**Лабораторна робота №15**

**Колекції в Java**

**Мета:** Ознайомлення з бібліотекою колекцій Java SE. Використання колекцій для розміщення об'єктів розроблених класів.

1. **ВИМОГИ**
2. Розробити консольну програму для реалізації завдання обробки даних згідно прикладної області.
3. Для розміщення та обробки даних використовувати контейнери (колекції) і алгоритми з Java Collections Framework.
4. Забезпечити обробку колекції об'єктів: додавання, видалення, пошук, сортування згідно розділу Прикладні задачі л.р. №10.
5. Передбачити можливість довготривалого зберігання даних: 1) за допомогою стандартної серіалізації; 2) не використовуючи протокол серіалізації.
6. Продемонструвати розроблену функціональність в діалоговому та автоматичному режимах за результатом обробки параметрів командного рядка.
   1. **Розробник**

* П.І.Б: Абдулаєв І. З.
* Группа: КІТ-119в
* Варіант: 1

1. **ОПИС ПРОГРАМИ**

**2.1 Засоби ООП:**

Scanner inInt, inStr = new Scanner(System.in) – для введення обраних опцій користувачем з клавіатури;

XMLEncoder encoder = new XMLEncoder(new BufferedOutputStream(new FileOutputStream(filename));

encoder.writeObject(container); – нестандартна серіалізація;

XMLDecoder decoder = new XMLDecoder(new BufferedInputStream(new FileInputStream(filename)));

container = (ArrayList<Challanger>) decoder.readObject(); – нестандартна десеріалізація;

ObjectOutputStream oos = new ObjectOutputStream(new BufferedOutputStream(newFileOutputStream(filename)));

oos.writeObject(container);

oos.flush(); – стандартна серіалізація;

ObjectInputStream ois = new ObjectInputStream(new BufferedOutputStream(new FileInputStream(filename)));

container = (ArrayList<Challanger>) ois.readObject(); – стандартна десеріалізація;

Pattern pattern = Pattern.compile() – компілює регулярний вираз у шаблон;

Matcher matcher = pattern.matcher(data); – створює matcher, який буде відповідати даному вводу для цього шаблону.

* 1. **Ієрархія та структура класів**

Було створено класи Main (головний клас програми), Challanger (містить всі поля та методи предметної області «Кадрове агентство»), 3 класи, що реалізують інтерфейс Comparator для сортування за певними критеріями, а також підключено класи з попередньої роботи: DemandsToWork та WorkExperience.

**2.3** **Важливі фрагменти програми**

**Клас Main**

package ua.khpi.oop.abdulaev15;  
  
import java.beans.XMLDecoder;  
import java.beans.XMLEncoder;  
import java.io.BufferedInputStream;  
import java.io.BufferedOutputStream;  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.List;  
import java.util.Scanner;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
  
import ua.khpi.oop.abdulaev07.DemandsToWork;  
import ua.khpi.oop.abdulaev07.WorkExperience;  
  
public class Main {  
  
 public static void main(String[] args) {  
 ArrayList<Challanger> recruitingAgency = new ArrayList<Challanger>();  
  
 for (String str : args) {  
 if(str.equals("-a") || str.equals("-auto")) {  
 recruitingAgency = *auto*(recruitingAgency);  
 return;  
 }  
 }  
 recruitingAgency = *menu*(recruitingAgency);  
 }  
  
 private static ArrayList<Challanger> auto(ArrayList<Challanger> recruitingAgency) {  
 System.*out*.println("Adding elements...");  
  
 File file = new File("recruitingAgency11.txt");  
  
 try {  
 String education;  
 int day;  
 int month;  
 int year;  
 String specializationPrevious;  
 int experience;  
 String specializationNext;  
 int minSalary;  
 String conditions;  
 Scanner reader = new Scanner(file);  
 while(reader.hasNextLine()) {  
 String data = reader.nextLine();  
 Pattern pattern = Pattern.*compile*("((\\w+(|\\s))\*,\\s([1-9]|[12]\\d|3[01])\\.([1-9]|1[012])\\.((19|20)\\d{2}),\\s" +  
 "(\\w+.)+,\\s([0-9]|[1-6][0-9]),\\s(\\w+.)+,\\s([1-9]\\d{3,}),\\s(\\w+(\\.|\\s)(\\s|))+)");  
 Matcher matcher = pattern.matcher(data);  
 if(matcher.matches()) {  
 String[] information = data.split(",\\s");  
 education = information[0];  
 specializationPrevious = information[2];  
 experience = Integer.*parseInt*(information[3]);  
 specializationNext = information[4];  
 minSalary = Integer.*parseInt*(information[5]);  
 conditions = information[6];  
 String[] date = information[1].split("\\.");  
 day = Integer.*parseInt*(date[0]);  
 month = Integer.*parseInt*(date[1]);  
 year = Integer.*parseInt*(date[2]);  
  
 int id = recruitingAgency.size();  
  
 WorkExperience workExperienceAdd = new WorkExperience(specializationPrevious, experience);  
 DemandsToWork demandsToWorkAdd = new DemandsToWork(specializationNext,minSalary,conditions);  
 Challanger challangerAdd = new Challanger(id++,education,day,month,year,workExperienceAdd,demandsToWorkAdd);  
 recruitingAgency.add(challangerAdd);  
 }  
 }  
 reader.close();  
 System.*out*.println("Adding was end.\n");  
 } catch (FileNotFoundException e){  
 e.printStackTrace();  
 }  
  
 System.*out*.println("List in Recruiting Agency:\n");  
 if(recruitingAgency.size() > 0) {  
 for(var element : recruitingAgency) {  
 element.print();  
 }  
 }  
 else {  
 System.*out*.println("The recruiting agency is empty!\n");  
 }  
  
 *task*(recruitingAgency);  
  
 Comparator comp = new workExperienceComparator();  
 Collections.*sort*(recruitingAgency, comp);  
  
 System.*out*.println("Data sorted by work experience");  
  
