Федеральное государственное автономное образовательное учреждение

высшего образования «Национальный исследовательский университет

ИТМО»

*Факультет программной инженерии и компьютерной техники*

*Направление подготовки: 09.03.04 – Нейротехнологии и программирование*

*Дисциплина «Программирование»*

**Отчет**

**По лабораторной работе №4**

Выполнил:

Мухин Никита Денисович

Группа: Р3123

Преподаватель:

Данилов Павел Юрьевич

Г. Санкт-Петербург, 2023 г.

**Оглавление**

[Задание 2](#_Toc114643031)

UML [диаграмма классов 3](#_Toc114643032)

[Код программы:](#_Toc114643033) 4

[Результат выполнения программы: 7](#_Toc114643034)

[Вывод](#_Toc114643035) 8

[Список литературы.](#_Toc114643036) 9

# Задание

Доработать программу из [лабораторной работы #3](https://se.ifmo.ru/disciplines/programming#lab3), обновив реализацию объектной модели в соответствии с новой версией описания предметной области.

Введите вариант:

Чтобы увидеть описание объектной модели, введите свой номер варианта.

**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:**

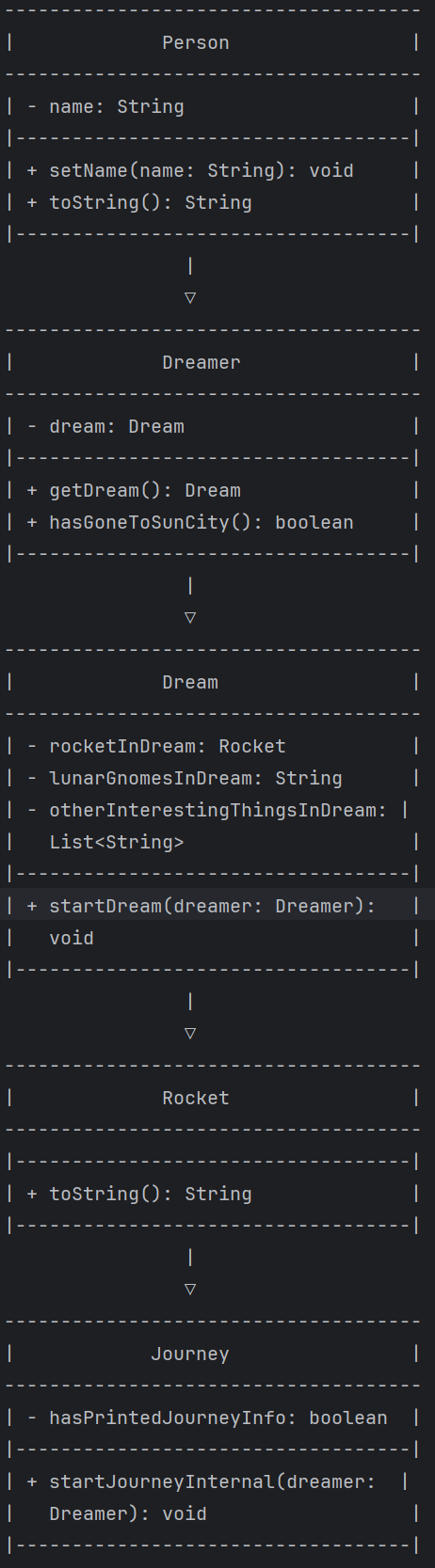
1. Доработать объектную модель приложения.
2. Перерисовать диаграмму классов в соответствии с внесёнными в модель изменениями.
3. Согласовать с преподавателем изменения, внесённые в модель.
4. Модифицировать программу в соответствии с внесёнными в модель изменениями.

**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.

# 

# UML-диаграмма классов:



# Код программы:

#### **Классы:**

// Абстрактный класс Person  
public abstract class Person {  
 protected String name;  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
}

public class Dreamer {  
 private String name;  
 private Dream dream;  
 private boolean hasGoneToSunCity;  
 private boolean journeyStarted;  
  
 public Dreamer(String name) {  
 this.name = name;  
 this.hasGoneToSunCity = false;  
 this.journeyStarted = false;  
 }  
  
 public void setDream(Dream dream) {  
 this.dream = dream;  
 }  
  
 public boolean hasGoneToSunCity() {  
 // Логика определения, уехал ли Знайка в Солнечный город  
 return false;  
 }  
  
 public void goToSunCity() {  
 this.hasGoneToSunCity = true;  
 }  
  
 public Dream getDream() {  
 return dream;  
 }  
  
 public boolean isJourneyStarted() {  
 return journeyStarted;  
 }  
  
 public void startJourney() {  
 this.journeyStarted = true;  
 }  
  
 @Override  
 public String toString() {  
 return "Знайка";  
 }  
}

import java.util.List;  
  
