НИУ ИТМО

Факультет программной инженерии и компьютерной техники

Отчет по Лабораторной работе №4    
по курсу “Программирование”  
Вариант № 1057894.6

Выполнила:

Студентка группы P3110

Бегинина Анастасия Алексеевна   
Преподаватель:

Сорокин Роман Борисович

Санкт-Петербург

2021

Оглавление

[Текст задания 3](#_Toc90406351)

[Диаграмма классов 4](#_Toc90406352)

[Исходный код 4](#_Toc90406353)

[Результат работы программы 5](#_Toc90406354)

[Main.java, anonymous inner classes 5](#_Toc90406355)

[BallIsNotTiedException.java 6](#_Toc90406356)

[SquadIsBusyException.java 7](#_Toc90406357)

[Net.java, static nested class, inner class 7](#_Toc90406358)

[Ball.java, local inner class 7](#_Toc90406359)

[Выводы по работе 8](#_Toc90406360)

# Текст задания

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

*Всем нашлась возле шара работа, а Незнайка только ходил вокруг да посвистывал. Он старался держаться от шара подальше, поглядывал на него издали и приговаривал:*

*— Лопнет пузырь! Вот сейчас, сейчас лопнет! Уф!*

*Но шар не лопался, а с каждой минутой становился всё больше и больше. Скоро он раздулся такой большой, что малышам приходилось взбираться на ореховый куст, который рос посреди двора, чтобы обмазывать шар сверху и по бокам. Работа по надуванию шара продолжалась два дня и прекратилась тогда, когда шар стал величиной с дом. После этого Знайка завязал верёвочкой резиновую трубку, которая была снизу, чтобы из шара не выходил воздух, и сказал:*

*— Теперь шар будет сохнуть, а мы с вами примемся за другую работу.*

*Он привязал шар верёвкой к ореховому кусту, чтобы не унесло ветром, после чего поделил малышей на два отряда. Одному отряду он велел собирать шелковичные коконы, чтобы размотать их и наделать шёлковых нитей. Из этих нитей он велел им сплести огромную сетку. Другому отряду Знайка велел сделать большую корзину из тонкой берёзовой коры. Пока Знайка со своими товарищами занимался этой работой, все жители Цветочного города приходили и смотрели на огромнейший шар, который был привязан к ореховому кусту. Каждому хотелось потрогать шар руками, а некоторые даже пытались его приподнять.*

*— Шар лёгкий, — говорили они, — его свободно можно поднять кверху одной рукой.*

*— Лёгкий-то он лёгкий, но, по-моему, он не полетит, — сказал малыш, по имени Топик.*

*— Почему не полетит? — спросили остальные.*

*— Как же он полетит? Если бы он мог летать, то взвивался бы кверху, а он просто лежит на земле. Значит, хоть он и лёгкий, а всё-таки тяжёлый, — ответил Топик.*

*Коротышки задумались.*

*— Гм! Гм! — говорили они. — Шар лёгкий, а всё-таки тяжёлый. Это верно. Как же он полетит?*

**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.

# Диаграмма классов

Graphical user interface

Description automatically generated

# Исходный код

Ссылка на Git-репозиторий: https://github.com/nastyabeggin/ITMO\_pe\_first\_term/tree/main/programming/lab4

# Результат работы программы

## Main.java, anonymous inner classes

public class Main {  
 public static void main(String[] args) {  
  
 Ball ball = new Ball();  
  
 Residents residents = new Residents("Все");  
 residents.come();  
 try {  
 residents.touch(ball);  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 System.*out*.println("Привязан ли шар? " + ball.is\_tied);  
 }  
 System.*out*.println(residents.getName() + residents.describe());  
  
 LittleBrainiac brainiac = new LittleBrainiac("Знайка");  
 Squad squad = new Squad("занятой отряд");  
 try {  
 brainiac.order(squad, OrderType.*РАЗМАТЫВАТЬ*);  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 System.*out*.println(squad.describe());  
 }  
 try {  
 brainiac.order(squad, OrderType.*СПЛЕСТИ*);  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 System.*out*.println(squad.describe());  
 }  
  
 System.*out*.println(brainiac.talk(brainiac.*phrases\_of\_brainiac*[0]));  
 System.*out*.println(residents.talk(residents.phrases\_of\_residents[0]));  
 Rope rope = new Rope();  
 Bush bush = new Bush();  
 ball.tied(rope, bush);  
 System.*out*.println(ball.describe());  
  