 System.*out*.println("List in Recruiting Agency:\n");  
 if(recruitingAgency.size() > 0) {  
 for(var element : recruitingAgency) {  
 element.print();  
 }  
 }  
  
 return recruitingAgency;  
 }  
  
 private static ArrayList<Challanger> menu(ArrayList<Challanger> recruitingAgency) {  
 boolean endprog = false;  
 Scanner inInt = new Scanner(System.*in*);  
 Scanner inStr = new Scanner(System.*in*);  
 int menu;  
 int menuSort;  
 int menuSerialization;  
 int menuDeserialization;  
  
 while(!endprog)  
 {  
 System.*out*.println("1. Show all challanger");  
 System.*out*.println("2. Add challanger");  
 System.*out*.println("3. Delete chellanger");  
 System.*out*.println("4. Clear list");  
 System.*out*.println("5. Is empty recruiting agency?");  
 System.*out*.println("6. Sort data");  
 System.*out*.println("7. Serialize data");  
 System.*out*.println("8. Deserialize data");  
 System.*out*.println("9. Task");  
 System.*out*.println("0. Exit");  
 System.*out*.print("Enter option: ");  
 try  
 {  
 menu = inInt.nextInt();  
 }  
 catch(java.util.InputMismatchException e)  
 {  
 System.*out*.println("Error! Îøèáêà ââîäà.");  
 endprog = true;  
 menu = 0;  
 }  
 System.*out*.println();  
 switch(menu)  
 {  
 case 1:  
 if(recruitingAgency.size() > 0) {  
 for(var element : recruitingAgency) {  
 element.print();  
 }  
 }  
 else {  
 System.*out*.println("The recruiting agency is empty!\n");  
 }  
 break;  
 case 2:  
 String education;  
 int day;  
 int month;  
 int year;  
 String specializationPrevious;  
 int experience;  
 String specializationNext;  
 int minSalary;  
 String conditions;  
  
 Pattern patternEducation = Pattern.*compile*("(\\w+.)+");  
 Pattern patternDay = Pattern.*compile*("([1-9]|[12]\\d|3[01])");  
 Pattern patternMonth = Pattern.*compile*("([1-9]|1[012])");  
 Pattern patternYear = Pattern.*compile*("(19|20)\\d{2}");  
 Pattern patternSpeñialization = Pattern.*compile*("(\\w+.)+");  
 Pattern patternExperience = Pattern.*compile*("[0-9]|[1-6][0-9]");  
 Pattern patternMinSalary = Pattern.*compile*("(^[1-9]\\d{3,})");  
 Pattern patternConditions = Pattern.*compile*("(\\w+(\\.|\\s)(\\s|))+");  
  
 System.*out*.println("Enter education of challanger: ");  
 try {  
 education = inStr.nextLine();  
 education = *stringRegexCheck*(education, patternEducation);  
 }catch(java.util.InputMismatchException e) {  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
  
 System.*out*.println("Enter day of dismissal: ");  
 try {  
 day = inInt.nextInt();  
 day = *intRegexCheck*(day, patternDay);  
 } catch(java.util.InputMismatchException e) {  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
  
 System.*out*.println("Enter month of dismissal: ");  
 try {  
 month = inInt.nextInt();  
 month = *intRegexCheck*(month, patternMonth);  
 } catch(java.util.InputMismatchException e) {  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
  
 System.*out*.println("Enter year of dismissal: ");  
 try {  
 year = inInt.nextInt();  
 year = *intRegexCheck*(year, patternYear);  
 } catch(java.util.InputMismatchException e) {  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
  
 System.*out*.println("Enter pervious job: ");  
 try {  
 specializationPrevious = inStr.nextLine();  
 specializationPrevious = *stringRegexCheck*(specializationPrevious, patternSpeñialization);  
 } catch(java.util.InputMismatchException e) {  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
  
 System.*out*.println("Enter experience of working: ");  
 try {  
 experience = inInt.nextInt();  
 experience = *intRegexCheck*(experience, patternExperience);  
 } catch(java.util.InputMismatchException e){  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
  
 System.*out*.println("Enter next job: ");  
 try {  
 specializationNext = inStr.nextLine();  
 specializationNext = *stringRegexCheck*(specializationNext, patternSpeñialization);  
 } catch(java.util.InputMismatchException e) {  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
  
 System.*out*.println("Enter min salary: ");  
 try {  
 minSalary = inInt.nextInt();  
 minSalary = *intRegexCheck*(minSalary, patternMinSalary);  
 }catch (java.util.InputMismatchException e) {  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
  
 System.*out*.println("Enter whishes to the next job: ");  
 try {  
 conditions = inStr.nextLine();  
 conditions = *stringRegexCheck*(conditions, patternConditions);  
 } catch(java.util.InputMismatchException e){  
 System.*out*.println("Error! Incorect input!");  
 break;  
 }  
 int id = recruitingAgency.size();  
  