// Класс Dream, представляющий сон  
class Dream {  
 private Rocket rocketInDream;  
 private String lunarGnomesInDream;  
 private List<String> otherInterestingThingsInDream;  
 private boolean hasPrintedDreamInfo = false;  
  
 public Dream(Rocket rocketInDream, String lunarGnomesInDream, List<String> otherInterestingThingsInDream) {  
 this.rocketInDream = rocketInDream;  
 this.lunarGnomesInDream = lunarGnomesInDream;  
 this.otherInterestingThingsInDream = otherInterestingThingsInDream;  
 }  
  
 public void startDream(Dreamer dreamer) {  
 if (!hasPrintedDreamInfo) {  
 System.*out*.println("Размечтавшись, " + dreamer + " не заметил, как погрузился в сон.");  
 System.*out*.println("И во сне " + this);  
 hasPrintedDreamInfo = true;  
 }  
 }  
  
 @Override  
 public String toString() {  
 return "ему снилась космическая ракета, и Луна, и лунные коротышки, и еще много разных интересных вещей.";  
 }  
}

// Класс Rocket, представляющий космическую ракету  
class Rocket {  
 public String getText() {  
 String text = "";  
  
 text += "Ведь сейчас, чтоб разогнать ракету до нужной скорости, приходится брать огромнейший запас топлива;\n" +  
 "если же ракета не будет ничего весить, то топлива понадобится совсем немного,\n" +  
 "и вместо запасов топлива можно взять побольше пассажиров и побольше пищи для них.\n";  
  
 int randomNumber = new Random().nextInt(5);  
  
 switch (randomNumber) {  
 case 0:  
 text += "В таком случае можно будет летать дальше и быстрее.";  
 break;  
 case 1:  
 text += "В таком случае можно будет перевозить больше людей и грузов.";  
 break;  
 case 2:  
 text += "В таком случае можно будет проводить более длительные космические экспедиции.";  
 break;  
 case 3:  
 text += "В таком случае можно будет осуществлять более частые запуски ракет.";  
 break;  
 case 4:  
 text += "В таком случае можно будет колонизировать другие планеты.";  
 break;  
 }  
 return text;  
 }  
}

class Journey {  
 private boolean hasStarted;  
 private boolean hasPrintedJourneyInfo = false;  
  
 public Journey() {  
 this.hasStarted = false;  
 this.hasPrintedJourneyInfo = false;  
 }  
  
 public void startJourney(Dreamer dreamer) {  
 startJourneyInternal(dreamer);  
 }  
  
 public void startJourneyInternal(Dreamer dreamer) {  
 if (!hasPrintedJourneyInfo) {  
 System.*out*.println("Размечтавшись, " + dreamer + " не заметил, как погрузился в сон. И во сне " + dreamer.getDream());  
 hasPrintedJourneyInfo = true;  
  
 if (!dreamer.hasGoneToSunCity()) {  
 System.*out*.println("А наутро " + dreamer + " исчез. К завтраку он не явился, " +  
 "а когда коротышки пришли к нему в комнату, они увидели на столе записку, " +  
 "в которой было всего три слова: \"В Солнечный город\", и подпись: \"" + dreamer + "\". \n" +  
 "Прочитав записку, коротышки сразу поняли, что " + dreamer + " уехал в Солнечный город.");  
 }  
 }  
 }  
}

import java.util.Random;  
  
public class Moon {  
  
 // Поле для хранения объекта класса Random  
 private Random random = new Random();  
  
 // Конструктор класса  
 public Moon() {  
 }  
  
 // Метод для получения текста о Луне  
 public String getText() {  
  
  
 String text = "Вот когда можно будет совершить длительную экспедицию на " + getRandomPlanet() + ".\n";  
 text += "проникнуть в её недра и, может быть, даже познакомиться с " + getRandomCreature() + ".";  
  
 return text;  
 }  
  
 // Метод для получения случайного существа  
 private String getRandomCreature() {  
  
 // Создаем массив существ  
 String[] creatures = {"лунными коротышками", "марсианами", "инопланетянами", "рептилоидами", "пришельцами", "чужими"};  
  
 // Получаем случайное существо  
 return creatures[random.nextInt(6)];  
 }  
  
 // Метод для получения случайной планеты  
 private String getRandomPlanet() {  
  
 // Возвращаем случайную планету из класса Planets  
 return Planets.*getRandomPlanet*();  
 }  
  
 // Метод для проверки исключений  
 public void checkExceptions() {  
  
 // Проверяем checked исключение  
 try {  
 throw new MyCheckedException("Это checked исключение");  
 } catch (MyCheckedException e) {  
 // Обрабатываем checked исключение  
 System.*out*.println("Обработка checked исключения: " + e.getMessage());  
 }  
  
