Звіт

Автор: Лунгол О., КІТ101.8б

ЛАБОРАТОРНА РОБОТА №9

МЕТА:

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

ВИМОГИ

1. Створити власний клас-контейнер, що параметризується ([Generic Type](https://docs.oracle.com/javase/tutorial/java/generics/types.html)), на основі [зв’язних списків](https://en.wikipedia.org/wiki/Linked_list) для реалізації колекції domain-об’єктів [лабораторної роботи №7](https://oop-khpi.github.io/#task_07).
2. Для розроблених класів-контейнерів забезпечити можливість використання їх об’єктів у циклі [foreach](https://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html) 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](https://docs.oracle.com/javase/8/docs/technotes/guides/collections/).

ОПИС ПРОГРАМИ

**Опис змінних**

**private** **long** NumberPassport;

**private** String specialty;

**private** String workingConditions;

**private** **int** marks;

**private** **int** payment;

**private** String data;

**private** String position;

**private** String education;

**private** String department;

**Ієрархія та структура класів**

**class** Container- клас в якому описуються та знаходяться контейнеры

**class** Lab09- головний клас

**class** HDR1 – клас в якому данні про відділ кадрів

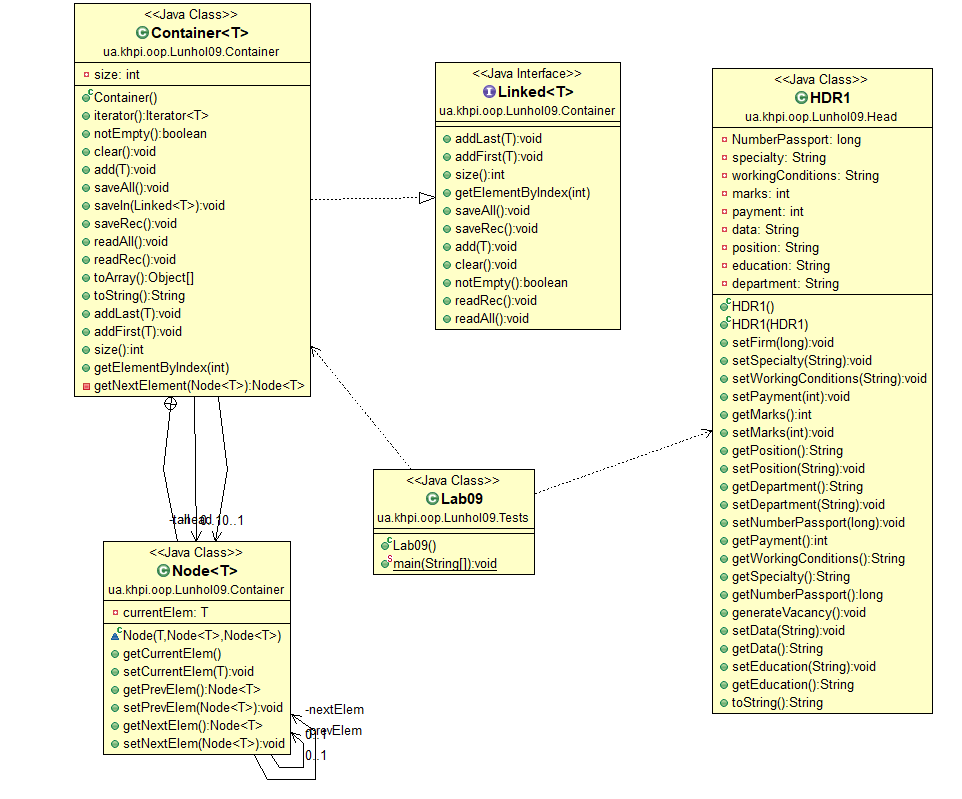


Рисунок 1 – Ієрархія класів

ТЕКСТ ПРОГРАМИ

**package ua.khpi.oop.Lunhol09.Tests;**

**import** java.io.IOException;

**import** ua.khpi.oop.Lunhol09.Container.Container;

**import** ua.khpi.oop.Lunhol09.Head.HDR1;

**public** **class** Lab09 {

**public** **static** **void** main(String[] args) **throws** IOException {

Container<HDR1> container = **new** Container<>();

HDR1 temp = **new** HDR1();

temp.generateVacancy();

System.***out***.println(container.size());

container.add(temp);

System.***out***.println(container.size());

container.readRec();

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

**int** counter = 0;

**for** (HDR1 s : container) {

System.***out***.println("#" + ++counter);

System.***out***.println(s);

}

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

System.***out***.println(container.size());

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

container.saveIn(container);

System.***out***.println(container.getElementByIndex(0));

container.clear();

System.***out***.println(container.size());

}

}

**package ua.khpi.oop.Lunhol09.Container;**

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.ObjectOutputStream;

**import** java.io.PrintWriter;

**import** java.io.Serializable;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.NoSuchElementException;

**import** ua.khpi.oop.Lunhol10.Head.HDR1;

**public** **class** Container<T> **implements** Linked<T>, Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

//конструктор инициализации

**public** Container() {

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<T>() {

**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** **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);

}

}

**public** **void** saveIn(Linked<T> obj) **throws** IOException {

FileOutputStream file = **new** FileOutputStream("save.sasha");