 WorkExperience workExperienceAdd = new WorkExperience(specializationPrevious, experience);  
 DemandsToWork demandsToWorkAdd = new DemandsToWork(specializationNext,minSalary,conditions);  
 Challanger challangerAdd = new Challanger(id++,education,day,month,year,workExperienceAdd,demandsToWorkAdd);  
 recruitingAgency.add(challangerAdd);  
 break;  
 case 3:  
 System.*out*.println("Enter ID to delete: ");  
 int id\_to\_remove = 0;  
 int delete = inInt.nextInt();  
 boolean isExist = false;  
 if(recruitingAgency.size() > 0) {  
 for(var element : recruitingAgency) {  
 if(element.getRegistrationNum() == delete) {  
 isExist = true;  
 break;  
 }  
 id\_to\_remove++;  
 }  
 if(isExist) {  
 recruitingAgency.remove(id\_to\_remove);  
 }  
 else  
 System.*out*.println("Error! Wrong ID.");  
 }  
 break;  
 case 4:  
 recruitingAgency.clear();  
 System.*out*.println("RecruitingAgency is empty now.\n");  
 break;  
 case 5:  
 if(recruitingAgency.isEmpty())  
 System.*out*.println("Recruiting agency is empty.\n");  
 else  
 System.*out*.println("Recruiting agency is not empty.");  
 break;  
 case 6:  
 System.*out*.println("1. Sort by Registration Number");  
 System.*out*.println("2. Sort by work experience");  
 System.*out*.println("3. Sort by demand to min salary");  
 System.*out*.println("4. Return to menu");  
 System.*out*.println("Enter option: ");  
 try  
 {  
 menuSort = inInt.nextInt();  
 }  
 catch(java.util.InputMismatchException e)  
 {  
 System.*out*.println("Error! Îøèáêà ââîäà.");  
 break;  
 }  
 switch(menuSort) {  
 case 1:  
 Comparator comp\_id = new idComparator();  
 Collections.*sort*(recruitingAgency, comp\_id);  
 System.*out*.println("Data sorted by Registration Number\n");  
 break;  
 case 2:  
 Comparator comp\_work = new workExperienceComparator();  
 Collections.*sort*(recruitingAgency, comp\_work);  
 System.*out*.println("Data sorted by work experience\n");  
 break;  
 case 3:  
 Comparator comp\_min = new minSalazyComparator();  
 Collections.*sort*(recruitingAgency, comp\_min);  
 System.*out*.println("Data sorted by demand to min salary");  
 break;  
 case 4:  
  
 break;  
 default:  
 System.*out*.println("Error! Wrong num in Sort menu.");  
 break;  
 }  
 break;  
 case 7:  
 String filenameSerialization;  
 String filenameXML;  
  
 System.*out*.println("1. Serialization");  
 System.*out*.println("2. XML serialization");  
 System.*out*.println("0. Exit serialization");  
 try  
 {  
 menuSerialization = inInt.nextInt();  
 }  
 catch(java.util.InputMismatchException e)  
 {  
 System.*out*.println("Error! Îøèáêà ââîäà.");  
 menuSerialization = 0;  
 }  
 switch(menuSerialization)  
 {  
 case 1:  
 System.*out*.println("\nEnter file name: ");  
 filenameSerialization = inStr.nextLine();  
 if (filenameSerialization.indexOf(".ser") == -1) {  
 filenameSerialization += ".ser";  
 }  
 try(ObjectOutputStream oos = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream (filenameSerialization)))){  
 oos.writeObject(recruitingAgency);  
 System.*out*.println("Serialization successful.");  
 } catch (Exception e){  
 System.*out*.println(e.getMessage());  
 }  
 break;  
 case 2:  
 System.*out*.print("Enter XML filename: ");  
 filenameXML = inStr.nextLine();  
 if (filenameXML.indexOf(".xml") == -1)  
 filenameXML += ".xml";  
 try(XMLEncoder encoder = new XMLEncoder(new BufferedOutputStream(new FileOutputStream (filenameXML)))){  
 encoder.writeObject(recruitingAgency);  
 System.*out*.println("Serialization successful.");  
 } catch (Exception e){  
 System.*out*.println(e.getMessage());  
 }  
 break;  
 case 0:  
 break;  
 default:  
 System.*out*.println("Error! Wrong num in menu.");  
 break;  
 }  
 break;  
 case 8:  
 String filenameDeserialization;  
  
 System.*out*.println("1. Deserialization");  
 System.*out*.println("2. XML deserialization");  
 System.*out*.println("0. Exit deserialization");  
 try  
 {  
 menuDeserialization = inInt.nextInt();  
 }  
 catch(java.util.InputMismatchException e)  
 {  
 System.*out*.println("Error! Îøèáêà ââîäà.");  
 menuDeserialization = 0;  
 }  
 switch(menuDeserialization)  
 {  
 case 1:  
 System.*out*.println("\nEnter file name: ");  
 filenameDeserialization = inStr.nextLine();  
 if (filenameDeserialization.indexOf(".ser") == -1) {  
 filenameDeserialization += ".ser";  
 }  
 try(ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(new FileInputStream (filenameDeserialization)))){  
 recruitingAgency.clear();  
 recruitingAgency = (ArrayList<Challanger>) ois.readObject();  
 System.*out*.println("Deserialization successful.");  
 } catch (Exception e){  
 System.*out*.println(e.getMessage());  
 }  
 break;  
 case 2:  
 System.*out*.print("Enter XML filename: ");  
 filenameDeserialization = inStr.nextLine();  
 if (filenameDeserialization.indexOf(".xml") == -1)  
 filenameDeserialization += ".xml";  
 try(XMLDecoder decoder = new XMLDecoder(new BufferedInputStream(new FileInputStream (filenameDeserialization)))){  
 recruitingAgency.clear();  
 recruitingAgency = (ArrayList<Challanger>) decoder.readObject();  
 System.*out*.println("Deserialization successful.");  
 } catch (Exception e){  
 System.*out*.println(e.getMessage());  
 }  
 break;  
 case 0:  
 break;  
 default:  
 System.*out*.println("Error! Wrong num in menu.");  
 break;  
 }  
 break;  
 case 9:  
 *task*(recruitingAgency);  
 break;  
 case 0:  
 endprog = true;  
 recruitingAgency.clear();  
 inInt.close();  
 inStr.close();  
 break;  
 default:  
 System.*out*.println("Error! Wrong num in menu.");  
 break;  
 }  
 }  
 return recruitingAgency;  
 }  
 public static int intRegexCheck(int value, Pattern pattern)  
 {  
 Matcher matcher;  
 Scanner in = new Scanner(System.*in*);  
 boolean ready = false;  
 do  
 {  
 matcher = pattern.matcher(Integer.*toString*(value));  
 if(!matcher.matches())  
 {  
 System.*out*.println("You've entered the wrong data. Try again:");  
 value = in.nextInt();  
 }  
 else  
 ready = true;  
 }  
 while(!ready);  
 return value;  
 }  
  