 Net net = new Net();  
 System.*out*.println(net.describe());  
 System.*out*.println(net.describe\_thread());  
 System.*out*.println(net.describe\_pods());  
  
 Residents residents\_1 = new Residents("Коротышки"){  
 @Override  
 public String describe(){  
  
 return this.getName() + " задумались";  
 }  
 };  
  
 Residents residents\_2 = new Residents("Жители Цветочного города"){  
 @Override  
 public String describe(){  
  
 return this.getName() + " приходили и смотрели на шар";  
 }  
 };  
 System.*out*.println(residents\_1.describe());  
 System.*out*.println(residents\_2.describe());

Нельзя шар трогать! Он не привязан и может улететь

Привязан ли шар? false

Все Жители хотели потрогать шар, но не трогали, потому что он не привязан и улетит

Отряд занят!

Отряд занимается работой Отряд разматывать

Знайка говорит: 'Лопнет пузырь! Вот сейчас, сейчас лопнет! Уф!'

Все говорит: 'Шар лёгкий, его свободно можно поднять кверху одной рукой.'

Шар привязан к ореховый куст с помощью Веревка привязана к ореховый куст. Шар не может быть унесен ветром Труба, к которой привязан Шар привязан к ореховый куст с помощью Веревка привязана к ореховый куст. Шар не может быть унесен ветром

огромная сетка из шелковые нити

шелковые нити

шелковые коконы

Коротышки задумались

Жители Цветочного города приходили и смотрели на шар

## BallIsNotTiedException.java

public class BallIsNotTiedException extends RuntimeException { // unchecked exception. Привязан ли шар  
 public BallIsNotTiedException(String message) {  
 super(message);  
 }  
}

## SquadIsBusyException.java

public class SquadIsBusyException extends Exception{ // checked exception. Занята ли команда  
 public SquadIsBusyException(String message) {  
 super(message);  
 }  
}

## Net.java, static nested class, inner class

import java.util.Arrays;  
  
public class Net implements Describable {  
 static String[] qualities = {"огромная"};  
 public static class Thread {  
 private static String[] qualities = {"шелковые"};  
 boolean are\_unwound = false;  
 public static String pods\_return = null;  
 public static String describe() {  
 return Arrays.toString(qualities).replace("[", "")  
 .replace("]", "") + " нити";  
 }  
 }  
 public class Pods { // коконы  
 String description\_pods = Arrays.toString(Thread.qualities).replace("[", "").replace("]", "") + " коконы";  
 }  
 public String describe() {  
 String for\_return = Arrays.toString(qualities).replace("[", "")  
 .replace("]", "") + " сетка" + " из " + Thread.describe();  
 return for\_return;  
 }  
 public String describe\_thread() {  
 return Thread.describe();  
 }  
 public String describe\_pods() {  
 Pods pods = new Pods();  
 return pods.description\_pods;  
 }  
}

## Ball.java, local inner class

public class Ball implements Describable { // шар  
 boolean is\_tied = false;  
 Rope rope\_tied\_with = null;  
 Bush bush\_tied\_to = null;  
 private String add\_return = null;  
 boolean can\_be\_carried = true; // может ли быть унесен ветром  
 String[] qualities = {"огромный"};  
  
  
 public void tied(Rope rope, Bush bush) {  
 is\_tied = true;  
 can\_be\_carried = false;  
 rope\_tied\_with = rope;  
 bush\_tied\_to = bush;  
 rope.tied(bush\_tied\_to);  
 bush.tied();  
 class Pipe implements Describable {  
 @Override  
 public String describe() {  
 String for\_return = Ball.this.describe();  
 return " Труба, к которой привязан " + for\_return;  
 }  
 }  
 Pipe pipe\_tied\_to = new Pipe();  
 add\_return = pipe\_tied\_to.describe();  
 }  
  
 @Override  
 public String describe() {  
 String for\_return = "";  
 String action = is\_tied ? "привязан" : "не привязан";  
 if (this.is\_tied) {  
 for\_return += "Шар " + action + bush\_tied\_to.describe() + " с помощью " + rope\_tied\_with.describe();  
 }  
 String action2 = can\_be\_carried ? "может быть унесен ветром" : "не может быть унесен ветром";  
 for\_return += ". Шар " + action2;  
 if (add\_return != null){  
 for\_return += add\_return;  
 }  
 return for\_return;  
  
 }  
}

## 

# Выводы по работе

Во время выполнения лабораторной работы я укрепила свои знания в ООП, получила опыт в использовании внутренних статических, анонимных и локальных классов, а также в создании собственных исключений.