Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационных технологий и анализа данных

Кафедра Автоматизированных систем

ОТЧЕТ

Вариант 16

к лабораторной работе по дисциплине:

|  |
| --- |
| Объектно–ориентированное программирование |
| Разработка классов с использованием механизмов наследования, |
| полиморфизма и инкапсуляции. |

наименование темы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы: | АСУб-19-1 |  |  |  | Гамаюнова А.С. |
|  | шифр группы |  | подпись |  | Фамилия ИО |
| Проверил: | доцент |  |  |  | Маланова Т.В, |
|  | должность |  | подпись |  | Фамилия ИО |

Иркутск 2020 г.

**Содержание**

[1 Постановка задачи 3](#_Toc26866821)

[2 Проектирование классов 4](#_Toc26866822)

[3 Описание структуры меню 7](#_Toc26866823)

[4 Таблица тестов 9](#_Toc26866824)

[5 Результаты тестирования 10](#_Toc26866825)

[6 Исходный код 13](#_Toc26866827)

[Список использованных источников 26](#_Toc26866828)

1 Постановка задачи

1. Изучение механизмов наследования, полиморфизма и инкапсуляции на практике.

2. Получение практических навыков описания иерархии классов и организации работы с объектами-членами классов-«групп».

3. Получение практических навыков использования абстрактных классов и переопределения унаследованных методов.

В соответствии с индивидуальным заданием описать иерархию классов, для каждого класса описать поля и соответствующие методы доступа к ним. В зависимости от задания некоторые из этих классов (как минимум один) являются абстрактными и служат для выделения общих данных и поведения для других классов. Абстрактный класс должен содержать как минимум один абстрактный метод, реализация которого у его наследников должна различаться. Помимо этого, в общую часть задания входит разработка класса, группирующего объекты описанных в соответствии с заданием классов. Для первого варианта индивидуального задания это может быть класс «кафедра». Для выполнения задания необходимо создать некоторое количество объектов, добавить их в группу используя предусмотренные методы класса-«группы» и для каждого из них вызвать унаследованный метод.

Завкафедрой, профессор, студент, персона

2 Проектирование классов

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 1 – Проектирование классов | | | |
| **№** | **Название** | **Назначение** | **Поля (атрибуты)** |
| 1 | public class Department | Класс-«группа» | protected static ArrayList<Person> *people* – список людей  private static BufferedReader Input – объект для чтения ввода |
| 2 | abstract public class Person | Абстрактный класс | protected name – имя и фамилия protected position – должность protected age - возраст protected id – индивидуальный номер protected infected – наличие заболевания |
| 3 | public class Student extends Person | Класс-наследник от Organization | -- -- |
| 4 | public class Professor extends Person | Класс-наследник от Organization | -- -- |
| 5 | public class HeadOfDepartment extends Person | Класс-наследник от Organization | -- -- |

**Department:**

Методы:

public static void main(String[] args) – начало программы и вызов меню

private static void menu() throws IOException – запуск меню

private static void statistics(ArrayList<Person> people) throws IOException – подсчет статистики

private static int all\_people(ArrayList<Person> people)– подсчет количества людей в списке

private static void informationByAge(ArrayList<Person> people) – вывод информации по возрасту

private static void add() throws IOException– добавление людей

private static void showList(ArrayList<Person> List) – вывод списка людей

public static void deletePerson(int id, ArrayList<Person> List) – удаление людей из списка

public static void searchByInfection(ArrayList<Person> List)– поиск зараженных

**Person:**

Методы:

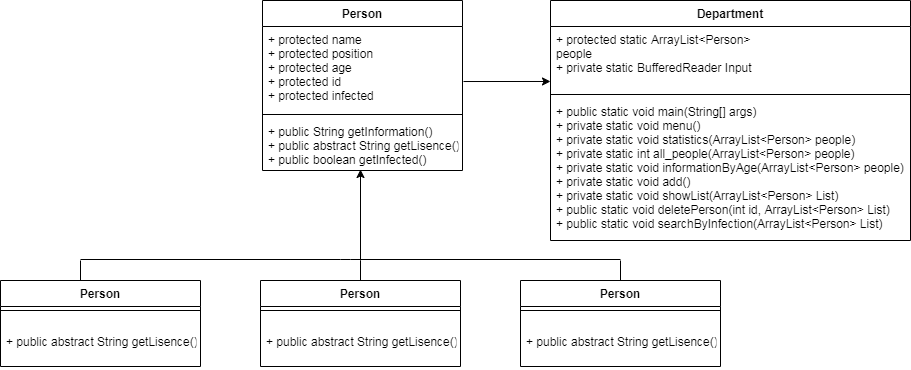
public String getInformation() - получить информацию о человеке

public abstract String getLisence() – получить информацию о том, может ли человек продолжать посещать вуз