 public static String stringRegexCheck(String value, Pattern pattern)  
 {  
 Matcher matcher;  
 Scanner in = new Scanner(System.*in*);  
 boolean ready = false;  
 do  
 {  
 matcher = pattern.matcher(value);  
 if(!matcher.matches())  
 {  
 System.*out*.println("You've entered the wrong data. Try again:");  
 value = in.nextLine();  
 }  
 else  
 ready = true;  
 }  
 while(!ready);  
 return value;  
 }  
  
 public static void task(List<Challanger> recruitingAgency) {  
 String conditions;  
 String prevJob;  
 Pattern patternManager = Pattern.*compile*(".\*(M|m)anager.\*");  
 Pattern patternNot = Pattern.*compile*(".\*(N|n)ot.\*");  
 Pattern patternBuisnessTrip = Pattern.*compile*(".\*(B|b)uisness trip.\*");  
 ArrayList<Challanger> task = new ArrayList<Challanger>();  
  
 if(recruitingAgency.size() > 0) {  
 for(var element : recruitingAgency) {  
 conditions = element.getDemandsToWork().getConditions();  
 prevJob = element.getWorkExperience().getSpecialization();  
 Matcher matcher = patternManager.matcher(prevJob);  
 if(matcher.matches()) {  
 Matcher matcherNot = patternNot.matcher(conditions);  
 if(matcherNot.matches()) {  
 Matcher matcherBuisnessTrip = patternBuisnessTrip.matcher(conditions);  
 if(matcherBuisnessTrip.matches()) {  
 task.add(element);  
 }  
 }  
 }  
 }  
 }  
 if(task.size() > 0) {  
 System.*out*.println("\nChallangers with wishes to dose not have a buiness trip:\n");  
 for(var challanger : task) {  
 challanger.print();  
 }  
 System.*out*.println();  
 }  
 else {  
 System.*out*.println("\nChallangers without wishes to dose not have a buisness trip.\n");  
 }  
  
 }  
  
}

**Класс Challanger**

package ua.khpi.oop.abdulaev15;  
  
import java.io.Serializable;  
import java.util.Comparator;  
import java.util.Comparator;  
  
import ua.khpi.oop.abdulaev07.DemandsToWork;  
import ua.khpi.oop.abdulaev07.WorkExperience;  
  
  
public class Challanger implements Serializable {  
 private static final long *serialVersionUID* = -5616690429713031285L;  
  
 private int registrationNum;  
 private String education;  
 private int dismissalDay;  
 private int dismissalMonth;  
 private int dismissalYear;  
 private DemandsToWork demandsToWork;  
 private WorkExperience workExperience;  
  
 */\*\*  
 \* Конструктор  
 \** ***@param*** *registrationNum ID претндента  
 \** ***@param*** *education образование претендента  
 \** ***@param*** *dismissalDay день увольнения претендента  
 \** ***@param*** *dismissalMonth месяц увольнения претендента  
 \** ***@param*** *dismissalYear год увольнения претендента  
 \** ***@param*** *workExperience опыт работы претендента  
 \** ***@param*** *demandsToWork пожелания к будующей работе  
 \*/* public Challanger(int registrationNum, String education, int dismissalDay, int dismissalMonth, int dismissalYear, WorkExperience workExperience, DemandsToWork demandsToWork ) {  
 this.registrationNum = registrationNum;  
 this.education = education;  
 this.dismissalDay = dismissalDay;  
 this.dismissalMonth = dismissalMonth;  
 this.dismissalYear = dismissalYear;  
 this.workExperience = workExperience;  
 this.demandsToWork = demandsToWork;  
 }  
 public Challanger()  
 {  
 super();  
 }  
 */\*\*  
 \* Геттер ID претендента  
 \** ***@return*** *ID претендента  
 \*/* public int getRegistrationNum() {  
 return registrationNum;  
 }  
 */\*\*  
 \* Сеттер ID претендента  
 \** ***@param*** *registrationNum ID претендента  
 \*/* public void setRegistrationNum(int registrationNum) {  
 this.registrationNum = registrationNum;  
 }  
 */\*\*  
 \* Геттер образования претендента  
 \** ***@return*** *образование претендента  
 \*/* public String getEducation() {  
 return education;  
 }  
 */\*\*  
 \* Сеттер образования претендента  
 \** ***@param*** *education Образование претендента  
 \*/* public void setEducation(String education) {  
 this.education = education;  
 }  
 */\*\*  
 \* Геттер дня увольнения  
 \** ***@return*** *день увольнения  
 \*/* public int getDismissalDay() {  
 return dismissalDay;  
 }  
 */\*\*  
 \* Сеттер дня увольнеия  
 \** ***@param*** *dismissalDay день увольнения  
 \*/* public void setDismissalDay(int dismissalDay) {  
 this.dismissalDay = dismissalDay;  
 }  
 */\*\*  
 \* Геттер месяца увольнения  
 \** ***@return*** *месяц увольнения  
 \*/* public int getDismissalMonth() {  
 return dismissalMonth;  
 }  
 */\*\*  
 \* Сеттер месяца увольнения  
 \** ***@param*** *dismissalMonth месяц увольнения  
 \*/* public void setDismissalMonth(int dismissalMonth) {  
 this.dismissalMonth = dismissalMonth;  
 }  
 */\*\*  
 \* Геттер года увольнения претендента  
 \** ***@return*** *год увольнения  
 \*/* public int getDismissalYear() {  
 return dismissalYear;  
 }  
 */\*\*  
 \* Сеттер года увольнения претендента  
 \** ***@param*** *dismissalYear год увольнения  
 \*/* public void setDismissalYear(int dismissalYear) {  
 this.dismissalYear = dismissalYear;  
 }  
 */\*\*  
 \* Геттер опыта работы претендента  
 \** ***@return*** *\*/* public DemandsToWork getDemandsToWork() {  
 return demandsToWork;  
 }  
 */\*\*  
 \* Сеттер требований к будующей работе  
 \** ***@param*** *demandsToWork  
 \*/* public void setDemandsToWork(DemandsToWork demandsToWork) {  
 this.demandsToWork = demandsToWork;  
 }  
  
