place == now_place){
            System.out.println(name + " is running on the spot");
        }else {
            if (now_place == TypeOfLocation.UNKNOWN) {
                System.out.println(name + " is running to the " + finish_place);
            } else {
                System.out.println(name + " is running from the " + now_place + "
to the " + finish_place);
            }
        }
        now_place = finish_place;
    }

    @Override
    public void sitting(TypeOfLocation place) {
        if(place == TypeOfLocation.UNKNOWN) System.out.println(name + " is
sitting");
        else System.out.println(name + " is sitting on the " + place);
        now_place = place;
    }

    @Override
    public void playingMusic() {
        System.out.println(name + " is playing the music");
    }

    @Override
    public void swingingLegs() {
        if(moodIndex < moodType.length-1) moodIndex++;
        System.out.println(name + " is swinging legs, while sitting on the " +
now_place);
    }

    @Override
    public void put_on(Object obj) {
        if(obj instanceof Headdress) {
            Headdress thing = (Headdress) obj;
            if (thing.getSize() >= clothes_size) {
                if(cnt_cl < 10){
                    clothes[cnt_cl] = thing.getType();
                    cnt_cl++;
                    System.out.println(name + " put on the " + thing.getType());
                }else{
                    System.out.println(name + " can't put on the " +
thing.getType() + ". Too many clothes.");
                }
            }
        }
    }

    @Override
    public void squint() {
        face_expression = "squint";
    }
```

```
    public void moodUP(){
        if(moodIndex < moodType.length-1) moodIndex++;
    }

    public void describeMood(){
        System.out.println(name + " is in a " + moodType[moodIndex] + " mood");
    }

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

Clothes.java

```
package fairytale;

public abstract class Clothes {
    private String condition;
    private int size_cl;
    private String type;

    public String getCondition() {
        return condition;
    }

    public int getSize(){
        return size_cl;
    }

    public String getType(){
        return type;
    }
}
```

Headdress.java

```
package fairytale;

import java.util.Objects;

public class Headdress extends Clothes{
    private final int size;
    private final String type;
    private final String condition;

    Headdress(String type, int size, String condition) {
        this.size = size;
        this.type = type;
        this.condition = condition;
    }

    @Override
    public String toString() {
        return "Headdress {" + "type = " + type + ", " + "size = " + size + ", "
+ "condition = " + condition + "}";
    }

    @Override
    public boolean equals(Object obj) {
        if(this == obj) return true;
        if(!(obj instanceof Headdress)) return false;
        Headdress headdress = (Headdress) obj;
        return Objects.equals(size, headdress.size) && Objects.equals(type,
headdress.type) && Objects.equals(condition, headdress.condition);
    }

    @Override
    public int hashCode() {
```

```
        return Objects.hash(size, type, condition);
    }

    @Override
    public String getCondition() {
        return condition;
    }

    @Override
    public int getSize() {
        return size;
    }

    @Override
    public String getType(){return type;}
}
```

Reservoirs.java

```
package fairytale;
import java.util.Objects;

public class Reservoirs implements LightEffects{
    private final String reservoir_type;
    private String speed;
    private int pollutionIndex;

    Reservoirs (String reservoir_type, String speed, int pollutionIndex){
        this.reservoir_type = reservoir_type;
        this.speed = speed;
        this.pollutionIndex = pollutionIndex;
    }

    @Override
    public String toString() {
        return "Water object {" + "type = " + reservoir_type + ", " + "speed = "
+ speed + ", " + "pollution_index = " + pollutionIndex + "}";
    }

    @Override
    public boolean equals(Object obj) {
        if(this == obj) return true;
        if(!(obj instanceof Reservoirs)) return false;
        Reservoirs reservoir = (Reservoirs) obj;
        return Objects.equals(reservoir_type, reservoir.reservoir_type) &&
Objects.equals(speed, reservoir.speed) && Objects.equals(pollutionIndex,
reservoir.pollutionIndex);
    }

    @Override
    public int hashCode() {
        return Objects.hash(reservoir_type, speed, pollutionIndex);
    }

    @Override
    public void sparkling(Character[] heroes) {
        if(pollutionIndex <= 2) {
            System.out.println("The " + speed + " " + reservoir_type + " is
sparkling");
            for(Character character: heroes){
                character.moodUP();
            }
        }
    }
}
```

Star.java

```
package fairytale;

import java.util.Objects;

public class Star implements LightActions{
    private final String name;
    private PlaceOfSky position;

    Star(String name, PlaceOfSky position){
        this.name = name;
        this.position = position;
    }

    @Override
    public String toString() {
        return "Star {" + "name = " + name + ", " + "position = " + position +
"}";
    }

    @Override
    public boolean equals(Object obj) {
        if(this == obj) return true;
        if(!(obj instanceof Star)) return false;
        Star star = (Star) obj;
        return Objects.equals(name, star.name) && Objects.equals(position,
star.position);
    }

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

    public void setPosition(PlaceOfSky pos){
        if(!(position == pos)) {
            System.out.println("The " + name + " has just entered the " + pos);
            position = pos;
        }
        if(position == PlaceOfSky.zenith){
            this.shining();
        }
    }

    @Override
    public void blinding(Character hero) {
        hero.squint();
        System.out.println(hero.getName() + " is squinting because of the " +
name);
    }

    @Override
    public void shining() {
        System.out.println("The " + name + " is shining");
    }
}
```

Actions.java

```
package fairytale;

interface RelaxActions{
    void swingingLegs();
}

interface Skills{
```

```
    void playingMusic();
}

interface PositionActions{
    void running(TypeOfLocation finish_place);
    void sitting(TypeOfLocation place);
}

interface ClothesActions{
    void put_on(Object obj);
}

interface LightActions{
    void blinding(Character hero);
    void shining();
}

interface LightEffects{
    void sparkling(Character[] heroes);
}

interface FaceActions{
    void squint();
}
```

PlaceOfSky.java

```
package fairytale;

public enum PlaceOfSky {
    rise,
    zenith,
    set,
    invisible
}
```

TypeOfLocation.java

```
package fairytale;

public enum TypeOfLocation{
    river,
    bridge,
    banister,
    sky,
    UNKNOWN
}
```

4. Результат работы программы.

