Звіт

Автор: Васильченко С., 1.КІТ118б

Дата: 09.02.2020

Лабораторна робота №9

**ПАРАМЕТРИЗАЦІЯ В JAVA**

***Мета:***

* Вивчення принципів параметризації в Java.
* Розробка параметризованих класів та методів

**Вимоги:**

1. Створити власний клас-контейнер, що параметризується (Generic Type), на основі

зв’язних списків для реалізації колекції domain-об’єктів лабораторної роботи №7.

2. Для розроблених класів-контейнерів забезпечити можливість використання їх

об’єктів у циклі foreach в якості джерела даних.

3. Забезпечити можливість збереження та відновлення колекції об’єктів: 1) за

допомогою стандартної серіалізації; 2) не використовуючи протокол серіалізації.

4. Продемонструвати розроблену функціональність: створення контейнера, додавання

елементів, видалення елементів, очищення контейнера, перетворення у масив,

перетворення у рядок, перевірку на наявність елементів.

5. Забороняється використання контейнерів (колекцій) з Java Collections Framework.

ОПИС ПРОГРАМИ

***2.1 Опис змінних:***

LinkedContainer<Recruitment> stringLinked = new LinkedContainer<>();// об’єкт параметризованого контейнера

Recruitment rec1 = new Recruitment(); // об’єкт класа кадрового агенства

Scanner scan = new Scanner(System.in); // змінна для активування зчитування з консолі

***2.2 Ієрархія та структура класів.***

final class Lab09 – головний клас. Містить метод main(точку входу у програму) та методи по роботі з програмою для реалізації індивідуального завдання.

interface Linked - інтерфейс контенеру

class Recruitment - клас прикладної задачі кадрового агенства

class LinkedContainer - параметризований клас-контейнер, котрий зберігає інформацію агенства

class DescendingIterator – клас, який реалізує обратний ітератор для переміщення по списку

**ТЕКСТ ПРОГРАМИ**

File Lab09.java:

package ua.khpi.oop.vasilchenko09.tests;

import ua.khpi.oop.vasilchenko09.First.Recruitment;

import ua.khpi.oop.vasilchenko09.MyList.LinkedContainer;

public final class Lab09 {

private Lab09() {

}

public static void main(final String[] args) {

LinkedContainer<Recruitment> stringLinked = new LinkedContainer<>();

Recruitment recruitment = new Recruitment();

System.out.println("==================");

System.out.println(stringLinked.size());

stringLinked.readRec();

System.out.println("==================");

for (Recruitment s : stringLinked) {

System.out.println(s);

}

System.out.println("==================");

System.out.println(stringLinked.size());

System.out.println("==================");

}

}

LinkedContainer.java:

package ua.khpi.oop.vasilchenko09.MyList;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.io.PrintWriter;

import java.io.Serializable;

import java.util.Arrays;

import java.util.Iterator;

import java.util.NoSuchElementException;

import ua.khpi.oop.vasilchenko09.First.Recruitment;

//обобщенный контейнер на основе связного списка

public class LinkedContainer<T extends Recruitment> implements Linked<T>, Serializable {

//конструктор инициализации

public LinkedContainer() {

head = new Node<>(null, null, tail);

tail = new Node<>(null, head, null);

head = new Node<>(null, null, tail);

}

@Override

public Iterator<T> iterator() {

return new Iterator<>() {

private int position = 0;

@Override

public boolean hasNext() {

return position < size;

}

@Override

public T next() {

if (this.hasNext()) {

return getElementByIndex(position++);

} else {

throw new NoSuchElementException();

}

}

};

}

@Override

public Iterator<T> descendingIterator() {

return new Iterator<>() {

int position = size - 1;

@Override

public boolean hasNext() {

return position >= 0;

}

@Override

public T next() {

if (this.hasNext()) {

position--;

return getElementByIndex(position--);

} else {

throw new NoSuchElementException();

}

}

};

}

@Override

public boolean notEmpty() {

return size > 0;

}

@Override

public void clear() {

for (Node<T> x = head; x != null; ) {

Node<T> next = x.nextElem;

x.currentElem = null;

x.nextElem = null;

x.prevElem = null;

x = next;

}

head = null;

tail = null;

tail = new Node<>(null, head, null);

head = new Node<>(null, null, tail);

size = 0;

}

@Override

public void add(final T obj) {

addLast(obj);

}

@Override

public void saveAll() {

try {

File file = new File("save.txt");

if (!file.exists()) {

file.createNewFile();

}

PrintWriter pw = new PrintWriter(file);

System.out.println();

pw.println(size);

for (int i = 0; i < size; i++) {

pw.println(getElementByIndex(i));

}

pw.close();

} catch (IOException e) {

System.out.println("Error" + e);

}

}

@Override

public void saveRec() {

try {

File file = new File("save.txt");

if (!file.exists()) {

file.createNewFile();

}

PrintWriter pw = new PrintWriter(file);

Recruitment temp;

