Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа по программированию №3-4

Вариант №31090

Выполнил:
Студент группы P3106
Некрутенко Максим Владимирович
Проверил:
Вербовой. А. А.,
Преподаватель-практик ФПиКТ

Санкт-Петербург, 2024

Оглавление

Задание

Этапы выполнения работы:

1. Получить вариант
2. Нарисовать UML-диаграмму, представляющую классы и интерфейсы объектной модели и их взаимосвязи;
3. Придумать сценарий, содержащий действия персонажей, аналогичные приведенным в исходном тексте;
4. Согласовать диаграмму классов и сценарий с преподавателем;
5. Написать программу на языке Java, реализующую разработанные объектную модель и сценарий взаимодействия и изменения состояния объектов. При запуске программа должна проигрывать сценарий и выводить в стандартный вывод текст, отражающий изменение состояния объектов, приблизительно напоминающий исходный текст полученного отрывка.
6. Продемонстрировать выполнение программы на сервере `helios`.
7. Ответить на контрольные вопросы и выполнить дополнительное задание.

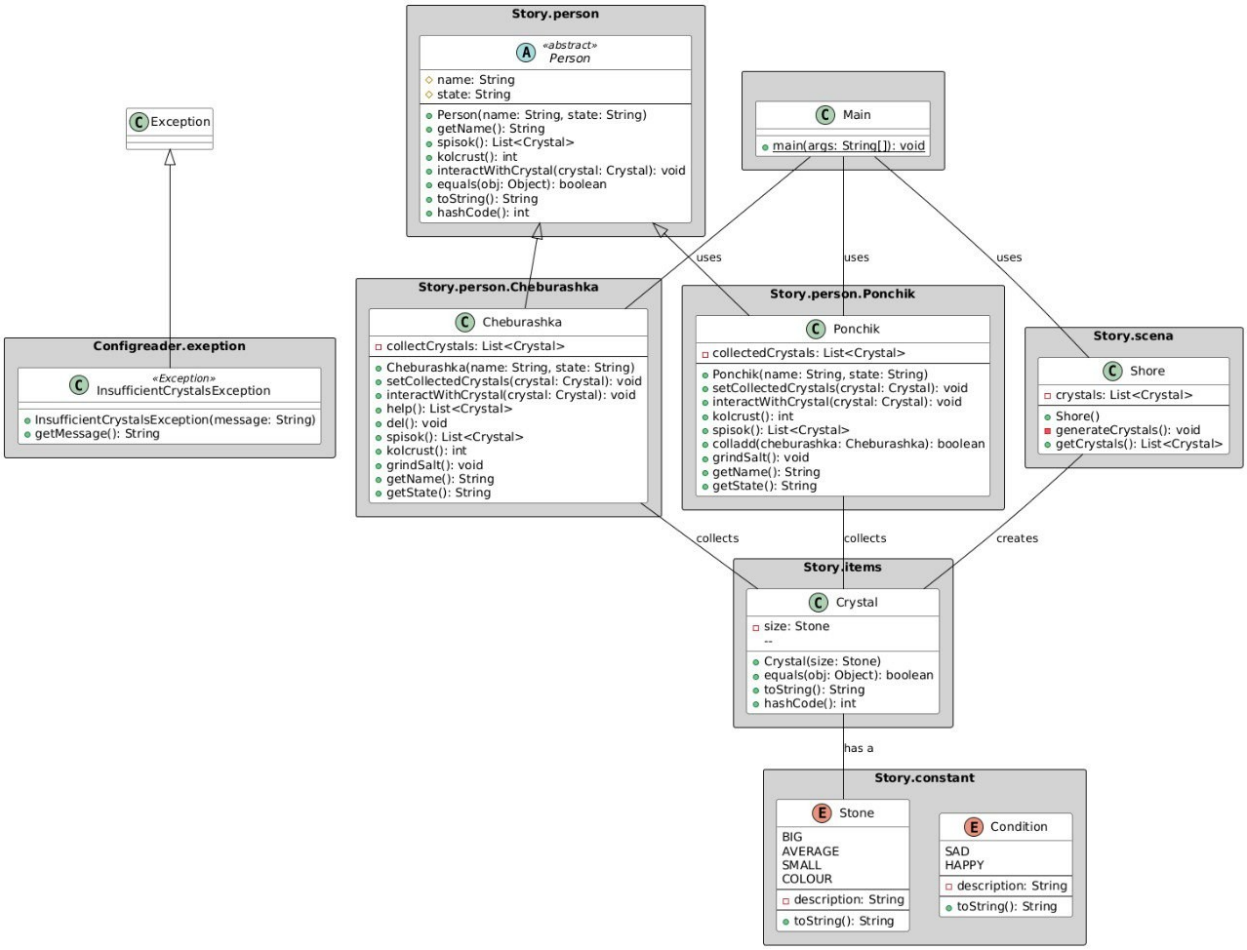
Текст, выводящийся в результате выполнения программы не обязан дословно повторять текст, полученный в исходном задании. Также не обязательно реализовывать грамматическое согласование форм и падежей слов выводимого текста.