 // Проверяем unchecked исключение  
 try {  
 throw new MyUncheckedException("Это unchecked исключение");  
 } catch (MyUncheckedException e) {  
 // Обрабатываем unchecked исключение  
 System.*out*.println("Обработка unchecked исключения: " + e.getMessage());  
 }  
 }  
  
 // Метод для вывода текста с вариацией  
 public void printTextWithVariation() {  
  
 int randomNumber = random.nextInt(3);  
  
 // В зависимости от случайного числа выводим один из вариантов текста  
 switch (randomNumber) {  
 case 0:  
 System.*out*.println("А еще можно будет построить базу.");  
 break;  
 case 1:  
 System.*out*.println("Или начать добывать ресурсы, которые могут быть очень ценными.");  
 break;  
 case 2:  
 System.*out*.println("А может быть, даже создать целую колонию.");  
 break;  
 }  
 }  
  
 // Статический класс для получения случайной планеты  
 static class Planets {  
  
 // Поле для хранения объекта класса Random  
 private static Random *random* = new Random();  
  
 // Массив планет  
 public static final String[] *PLANETS* = {"Луну", "Меркурий", "Марс", "Европу, спутник Юпитера", "Планету Х"};  
  
 // Метод для получения случайной планеты  
 public static String getRandomPlanet() {  
  
 // Получаем случайное число от 0 до 4  
 int randomNumber = *random*.nextInt(5);  
  
 // Возвращаем планету с соответствующим индексом  
 return *PLANETS*[randomNumber];  
 }  
 }  
}  
  
// Класс для проверки checked исключений  
class MyCheckedException extends Exception {  
  
 // Конструктор класса  
 public MyCheckedException(String message) {  
 super(message);  
 }  
}  
  
// Класс для проверки unchecked исключений  
class MyUncheckedException extends RuntimeException {  
  
 // Конструктор класса  
 public MyUncheckedException(String message) {  
 super(message);  
 }  
}

import java.util.Random;  
import java.lang.reflect.InvocationTargetException;  
  
public class Weightless {  
  
 private String text = "В эту ночь Знайка долго не мог заснуть:\n" +  
 "все думал, какую пользу может принести состояние невесомости.";  
  
 public void printText() {  
 System.*out*.println(text);  
 }  
  
 public double randomValue;  
  
 public Weightless() {  
 // Генерация случайного числа для определения класса для вывода текста  
 randomValue = Math.*random*();  
 }  
  
 public static String getText() {  
// Определение класса для вывода текста  
 Class<?> clazz = null;  
 double randomValue = Math.*random*();  
  
 if (randomValue < 0.3) {  
 // Создаем экземпляр статического вложенного класса,  
 // который реализует TextProvider  
 clazz = StaticNestedTextProvider.class;  
 } else if (randomValue < 0.6) {  
 // Создаем экземпляр вложенного класса  
 clazz = NonStaticNestedClass.class;  
  
 } else {  
 // Возвращаем значение по умолчанию  
 return "Гравитации не существует, математика лже-наука.";  
 }  
 // Вывод текста  
 try {  
 return ((TextProvider) clazz.newInstance()).getText();  
 } catch (InstantiationException | IllegalAccessException e) {  
 // Обработка исключений  
 String text = "Гравитации не существует, математика лже-наука.";  
 // Возврат значения по умолчанию  
 return text;  
 }  
 }  
  
  
 // Интерфейс для получения текста  
 interface TextProvider {  
 String getText();  
 }  
  
 // Статический вложенный класс  
 static class StaticNestedTextProvider implements TextProvider {  
  
 public static String *text* = "Как хорошо, что ее можно ощутить не только в космосе.";  
  
 @Override  
 public String getText() {  
 return *text*;  
 }  
 }  
  
  
 // Анонимный класс  
 static class StaticNestedClass implements TextProvider {  
 public String getText() {  
 // Всегда возвращаем первое предложение  
 String firstText = "Невесомость – это огромная сила, если знать, как подступиться к ней, – размышлял он.\n";  
  
 // Генерация случайного числа от 0 до 4  
 int randomNumber = new Random().nextInt(5);  
  
 // Возврат соответствующего предложения  
 switch (randomNumber) {  
 case 0:  
 return firstText +  
 "– С помощью невесомости можно поднимать и передавать огромные тяжести.";  
 case 1:  
 return firstText +  
 "– А с большой силой приходит большая ответственность.";  
 case 2:  
 return firstText +  
 "– Она дает возможность бесконечного полета.";  
 case 3:  
 return firstText +  
 "– Она открывает путь к новым технологиям.";  
 case 4:  
 return firstText +  
 "За ней будущее человечества.";  
 default:  
 return null;  
 }  
  