Snufkin is playing the music

Mummy-troll is running to the river

Snufkin is sitting on the banister

Snufkin put on the hat

Snufkin is swinging legs, while sitting on the banister

The Sun has just entered the rise

The Sun is shining

Mummy-troll is squinting because of the Sun

Snufkin is squinting because of the Sun

Mummy-troll is sitting on the bridge

Mummy-troll is swinging legs, while sitting on the bridge

Snufkin is sitting on the bridge

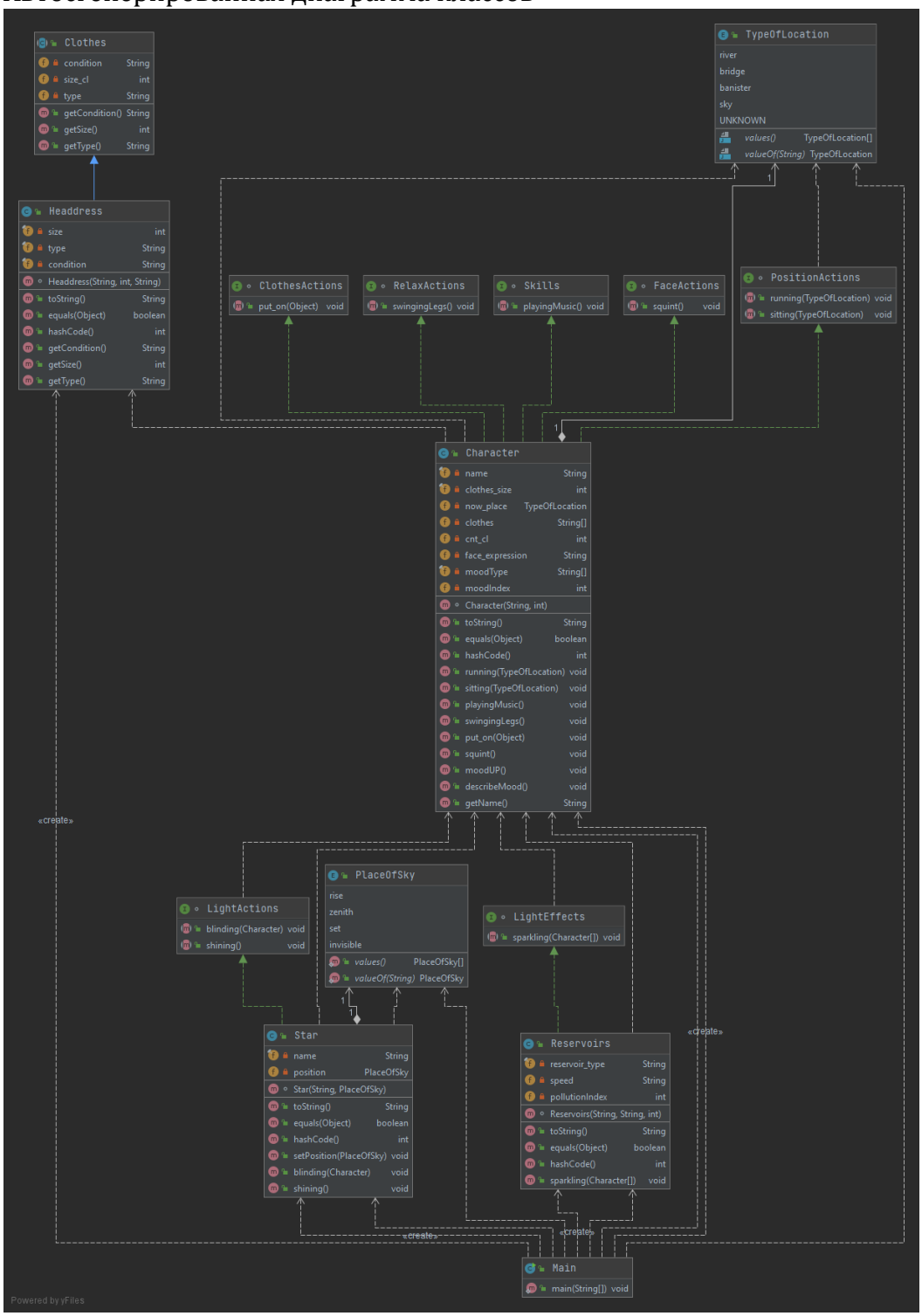
Snufkin is swinging legs, while sitting on the bridge

The running river is sparkling

Mummy-troll is in a free mood

Snufkin is in a carefree mood

5. Автосгенерированная диаграмма классов



6. Выводы.
В процессе выполнения лабораторной были получены базовые навыки и знания использования объектно – ориентированного программирования в языке Java.