Стоит отметить, что цель разработки объектной модели состоит не в выводе текста, а в эмуляции объектов предметной области, а именно их состояния (поля) и поведения (методы). Методы в разработанных классах должны изменять состояние объектов, а выводимый текст должен являться побочным эффектом, отражающим эти изменения.

Требования к объектной модели, сценарию и программе:

1. В модели должны быть представлены основные персонажи и предметы, описанные в исходном тексте. Они должны иметь необходимые атрибуты и характеристики (состояние) и уметь выполнять свойственные им действия (поведение), а также должны образовывать корректную иерархию наследования классов.
2. Объектная модель должна реализовывать основные принципе ООП - инкапсуляцию, наследование и полиморфизм. Модель должна соответствовать принципам SOLID, быть расширяемой без глобального изменения структуры модели.
3. Сценарий должен быть вариативным, то есть при изменении начальных характеристик персонажей, предметов или окружающей среды, их действия могут изменяться и отклоняться от базового сценария, приведенного в исходном тексте. Кроме того, сценарий должен поддерживать элементы случайности (при генерации персонажей, при задании исходного состояния, при выполнении методов).
4. Объектная модель должна содержать как минимум один корректно использованный элемент каждого типа из списка:
    - абстрактный класс как минимум с одним абстрактным методом;
    - интерфейс;
    - перечисление (enum);
    - запись (record);
    - массив или ArrayList для хранения однотипных объектов;
    - проверяемое исключение.
5. В созданных классах основных персонажей и предметов должны быть корректно переопределены методы `equals()`, `hashCode()` и `toString()`. Для классов-исключений необходимо переопределить метод `getMessage()`.
6. Созданные в программе классы-исключения должны быть использованы и обработаны. Кроме того, должно быть использовано и обработано хотя бы одно unchecked исключение (можно свое, можно из стандартной библиотеки).
7. При необходимости можно добавить внутренние, локальные и анонимные классы.