 }  
 }  
  
 public class WeightlessException extends RuntimeException {  
 private String message;  
  
 public WeightlessException(String message) {  
 super(message);  
 this.message = message;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
 }  
  
 // Вложенный класс (нестатический)  
 public class NonStaticNestedClass implements TextProvider {  
  
 public NonStaticNestedClass() {  
 // В конструкторе не нужно бросать исключение NoSuchMethodException  
 }  
  
 public NonStaticNestedClass(Object obj) {  
 // В этом конструкторе мы сохраняем значение параметра obj в поле name  
 this.name = obj.toString();  
 }  
  
 @Override  
 public String getText() throws WeightlessException {  
 // В данном примере бросаем исключение класса WeightlessException  
 throw new WeightlessException("Исключение, связанное с состоянием невесомости");  
 }  
  
 private String name;  
 }  
}

**Main:**

import java.util.List;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 // Создаем объекты  
 Dreamer znayka = new Dreamer("Знайка");  
 Weightless weightless = new Weightless();  
 Rocket rocket = new Rocket();  
 Moon moon = new Moon();  
  
  
 weightless.printText();  
 System.*out*.println(new Weightless().*getText*());  
 System.*out*.println(rocket.getText());  
 System.*out*.println(moon.getText());  
 moon.printTextWithVariation();  
  
 // Создаем объекты классов `Rocket`, `String` и `List`.  
 Rocket rocketInDream = new Rocket();  
 String lunarGnomesInDream = "лунные коротышки";  
 List<String> otherInterestingThingsInDream = List.*of*("Еще много разных интересных вещей");  
  
 // Создаем объекты классов `Journey` и `Dream`.  
 Journey journey = new Journey(znayka);  
 Dream dream = new Dream(rocketInDream, lunarGnomesInDream, otherInterestingThingsInDream);  
  
  
  
 dream.startDream(znayka);  
 znayka.goToSunCity();  
  
 // Проверяем, был ли успешным полет Знайки.  
 // Если да, то запускаем путешествие.  
 if (znayka.isJourneySuccessful()) {  
 journey.startJourney();  
 } else {  
 // Бросаем исключение  
 }  
 }  
}

# Результат работы программы:

В эту ночь Знайка долго не мог заснуть:

все думал, какую пользу может принести состояние невесомости.

Как хорошо, что ее можно ощутить не только в космосе.

Ведь сейчас, чтоб разогнать ракету до нужной скорости, приходится брать огромнейший запас топлива;

если же ракета не будет ничего весить, то топлива понадобится совсем немного,

и вместо запасов топлива можно взять побольше пассажиров и побольше пищи для них.

В таком случае можно будет колонизировать другие планеты.

Вот когда можно будет совершить длительную экспедицию на Луну.

проникнуть в её недра и, может быть, даже познакомиться с пришельцами.

А может быть, даже создать целую колонию.

Размечтавшись, Знайка не заметил, как погрузился в сон.

И во сне ему снилось, что он встретил инопланетян.

А наутро Знайка исчез. К завтраку он не явился,

а когда коротышки пришли к нему в комнату, они увидели на столе записку,

в которой было всего три слова: "В Солнечный город", и подпись: "Знайка".

Знайка отправился в Солнечный город пешком.

# Вывод:

В ходе выполнения работы была доработана объектная модель приложения в соответствии с новой версией описания предметной области. В программу были добавлены два собственных класса исключений, а также обработка исключений этих классов. Кроме того, в программу были добавлены локальные, анонимные и вложенные классы (static и non-static).

# Список литературы.

1. https://se.ifmo.ru/courses/programming
2. <https://javarush.ru/groups/posts/1981-dlja-chego-v-java-nuzhnih-interfeysih>
3. <https://javarush.ru/groups/posts/591-vot-tak-final>
4. <https://www.youtube.com/watch?v=3Xo6zSBgdgk>
5. <https://javarush.ru/groups/posts/1985-raznica-mezhdu-abstraktnihmi-klassami-i-interfeysami>
6. <https://skillbox.ru/media/base/klassy-i-obekty-v-java/>
7. <https://www.jetbrains.com/help/idea/class-diagram.html#analyze_graph>
8. <https://metanit.com/java/tutorial/3.3.php>
9. JAVA from EPAM, 2-е издание, исправленное. И. Н. Блинов, В. С. Романчук
10. <https://javarush.ru/groups/posts/1967-otnoshenija-mezhdu-klassami-nasledovanie-kompozicija-i-agregirovanie->
11. <https://javarush.ru/groups/posts/599-inicializacija-poley>
12. https://javarush.ru/groups/posts/2347-klass-string-v-java