System.out.println();

pw.println(size);

for (int i = 0; i < size; i++) {

temp = (Recruitment) getElementByIndex(i);

pw.println(temp.getFirm());

pw.println(temp.getSpecialty());

pw.println(temp.getWorkingConditions());

pw.println(temp.getPayment());

pw.println(temp.getConfirms());

if (temp.getConfirms()) {

pw.println(temp.getNeedsSpeciality());

pw.println(temp.getExperience());

pw.println(temp.getEducation());

}

}

pw.close();

} catch (IOException e) {

System.out.println("Error" + e);

}

}

@Override

public void readAll() {

try (BufferedReader br = new BufferedReader(new FileReader("save.txt"))) {

Object temp;

String line;

line = br.readLine();

int count = Integer.parseInt(line);

for (int i = 0; i < count; i++) {

line = br.readLine();

temp = line;

add((T) temp);

}

} catch (IOException ex) {

ex.printStackTrace();

}

}

@Override

public void readRec() {

try (BufferedReader br = new BufferedReader(new FileReader("save.txt"))) {

Recruitment temp = new Recruitment();

String line;

line = br.readLine();

int count = Integer.parseInt(line);

for (int i = 0; i < count; i++) {

line = br.readLine();

temp.setFirm(line);

line = br.readLine();

temp.setSpecialty(line);

line = br.readLine();

temp.setWorkingConditions(line);

line = br.readLine();

temp.setPayment(Integer.parseInt(line));

line = br.readLine();

temp.setConfirm(Boolean.parseBoolean(line));

if (temp.getConfirms()) {

line = br.readLine();

temp.setNeedsSpeciality(line);

line = br.readLine();

temp.setExperience(Integer.parseInt(line));

line = br.readLine();

temp.setEducation(line);

}

add((T) new Recruitment(temp));

}

} catch (IOException ex) {

ex.printStackTrace();

}

}

public Object[] toArray() {

Object[] result = new Object[size];

int i = 0;

for (Node<T> temp = head; i < size; temp = temp.nextElem) {

result[i++] = temp.currentElem;

}

return result;

}

@Override

public String toString() {

return Arrays.toString(toArray());

}

// // @Override

// public void saveSerializable(LinkedContainer<T> obj) throws IOException {

// FileOutputStream file = new FileOutputStream("save.data");

// ObjectOutputStream object = new ObjectOutputStream(file);

// object.writeObject(obj);

// object.close();

// }

private Node<T> head; //первый элемент

private Node<T> tail; //последний элемент

private int size = 0; //размер списка

@Override

public void addLast(final T obj) {

Node<T> prev = tail; //сохранение данных хвоста

prev.setCurrentElem(obj); //установка значения

tail = new Node<>(null, prev, null); //изменение указателя хвоста

prev.setNextElem(tail); //установка указателя на хвост

size++; //увелечение размера списка

}

@Override

public void addFirst(final T obj) {

Node<T> next = head;

next.setCurrentElem(obj);

head = new Node<>(null, null, next);

next.setPrevElem(head);

size++;

}

@Override

public int size() {

return size;

}

@Override

public T getElementByIndex(final int index) {

Node<T> target = head.getNextElem(); //след элемент первого узла

for (int i = 0; i < index; i++) {

target = getNextElement(target);

}

return target.getCurrentElem();

}

private Node<T> getNextElement(final Node<T> index) {

return index.getNextElem();

}

// head -> null & tail -> null

// null <- prevElem [head(t = null)] nextElem-> & <- prevElem [head(t = null)] nextElem -> null

private class Node<T> {

private T currentElem; //текущий

private Node<T> prevElem; //предыдущий

private Node<T> nextElem; //следущий

//конструктор копирования

Node(final T currentElem, final Node<T> prevElem, final Node<T> nextElem) {

this.currentElem = currentElem;

this.prevElem = prevElem;

this.nextElem = nextElem;

}

//геттеры и сеттеры

public T getCurrentElem() {

return currentElem;

}

public void setCurrentElem(final T currentElem) {

this.currentElem = currentElem;

}

public Node<T> getPrevElem() {

return prevElem;

}

public void setPrevElem(final Node<T> prevElem) {

this.prevElem = prevElem;

}

public Node<T> getNextElem() {

return nextElem;

}

public void setNextElem(final Node<T> nextElem) {

this.nextElem = nextElem;

}

}

}

Recruitment.java :

package ua.khpi.oop.vasilchenko09.First;

import java.util.Scanner;

public class Recruitment {

private String firm;

private String specialty;

private String workingConditions;

private int payment;

private String needsSpeciality;

private int experience;

private String education;

private boolean confirm ;

public void setFirm(final String firm) {

this.firm = firm;

}

public void setSpecialty(final String specialty) {

this.specialty = specialty;

}

public void setWorkingConditions(final String workingConditions) {

this.workingConditions = workingConditions;

}

public void setPayment(final int payment) {

this.payment = payment;

}

public void setConfirm(final boolean confirm) {

this.confirm = confirm;

}

public int getPayment() {

return payment;

}

public String getWorkingConditions() {

return workingConditions;

}

public String getSpecialty() {

return specialty;