 */\*\*  
 \* Геттер опыта работы претендента  
 \** ***@return*** *\*/* public WorkExperience getWorkExperience() {  
 return workExperience;  
 }  
  
 */\*\*  
 \* Сеттер опыта работы претендента  
 \** ***@param*** *workExperience  
 \*/* public void setWorkExperience(WorkExperience workExperience) {  
 this.workExperience = workExperience;  
 }  
  
  
 public void print() {  
 System.*out*.println("ID: " + getRegistrationNum());  
 System.*out*.println("Образование: " + getEducation());  
 System.*out*.println("Дата увольнения: " + getDismissalDay()+"/" + getDismissalMonth()+"/"+getDismissalYear());  
 System.*out*.println("---Опыт работы---");  
 System.*out*.println("Место предыдущей работы: " + getWorkExperience().getSpecialization());  
 if(getWorkExperience().getExperience() <= 4)  
 System.*out*.println("Стаж: " + getWorkExperience().getExperience() + " год(а)");  
 else  
 System.*out*.println("Стаж: " + getWorkExperience().getExperience() + " лет");  
 System.*out*.println("---Желания по будующей работе---" );  
 if(getDemandsToWork().getMinSalary() == 0 && getDemandsToWork().getSpecialization() == null && getDemandsToWork().getConditions() == null)  
 System.*out*.println("Предендет не имеет никаких желаний по будующей работе");  
 else {  
 if(getDemandsToWork().getMinSalary() != 0)  
 System.*out*.println("Желаемая минимальная зарплата: " + getDemandsToWork().getMinSalary());  
 else  
 System.*out*.println("Желаемая минимальная зарплата: Претендент не имеет пожеланий к этому пунку " );  
 if(getDemandsToWork().getSpecialization() != null)  
 System.*out*.println("Желаемая будующая работа: " + getDemandsToWork().getSpecialization());  
 else  
 System.*out*.println("Желаемая будующая работа: Претендент не имеет пожеланий к этому пунку");  
 if(getDemandsToWork().getConditions() != null)  
 System.*out*.println("Желаемые условия будующей работы: " + getDemandsToWork().getConditions());  
 else  
 System.*out*.println("Желаемые условия будующей работы: Претендент не имеет пожеланий к этому пунку");  
 }  
 System.*out*.println("------------------------------------------");  
  
 }  
}  
  
class idComparator implements Comparator<Challanger>{  
 @Override  
 public int compare(Challanger o1, Challanger o2) {  
 if(o1.getRegistrationNum() > o2.getRegistrationNum())  
 return 1;  
 else if (o1.getRegistrationNum() < o2.getRegistrationNum())  
 return -1;  
 else  
 return 0;  
 }  
}  
  
class workExperienceComparator implements Comparator<Challanger>{  
 public int compare(Challanger o1, Challanger o2) {  
 if(o1.getWorkExperience().getExperience() > o2.getWorkExperience().getExperience())  
 return 1;  
 else if (o1.getWorkExperience().getExperience() < o2.getWorkExperience().getExperience())  
 return -1;  
 else  
 return 0;  
 }  
}  
  
class minSalazyComparator implements Comparator<Challanger>{  
 @Override  
 public int compare(Challanger o1, Challanger o2) {  
 if(o1.getDemandsToWork().getMinSalary() > o2.getDemandsToWork().getMinSalary())  
 return 1;  
 else if (o1.getDemandsToWork().getMinSalary() < o2.getDemandsToWork().getMinSalary())  
 return -1;  
 else  
 return 0;  
 }  
  