Диаграмма классов объектной модели



Исходный код программы

```
import Configreader.exeption.InsufficientCrystalsException;
import Story.constant.Stone;
import Story.items.Crystal;
import Story.person.Cheburashka.Cheburashka;
import Story.person.Ponchik.Ponchik;
import Story.scena.Shore;


public class Main{
    public static void main(String[] args){
        Shore shore = new Shore();
        Ponchik ponchik = new Ponchik("Mak", "");
        Cheburashka cheburashka = new Cheburashka("Nek", "");
        System.out.println(Stone.COLOUR);

        for (Crystal crystal : shore.getCrystals()) {
            cheburashka.setCollectedCrystals(crystal);
            cheburashka.interactWithCrystal(crystal);
        }

        System.out.println(cheburashka.getName() + " " + cheburashka.spisok());

        for (Crystal crystal : shore.getCrystals()) {
            ponchik.setCollectedCrystals(crystal);
            ponchik.interactWithCrystal(crystal);
        }

        System.out.println(ponchik.getName() + " " + ponchik.spisok());

        if (ponchik.kolcrust() < 4 && cheburashka.kolcrust() > 4){
            if ((cheburashka.kolcrust() - (4 - ponchik.kolcrust())) >= 4){
                System.out.println(cheburashka.getName() + " " + cheburashka.spisok());
                System.out.println(cheburashka.getName() + " помогает " + ponchik.getName());
                ponchik.colladd(cheburashka);

                System.out.println(ponchik.getName() + " " + ponchik.spisok());

                cheburashka.del();

                System.out.println(cheburashka.getName() + " " + cheburashka.spisok());

                try {
                    ponchik.grindSalt();
                } catch (InsufficientCrystalsException e) {
                    System.out.println(ponchik.getName() + " ошибка: " + e.getMessage());
                }
            }
            else {
                try {
                    ponchik.grindSalt();
                } catch (InsufficientCrystalsException e) {
                    System.out.println(ponchik.getName() + " ошибка: " + e.getMessage());
```

```
            }
        }
    }
    else {
        try {
            ponchik.grindSalt();
        } catch (InsufficientCrystalsException e) {
            System.out.println(ponchik.getName() + " ошибка: " + e.getMessage());
        }
    }

    try {
        cheburashka.grindSalt();
    } catch (InsufficientCrystalsException e) {
        System.out.println(cheburashka.getName() + " ошибка: " + e.getMessage());
    }
  }
}
```

```
package Configreader.exeption;

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

  @Override
  public String getMessage() {
    return super.getMessage();
  }
}
```

```
package Story.constant;

public enum Condition {
  SAD("грустный"),
  HAPPY("счастливый");


  private final String description;
  Condition(String description) {
    this.description = description;
  }

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

```
    }

}
```

```
package Story.constant;

public enum Stone {
    BIG("большой"),
    AVERAGE("средний"),
    SMALL("маленький"),

    COLOUR("Узкий пологий берег, тянувшийся полосой вдоль моря, \n" +
            "был ограничен с противоположной стороны обрывистыми, словно подмытыми водой, холмами,
которые поросли сверху зеленой травкой и мелким кустарником. \n" +
            "Сам берег был покрыт ослепительно белым песочком и какими-то прозрачными камнями,
напоминавшими обломки ледяных или стеклянных глыб.");

    private final String description;

    Stone(String description){
        this.description = description;
    }

    @Override
    public String toString(){
        return description;
    }
}
```

```
package Story.items;


public record Crystal(Story.constant.Stone size){
    @Override
    public boolean equals(Object obj){
        if (!(obj instanceof Crystal)) {
            return false;
        }
        Crystal crystal = (Crystal) obj;
        return this.size.equals(crystal.size);
    }