**Student, Professor, HeadOfDepartment**

Методы эквивалентны методам класса Person

Рисунок 1 – Структура классов программы



3 Описание структуры меню

****

Рисунок 2 – Структура пользовательского меню

4 Таблица тестов

Таблица 2 – Таблица тестов.

|  |  |  |
| --- | --- | --- |
| **№** | **Словесное описание теста** | **Что должна сделать программа** |
| 1 | Выбор в меню пункт 1 | Переход к добавлению человека |
| 2 | Выбор в меню пункт 2 | Вывод списка |
| 3 | Выбор в меню пункт 3 | Переход к удалению человека |
| 4 | Выбор в меню пункт 4 | Вывод количества зараженных |
| 5 | Выбор в меню пункт 5 | Вывод статистики |
| 6 | Выбор в меню пункт 0 | Завершение программы |

5 Результаты тестирования

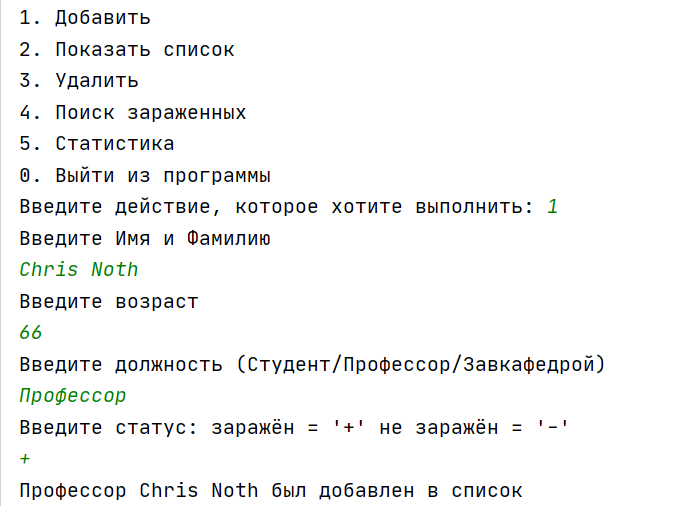


Рисунок 3 – Тест 1

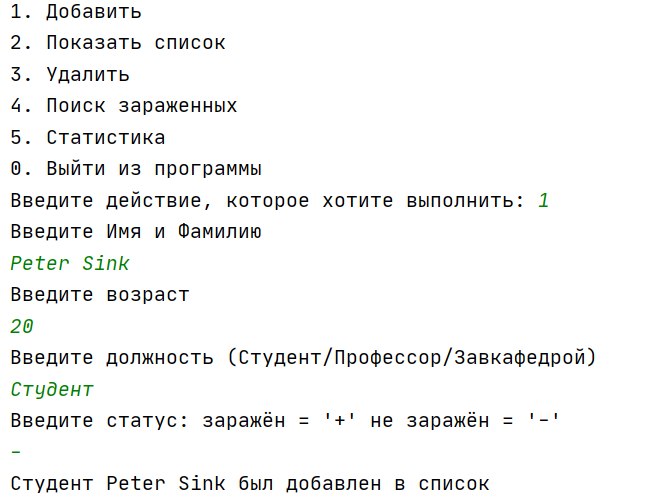


Рисунок 4 – Тест 1

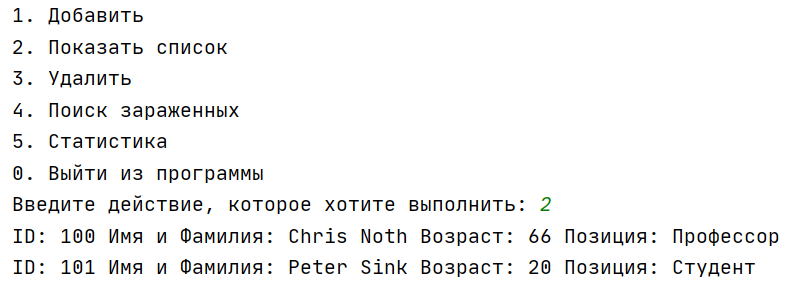


Рисунок 5 – Тест 2

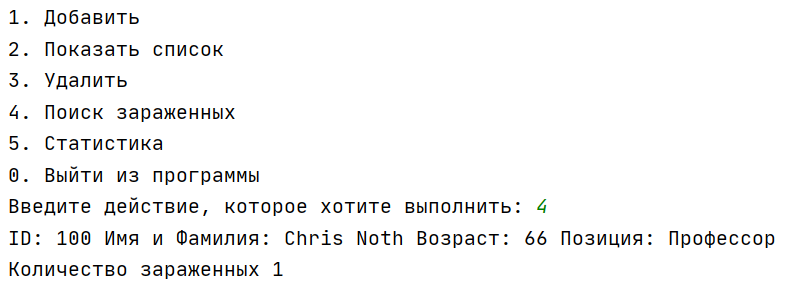


Рисунок 6 – Тест 4

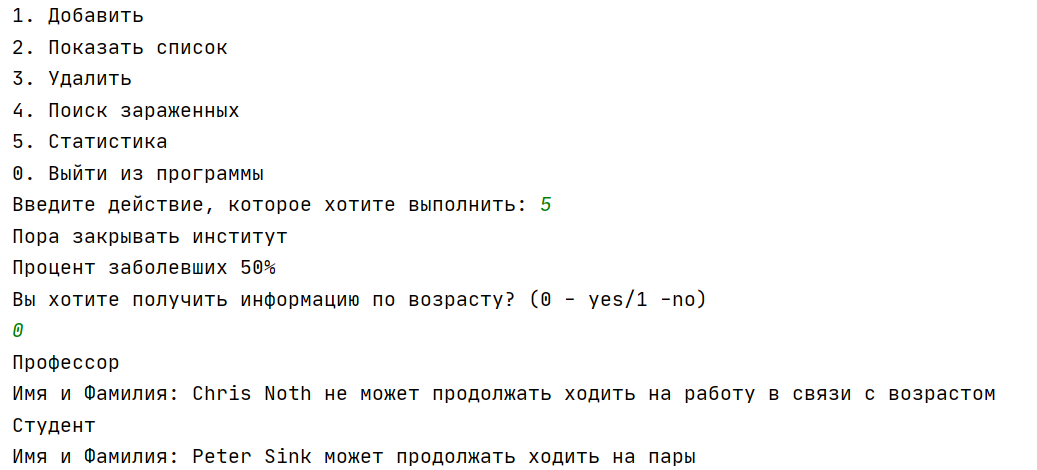


Рисунок 7 – Тест 5

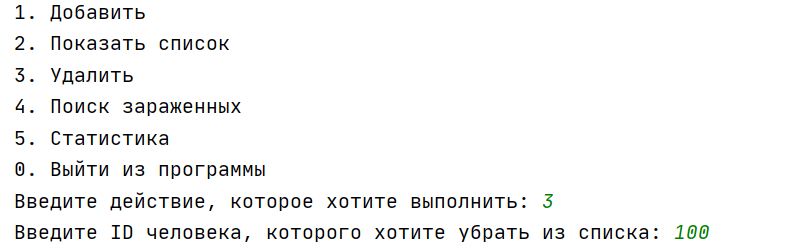


Рисунок 8 – Тест 3

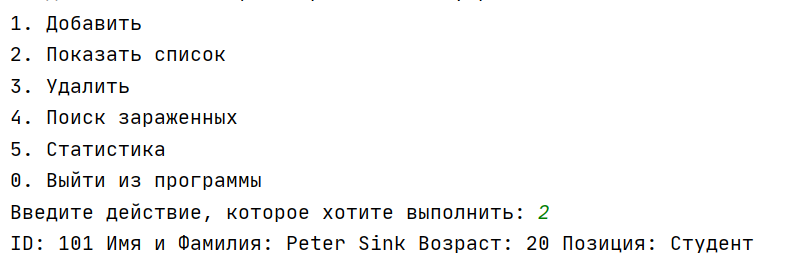


Рисунок 9 – Тест 2 ( вывод списка после удаления)