}

**3 Результати роботи програми**

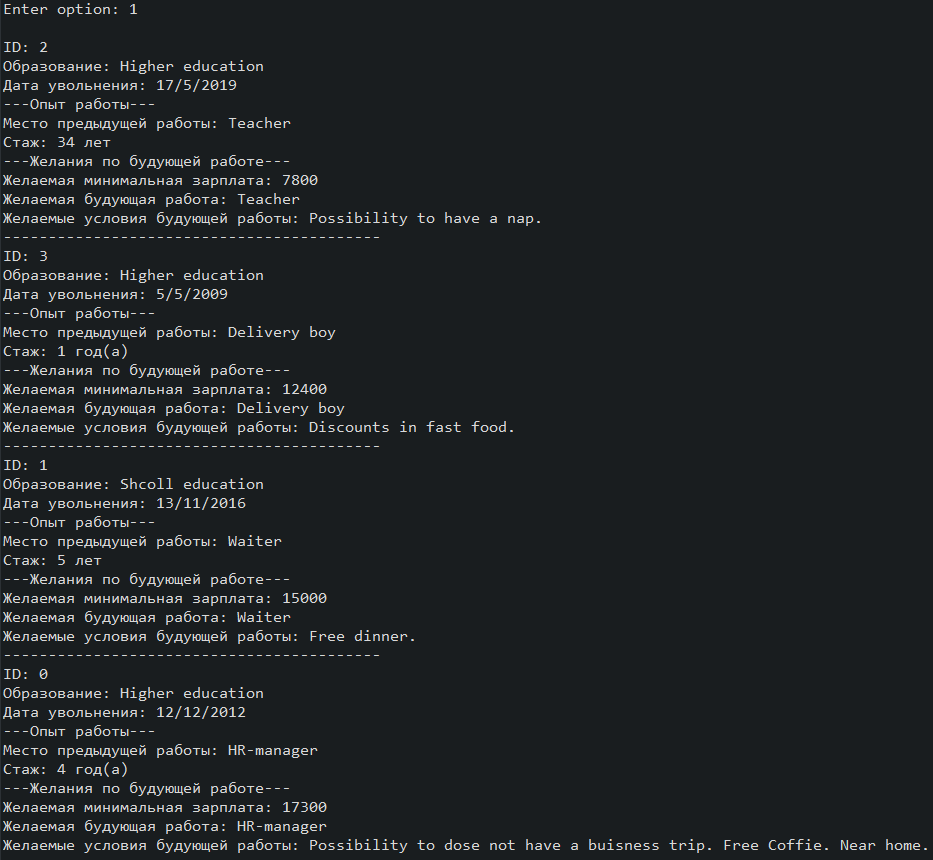


Рисунок 15.1 – Результат виводу

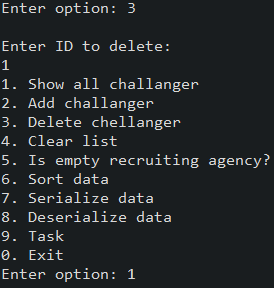


Рисунок 15.2 – Операція видалення елемента

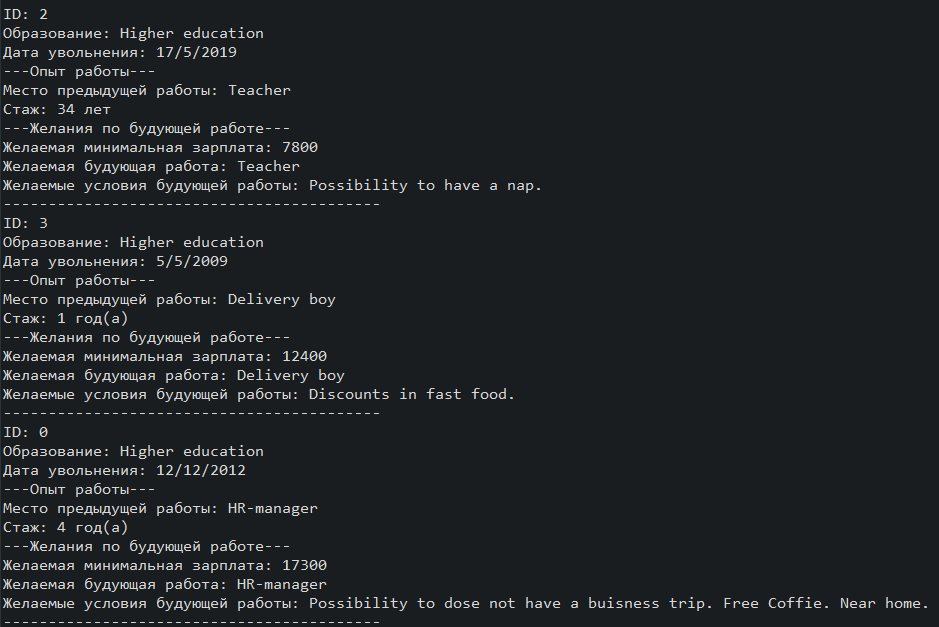


Рисунок 15.3 – Результат видалення

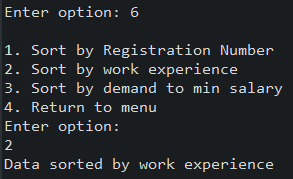


Рисунок 15.4 – Операція сортування за досвідом роботи

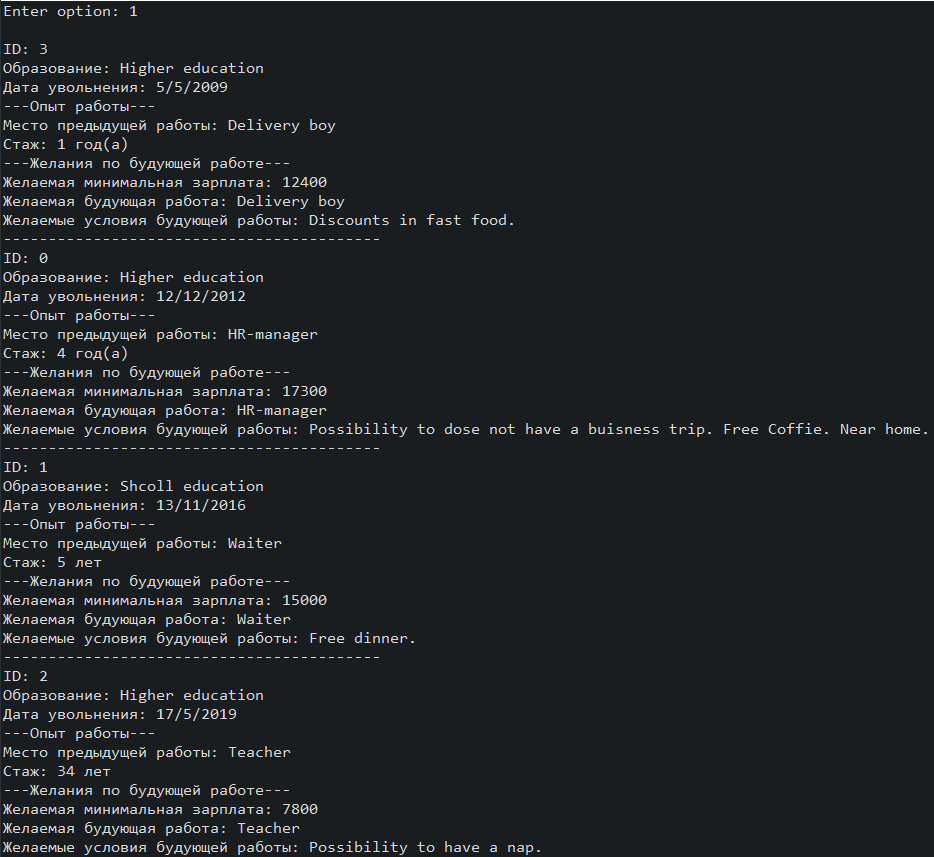


Рисунок 15.4 – Результат сортування за досвідом роботи

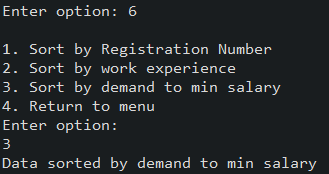