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

```
    @Override
    public int hashCode() {
        return size.hashCode();
    }
}
```

```
package Story.person.Cheburashka;

import Configreader.exeption.InsufficientCrystalsException;
import Story.constant.Condition;
import Story.items.Crystal;
import Story.person.Person;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;


public class Cheburashka extends Person {
    private List<Crystal> collectCrystals;


    public Cheburashka(String name, String state){
        super(name, state);
        this.collectCrystals = new ArrayList<>();
    }

    public void setCollectedCrystals(Crystal crystal){
        System.out.println(name + " собрал " + crystal.size() + " кристал");
    }

    public void interactWithCrystal(Crystal crystal){
        System.out.println(name + " взаимодействует с " + crystal.size() + " кристал");
        if (new Random().nextBoolean()){
            collectCrystals.add(crystal);
            System.out.println(name + " облизал " + crystal.size() + " кристал" + " и нашел его вкусным!");
            System.out.println(name +  " " + Condition.HAPPY);
        }

        else {
            System.out.println(name + " не понравился " + crystal.size() + " кристал");
            System.out.println(name + " " + Condition.SAD);
        }
    }

    // метод который дает лишние кристаллы, возвращает список
    public List<Crystal> help(){
        return collectCrystals.subList(0, kolcrust() - 4);
    }

    // удаляет кристалл из списка
    public void del(){
```

```
        collectCrystals.subList(0, kolcrust() - 4).clear();
    }

    // выводит список кристаллов
    public List<Crystal> spisok(){
        return collectCrystals;
    }

    // выводит кол-во кристаллов
    public int kolcrust(){
        return collectCrystals.size();
    }

    public void grindSalt() throws InsufficientCrystalsException {
        if (collectCrystals.size() < 4) {
            throw new InsufficientCrystalsException("Недостаточно кристаллов для толчения соли.");
        }
        else {
            System.out.println(name + " измельчает соль из собранных кристаллов.");
            collectCrystals.clear();
        }
    }

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

```
package Story.person.Ponchik;

import Configreader.exeption.InsufficientCrystalsException;
import Story.constant.Condition;
import Story.constant.Stone;
import Story.items.Crystal;
import Story.person.Cheburashka.Cheburashka;
import Story.person.Person;

import java.util.ArrayList;
import java.util.List;


public class Ponchik extends Person {
    private List<Crystal> collectedCrystals;


    public Ponchik(String name, String state){
        super(name, state);
        this.collectedCrystals = new ArrayList<>();
    }

    public void setCollectedCrystals(Crystal crystal){
```

```
        System.out.println(name + " собрал " + crystal.size() + " кристал");
    }

    public void interactWithCrystal(Crystal crystal){
        System.out.println(name + " взаимодействует с " + crystal.size() + " кристал");
        if (crystal.size() == Stone.BIG) {
            collectedCrystals.add(crystal);
            System.out.println(name + " облизал " + crystal.size() + " кристал" + " и нашел его вкусным!");
            System.out.println(name + " " + Condition.HAPPY);
        }
        else {
            System.out.println(name + " не понравился " + crystal.size() + " кристал");
            System.out.println(name + " " + Condition.SAD);
        }
    }

    // выводит кол-во кристаллов
    public int kolcrust(){
        return collectedCrystals.size();
    }

    // выводит список кристаллов
    public List<Crystal> spisok(){
        return collectedCrystals;
    }

    // с помощью функции help в collectedCrystals добавляются недостающие кристаллы
    public boolean colladd(Cheburashka cheburashka){
        List<Crystal> pomosh = cheburashka.help();
        return collectedCrystals.addAll(pomosh);
    }

    public void grindSalt() throws InsufficientCrystalsException {
        if (collectedCrystals.size() < 4) {
            throw new InsufficientCrystalsException("Недостаточно кристаллов для толчения соли.");
        }
        else {
            System.out.println(name + " измельчает соль из собранных кристаллов.");
            collectedCrystals.clear();
        }
    }

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

```
package Story.person;


import Story.items.Crystal;
```

```java
import java.util.List;

public abstract class Person {
    protected String name;
    protected String state;

    public Person(String name, String state){
        this.name = name;
        this.state = state;
    }

    abstract public String getName();

    abstract public List<Crystal> spisok();

    abstract public int kolcrust();

    public abstract void interactWithCrystal(Crystal crystal);

    @Override
    public boolean equals(Object obj){
        if (!(obj instanceof Person)) {
            return false;
        }
        Person person = (Person) obj;
        return this.name.equals(person.name);
    }

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

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