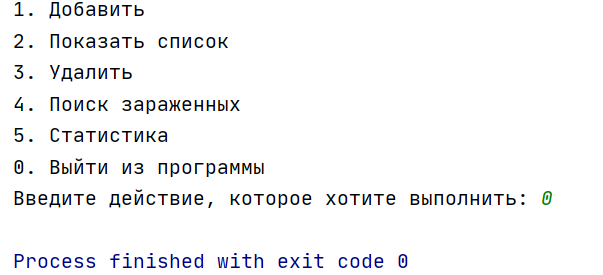


Рисунок 10 – Тест 7

6 Исходный код

Department:

import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
import java.io.BufferedReader;  
  
public class Department {  
 private static int *id* = 100;  
 protected static ArrayList<Person> *people* = new ArrayList<>();  
 private static BufferedReader *Input* = new BufferedReader(new InputStreamReader(System.*in*));  
  
 public static void main(String[] args) throws IOException {  
 *menu*();  
 }  
  
 private static void menu() throws IOException {  
 System.*out*.println("1. Добавить\n2. Показать список\n3. Удалить\n4. Поиск зараженных\n5. Статистика\n0. Выйти из программы");  
 System.*out*.print("Введите действие, которое хотите выполнить: ");  
 int Choice = Integer.*parseInt*(*Input*.readLine());  
 switch (Choice) {  
 case 1:  
 *add*();  
 *id* += 1;  
 break;  
 case 2: {  
 *showList*(*people*);  
 break;  
 }  
 case 3:  
 System.*out*.print("Введите ID человека, которого хотите убрать из списка: ");  
 int id = Integer.*parseInt*(*Input*.readLine());  
 *deletePerson*(id, *people*);  
 break;  
 case 4:  
 *searchByInfection*(*people*);  
 break;  
 case 5:  
 *statistics*(*people*);  
 System.*out*.println("Вы хотите получить информацию по возрасту? (0 - yes/1 -no)");  
 int choice = Integer.*parseInt*(*Input*.readLine());  
 if (choice == 0) {  
 *informationByAge*(*people*);  
 }  
 break;  
 case 0:  
 System.*exit*(0);  
 break;  
 }  
 *menu*();  
 }  
  
 private static void statistics(ArrayList<Person> people) throws IOException {  
 boolean people\_inf;  
 int all\_people = *all\_people*(people);  
 int amount\_people\_inf = 0;  
 for (Person a : people) {  
 people\_inf = a.getInfected();  
 if (people\_inf){  
 amount\_people\_inf += 1;  
 }  
 }  
 if ((amount\_people\_inf\*100/all\_people) >= (100 - (amount\_people\_inf\*100/all\_people))){  
 System.*out*.println("Пора закрывать институт");  
 System.*out*.println("Процент заболевших " + (amount\_people\_inf\*100/all\_people) + "%");  
 }  
 else {  
 System.*out*.println("Обстановка нормальная");  
 System.*out*.println("Процент заболевших " + (amount\_people\_inf\*100/all\_people) + "%");  
 }  
  
 }  
  
 private static int all\_people(ArrayList<Person> people){  
 return people.size();  
 }  
  
 private static void informationByAge(ArrayList<Person> people){  
 String str;  
 for (Person a : people) {  
 str = a.getLisence();  
 System.*out*.println(str);;  
 }  
 }  
  
 private static void add() throws IOException {  
 System.*out*.println("Введите Имя и Фамилию");  
 String name = *Input*.readLine();  
 System.*out*.println("Введите возраст");  
 int age = Integer.*parseInt*(*Input*.readLine());  
 System.*out*.println("Введите должность (Студент/Профессор/Завкафедрой)");  
 String position = *Input*.readLine();  
 System.*out*.println("Введите статус: заражён = '+' не заражён = '-'");  
 String infected = *Input*.readLine();  
 if (position.equals("Студент")) {  
 *people*.add(new Student(name, age, position, *id*, infected));  
 System.*out*.println(position + " " + name + " был добавлен в список");  
 } else if (position.equals("Профессор")) {  
 *people*.add(new Professor(name, age, position, *id*, infected));  
 System.*out*.println(position + " " + name + " был добавлен в список");  
 } else if (position.equals("Завкафедрой")) {  
 *people*.add(new HeadOfTheDepartment(name, age, position, *id*, infected));  
 System.*out*.println(position + " " + name + " был добавлен в список");  
 } else {  
 System.*out*.println("К сожалению, такой позиции нет, попробуйте снова");  
 *add*();  
 }  
 }  
  