Рисунок 15.5 – Операція сортування за мінімальною зарплатнею

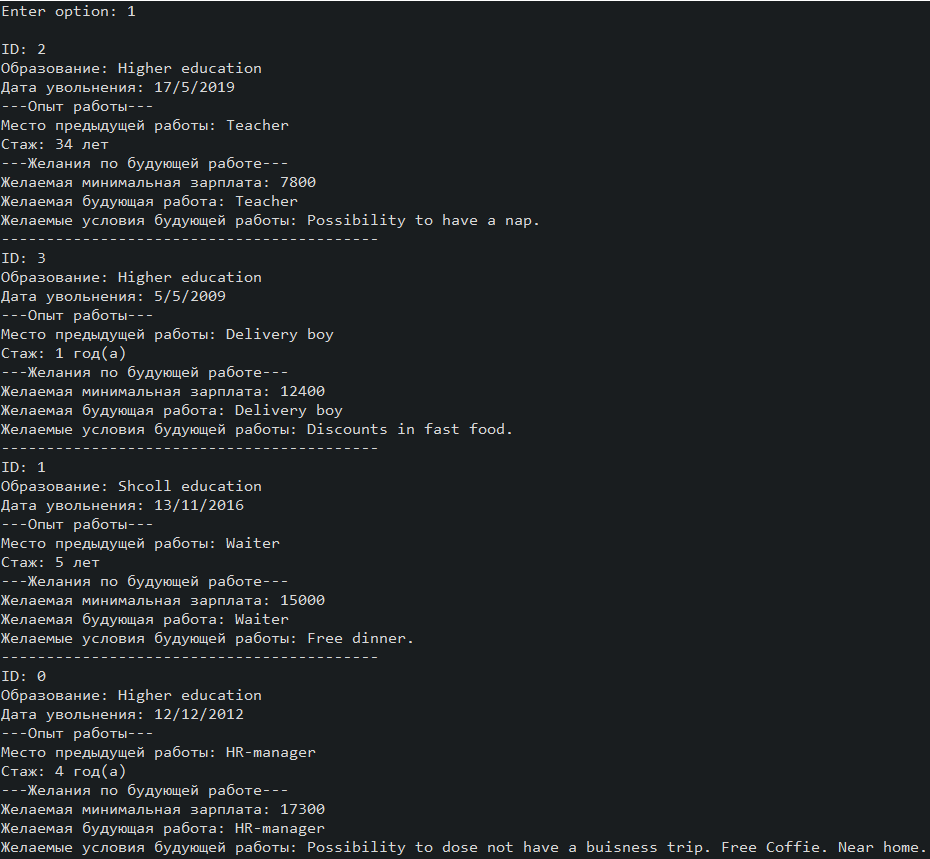


Рисунок 15.6 – Результат сортування за мінімальною заралатнею

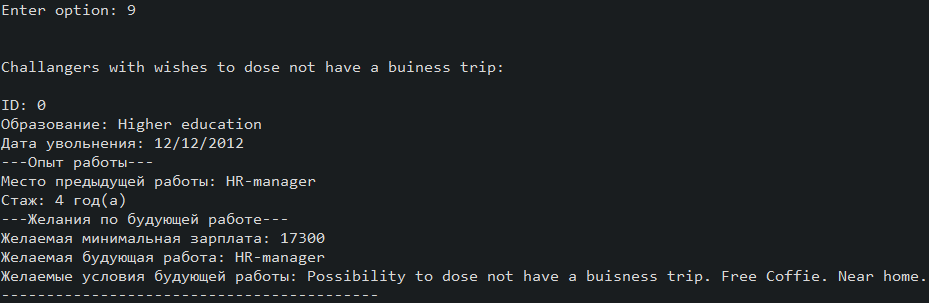


Рисунок 15.7 – Результат виконання завдання з 12 Лабораторної роботит



Рисунок 15.8 – Результат перевірки контейнера на наявність елементів

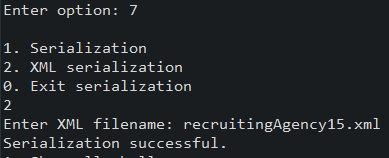


Рисунок 15.9 – Результат виконання серіалізації

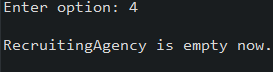


Рисунок 15.10 – Результат очищення контейнера