```java
package Story.scena;

import Story.constant.Stone;
import Story.items.Crystal;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Shore {
    private List<Crystal> crystals;

    public Shore(){
```

```
        this.crystals = new ArrayList<>();
        generateCrystals();
    }
    private void generateCrystals() {
        Stone[] sizes = {Stone.BIG, Stone.AVERAGE, Stone.SMALL};
        Random random = new Random();
        for (int i = 0; i < 9; i++){
            crystals.add(new Crystal(sizes[random.nextInt(sizes.length)]));
        }
    }

    public List<Crystal> getCrystals() {
        return crystals;
    }
}
```

Результат работы программы

Узкий пологий берег, тянувшийся полосой вдоль моря,
был ограничен с противоположной стороны обрывистыми, словно подмытыми водой, холмами, которые поросли сверху зеленой травкой и мелким кустарником.
Сам берег был покрыт ослепительно белым песочком и какими-то прозрачными камнями, напоминавшими обломки ледяных или стеклянных глыб.
Nek собрал средний кристал
Nek взаимодействует с средний кристал
Nek облизал средний кристал и нашел его вкусным!
Nek счастливый
Nek собрал средний кристал
Nek взаимодействует с средний кристал
Nek не понравился средний кристал
Nek грустный
Nek собрал большой кристал
Nek взаимодействует с большой кристал
Nek облизал большой кристал и нашел его вкусным!
Nek счастливый
Nek собрал средний кристал
Nek взаимодействует с средний кристал
Nek облизал средний кристал и нашел его вкусным!
Nek счастливый
Nek собрал средний кристал
Nek взаимодействует с средний кристал
Nek не понравился средний кристал
Nek грустный
Nek собрал большой кристал
Nek взаимодействует с большой кристал
Nek облизал большой кристал и нашел его вкусным!
Nek счастливый
Nek собрал маленький кристал
Nek взаимодействует с маленький кристал
Nek облизал маленький кристал и нашел его вкусным!
Nek счастливый
Nek собрал большой кристал
Nek взаимодействует с большой кристал
Nek не понравился большой кристал
Nek грустный
Nek собрал средний кристал
Nek взаимодействует с средний кристал
Nek облизал средний кристал и нашел его вкусным!
Nek счастливый
Nek [AVERAGE, BIG, AVERAGE, BIG, SMALL, AVERAGE]
Mak собрал средний кристал

Mak взаимодействует с средний кристал
Mak не понравился средний кристал
Mak грустный
Mak собрал средний кристал
Mak взаимодействует с средний кристал
Mak не понравился средний кристал
Mak грустный
Mak собрал большой кристал
Mak взаимодействует с большой кристал
Mak облизал большой кристал и нашел его вкусным!
Mak счастливый
Mak собрал средний кристал
Mak взаимодействует с средний кристал
Mak не понравился средний кристал
Mak грустный
Mak собрал средний кристал
Mak взаимодействует с средний кристал
Mak не понравился средний кристал
Mak грустный
Mak собрал большой кристал
Mak взаимодействует с большой кристал
Mak облизал большой кристал и нашел его вкусным!
Mak счастливый
Mak собрал маленький кристал
Mak взаимодействует с маленький кристал
Mak не понравился маленький кристал
Mak грустный
Mak собрал большой кристал
Mak взаимодействует с большой кристал
Mak облизал большой кристал и нашел его вкусным!
Mak счастливый
Mak собрал средний кристал
Mak взаимодействует с средний кристал
Mak не понравился средний кристал
Mak грустный
Mak [BIG, BIG, BIG]
Nek [AVERAGE, BIG, AVERAGE, BIG, SMALL, AVERAGE]
Nek помогает Mak
Mak [BIG, BIG, BIG, AVERAGE, BIG]
Nek [AVERAGE, BIG, SMALL, AVERAGE]
Mak измельчает соль из собранных кристаллов.
Nek измельчает соль из собранных кристаллов.

Вывод

При выполнении лабораторной работы я познакомился с принципами SOLID. Также я реализовывал собственные классы. интерфейсы, enum, record и абстрактные классы и их взаимодействие друг с другом. Данная работа улучшила мои навыки в программировании.