}

public String getFirm() {

return firm;

}

public Recruitment() {

firm = null;

specialty = null;

workingConditions = null;

payment = 0;

needsSpeciality = null;

experience = 0;

education = null;

confirm = false;

}

public String show() {

String show;

show = "Фирма: " + firm + "\n" +

"Специальность: " + specialty + "\n" +

"Условия работы: " + workingConditions + "\n" +

"Оплата: " + payment + "\n";

if (confirm) {

show += "Необходимая специальность: " + needsSpeciality + "\n";

show += "Опыт: " + experience + "\n";

show += "Образование: " + education + "\n";

}

return show;

}

public Recruitment(final Recruitment obj) {

firm = obj.firm;

specialty = obj.specialty;

workingConditions = obj.workingConditions;

payment = obj.payment;

needsSpeciality = obj.needsSpeciality;

experience = obj.experience;

education = obj.education;

confirm = obj.confirm;

}

public void generateVacancy() {

Scanner scan = new Scanner(System.in);

Scanner scan2 = new Scanner(System.in);

int choose;

System.out.print("\nВведите фирму: ");

firm = scan.nextLine();

System.out.print("\nВведите специальность: ");

specialty = scan.nextLine();

System.out.print("\nВведите условия работы: ");

workingConditions = scan.nextLine();

System.out.print("\nВведите оплату: ");

payment = scan.nextInt();

System.out.println("Желаете добавить дополнительные условия работы? 1 - Да. 0 - Нет: ");

choose = scan.nextInt();

while (true) {

if (choose == 1) {

System.out.print("\nВведите необходимую специальность: ");

needsSpeciality = scan2.nextLine();

System.out.print("\nНеобходимое образование: ");

education = scan2.nextLine();

System.out.print("\nеобходимый опыт работы: ");

experience = scan2.nextInt();

confirm = true;

break;

} else if (choose == 0) {

needsSpeciality = null;

experience = 0;

education = null;

break;

} else {

System.out.println("Ошибка! Повторите ввод: ");

}

}

}

public void setExperience(final int experience) {

this.experience = experience;

}

public int getExperience() {

return experience;

}

public void setNeedsSpeciality(final String needsSpeciality) {

this.needsSpeciality = needsSpeciality;

}

public String getNeedsSpeciality() {

return needsSpeciality;

}

public void setEducation(final String education) {

this.education = education;

}

public String getEducation() {

return education;

}

public boolean getConfirms() {

return confirm;

}

@Override

public String toString() {

return show();

}

}

Linked.java:

package ua.khpi.oop.vasilchenko09.MyList;

import java.io.Serializable;

public interface Linked<T> extends DescendingIterator<T>, Serializable, Iterable<T> {

void addLast(T obj);

void addFirst(T obj);

int size();

T getElementByIndex(int index);

void saveAll();

void saveRec();

void add(T obj);

void clear();

boolean notEmpty();

void readRec();

void readAll();

**}**

DescendingIterator.java:

package ua.khpi.oop.vasilchenko09.MyList;

import java.util.Iterator;

public interface DescendingIterator<T> {

Iterator<T> descendingIterator();

}

**ВАРІАНТИ ВИКОРИСТАННЯ**

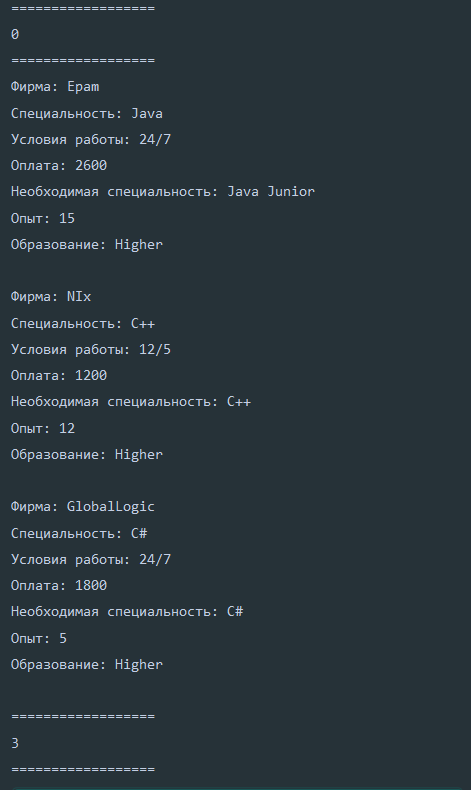
****

Рис. 9.1 – Результат роботи програми

Програму можна використовувати задля створення бази даних. Завдяки параметризації зв’язного списка, базу даних можна використати для будь-яких типів даних. Переважно у нашому варіанті - кадрове агенство, в якому представляються різноманітні вакансії. Також для вибору доступно багато інших можливостей.

**ВИСНОВКИ**

При виконанні лабораторної роботи набуто практичних навичок щодо розробки параметризованих класів. Завдяки цієї можливості в JAVA, можливо створювати колекції та інші класи на основі будь-яких типів. Завдання виконане! Програма працює успішно!