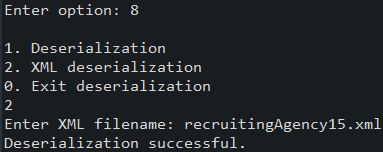


Рисунок 15.11 – Виконання десеріалізації

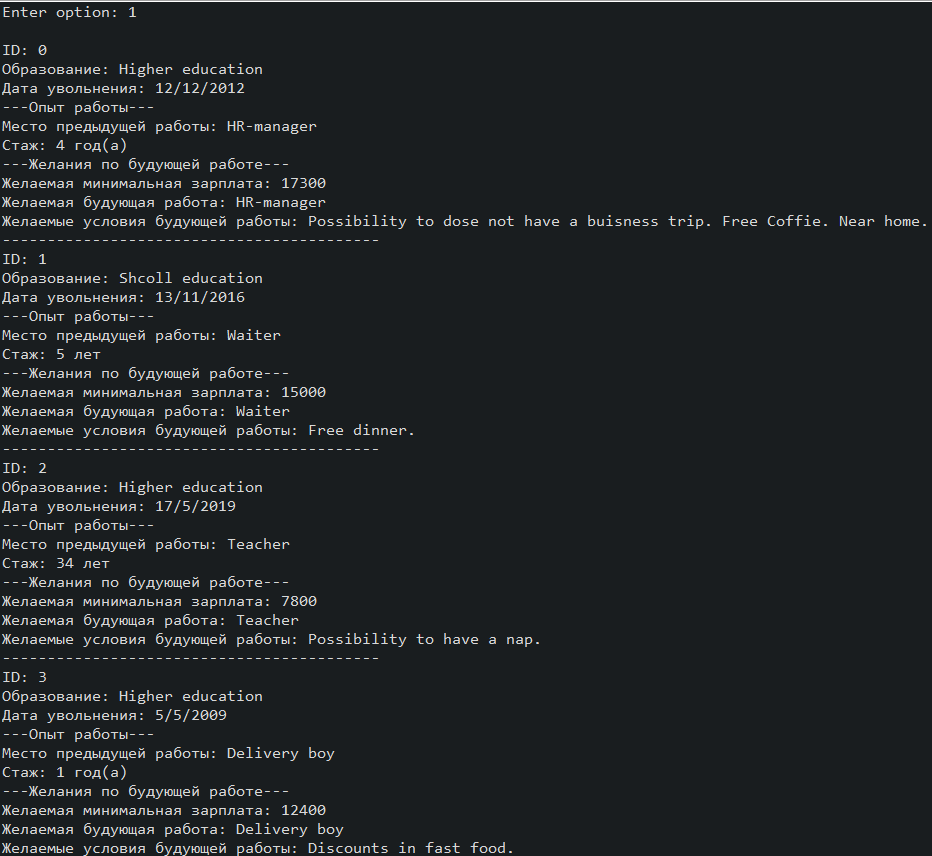


Рисунок 15.12 – Результат виконання десеріалізації

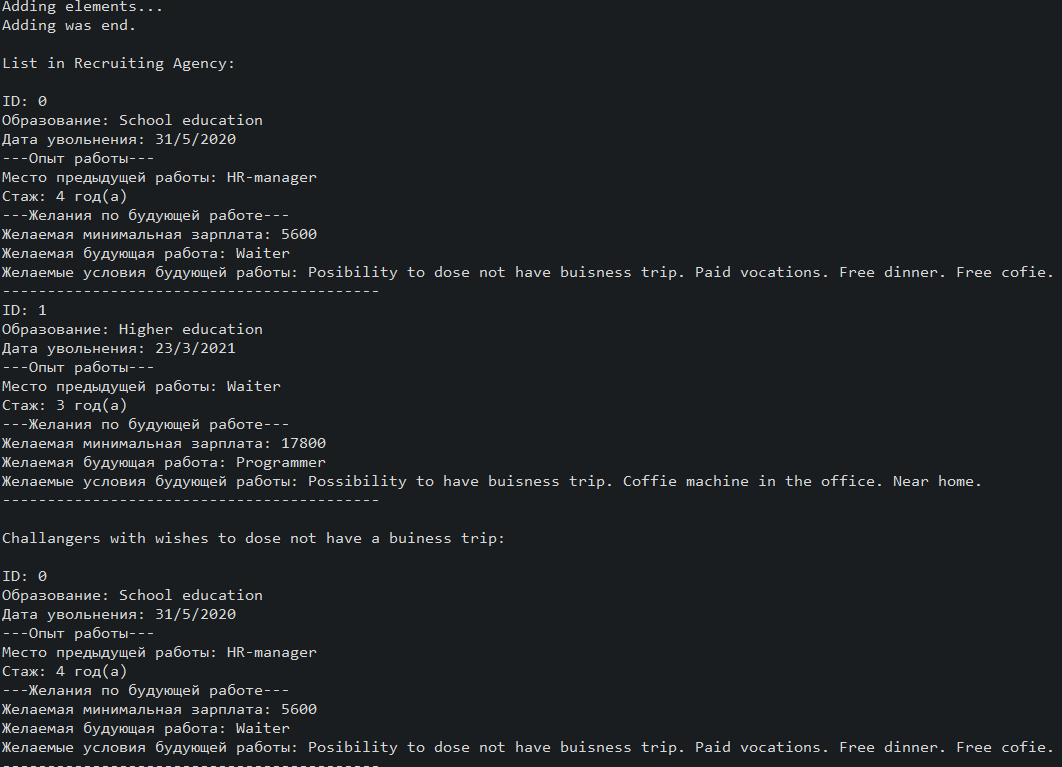


Рисунок 15.14 – Виконання програми у автоматичному режимі

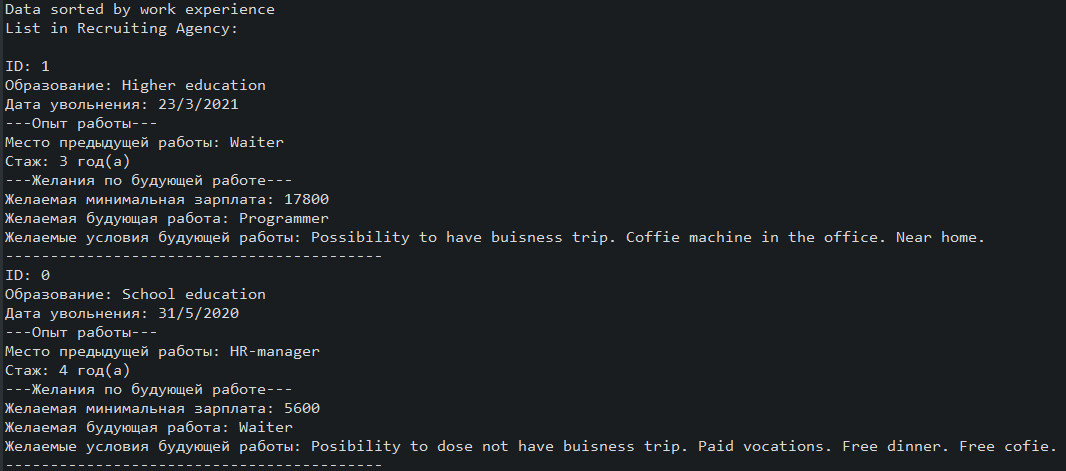


Рисунок 15.15 – Результат виконання програми у автоматичному режимі

**Висновок**

Під час виконання лабораторної роботи було набуто навички роботи з колекціями та їх обробкою в середовищі IntelliJ IDEA.