 private static void showList(ArrayList<Person> List) {  
 for (Person a : List) {  
 System.*out*.println(a.getInformation());  
 }  
 }  
  
 public static void deletePerson(int id, ArrayList<Person> List) {  
 String stroka;  
 int valueOfID;  
 char[] dst = new char[3];  
 Person tmp = null;  
 for (Person a : List) {  
 stroka = a.getInformation();  
 stroka.getChars(4, 7, dst, 0);  
 valueOfID = Integer.*parseInt*(String.*valueOf*(dst));  
 if (valueOfID == id) {  
 tmp = a;  
 }  
 }  
 *people*.remove(tmp);  
 }  
  
 public static void searchByInfection(ArrayList<Person> List) {  
 boolean people\_inf;  
 int amount\_people\_inf = 0;  
 for (Person a : List) {  
 people\_inf = a.getInfected();  
 if (people\_inf) {  
 amount\_people\_inf += 1;  
 System.*out*.println(a.getInformation());  
 }  
 }  
 System.*out*.println("Количество зараженных " + amount\_people\_inf);  
 }  
}

Person:

abstract public class Person {  
 protected String name;  
 protected String position;  
 protected int age;  
 protected int id;  
 protected String infected;  
  
 public String getInformation() {  
 return String.*format*("ID: %s Имя и Фамилия: %s Возраст: %s Позиция: %s", this.id, this.name, this.age, this.position);  
 };  
 public boolean getInfected(){  
 if (infected.equals("+")){  
 return true;  
 }  
 else{  
 return false;  
 }  
 };  
 public abstract String getLisence();  
}

Student:

public class Student extends Person{  
  
 public Student(String name, int age, String position, int id, String infected) {  
 this.name = name;  
 this.position = "Студент";  
 this.age = age;  
 this.id = id;  
 this.infected = infected;  
 }  
  
 @Override  
 public String getLisence() {  
 if (this.infected.equals("-")) {  
 return String.*format*("Студент \nИмя и Фамилия: %s может продолжать ходить на пары", this.name);  
 }  
 else{  
 return String.*format*("Студент \n Имя и Фамилия: %s не может продолжать ходить на пары", this.name);  
 }  
 }  
}

Professor:

public class Professor extends Person{  
  
 public Professor(String name, int age, String position, int id, String infected) {  
 this.name = name;  
 this.position = "Профессор";  
 this.age = age;  
 this.id = id;  
 this.infected = infected;  
 }  
  
  
 @Override  
 public String getLisence() {  
 if (this.infected.equals("+")){  
 return String.*format*("Профессор \nИмя и Фамилия: %s не может продолжать ходить на работу в связи с возрастом", this.name);  
 }  
 else {  
 if (this.age < 65) {  
 return String.*format*("Профессор \nИмя и Фамилия: %s может продолжать ходить на работу", this.name);  
 } else {  
 return String.*format*("Профессор \nИмя и Фамилия: %s не может продолжать ходить на работу в связи с возрастом", this.name);  
 }  
 }  
 }  
}

HeadOfDepartment:

public class HeadOfTheDepartment extends Person {  
  
 public HeadOfTheDepartment(String name, int age, String position, int id, String infected) {  
 this.name = name;  
 this.position = "Завкафедрой";  
 this.age = age;  
 this.id = id;  
 this.infected = infected;  
 }  
  
  
  
 @Override  
 public String getLisence() {  
 if (this.infected.equals("+")){  
 return String.*format*("Завкафедрой \nИмя и Фамилия: %s не может продолжать ходить на работу в связи с возрастом", this.name);  
 }  
 else {  
 if (this.age < 65) {  
 return String.*format*("Завкафедрой \nИмя и Фамилия: %s может продолжать ходить на работу", this.name);  
 } else {  
 return String.*format*("Завкафедрой \nИмя и Фамилия: %s не может продолжать ходить на работу в связи с возрастом", this.name);  
 }  
 }  
 }  
}

Список использованных источников

1. Аршинский В.Л. Объектно-ориентированное программирование. Метод. указания по выполнению лабораторных работ / В.Л. Аршинский. – Иркутск: Изд-во ИРНИТУ, 2015. – 23 c.