ObjectOutputStream object = **new** ObjectOutputStream(file);

object.writeObject(obj);

object.close();

}

@Override

**public** **void** saveRec() {

**try** {

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

**if** (!file.exists()) {

file.createNewFile();

}

PrintWriter pw = **new** PrintWriter(file);

HDR1 temp;

System.***out***.println();

pw.println(size);

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

temp = (HDR1) getElementByIndex(i);

pw.println(temp.getNumberPassport());

pw.println(temp.getEducation());

pw.println(temp.getPayment());

pw.println(temp.getData());

pw.println(temp.getPosition());

pw.println(temp.getDepartment());

pw.println(temp.getWorkingConditions());

pw.println(temp.getMarks());

}

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() **throws** IOException {

BufferedReader br = **null**;

HDR1 temp = **new** HDR1();

**try** {

br = **new** BufferedReader(**new** FileReader("save.txt"));

String line;

line = br.readLine();

**int** count = Integer.*parseInt*(line);

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

line = br.readLine();

temp.setNumberPassport(Integer.*parseInt*(line));

line = br.readLine();

temp.setEducation(line);

line = br.readLine();

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

line = br.readLine();

temp.setData(line);

line = br.readLine();

temp.setPosition(line);

line = br.readLine();

temp.setDepartment(line);

line = br.readLine();

temp.setWorkingConditions(line);

line = br.readLine();

temp.setMarks(Integer.*parseInt*(line));

add((T) **new** HDR1(temp));

}

} **catch** (IOException ex) {

ex.printStackTrace();

} **finally** {

br.close();

}

}

**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());

}

**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> **implements** Serializable {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**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;

}

}

}

**package ua.khpi.oop.Lunhol09.Container;**

**import** java.io.IOException;

**import** java.io.Serializable;

//создаем интерфейс линкд

//обьявление прототипов функций

// Т джунерик (обобщение)

**public** **interface** Linked<T> **extends** 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() **throws** IOException;

**void** readAll();

}

**package ua.khpi.oop.Lunhol09.Head;**

**import** java.io.Serializable;

**import** java.util.Scanner;

**public** **class** HDR1 **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **long** NumberPassport;

**private** String specialty;

**private** String workingConditions;

**private** **int** marks;

**private** **int** payment;

**private** String data;

**private** String position;

**private** String education;

**private** String department;

**public** HDR1() {

NumberPassport = 0;

specialty = **null**;

workingConditions = **null**;

marks = 0;

payment = 0;

data = **null**;

position = **null**;

education = **null**;

department = **null**;

}

**public** HDR1 (**final** HDR1 obj) {

NumberPassport = obj.NumberPassport;

specialty = obj.specialty;

workingConditions = obj.workingConditions;

marks = obj.marks;

payment = obj.payment;

data = obj.data;

position = obj.position;

education = obj.education;

department = obj.department;

}

**public** **void** setFirm(**long** NumberPassport) {

**this**.NumberPassport = NumberPassport;

}

**public** **void** setSpecialty(String specialty) {

**this**.specialty = specialty;

}

**public** **void** setWorkingConditions(String workingConditions) {

**this**.workingConditions = workingConditions;

}

**public** **void** setPayment(**int** payment) {

**this**.payment = payment;

}

**public** **int** getMarks() {

**return** marks;

}

**public** **void** setMarks(**int** marks) {

**this**.marks = marks;

}

**public** String getPosition() {

**return** position;

}

**public** **void** setPosition(String position) {

**this**.position = position;

}

**public** String getDepartment() {

**return** department;

}

**public** **void** setDepartment(String department) {

**this**.department = department;

}

**public** **void** setNumberPassport(**long** numberPassport) {

NumberPassport = numberPassport;

}

**public** **int** getPayment() {

**return** payment;

}

**public** String getWorkingConditions() {

**return** workingConditions;

}

**public** String getSpecialty() {

**return** specialty;

}

**public** **long** getNumberPassport() {

**return** NumberPassport;

}

**public** **void** generateVacancy() {

Scanner scan = **new** Scanner(System.***in***);

Scanner scan2 = **new** Scanner(System.***in***);

**int** choose = 0;

System.***out***.print("\nВведите номер паспорта: ");

NumberPassport = scan.nextLong();

System.***out***.print("\nВведите образование: ");

education = scan2.nextLine();

System.***out***.print("\nОплата: ");

payment = scan.nextInt();

System.***out***.print("\nДата взятия на должность: ");

data = scan2.nextLine();

System.***out***.print("\nВведите должность: ");

position = scan2.nextLine();

System.***out***.print("\nВведите отдел: ");

department = scan2.nextLine();

System.***out***.print("\nНабор свойств: ");

workingConditions = scan2.nextLine();

System.***out***.println("\nОценка кандидата: ");

marks = scan.nextInt();

}

**public** **void** setData(String data) {

**this**.data = data;

}

**public** String getData() {

**return** data;

}

**public** **void** setEducation(String education) {

**this**.education = education;

}

**public** String getEducation() {

**return** education;

}

@Override

**public** String toString() {

**return** "NumberPassport = " + NumberPassport + "\n" +"specialty = " + specialty +"\n" + "workingConditions = "

+ workingConditions + "\n" +"marks = " + marks + "\n" +"payment = " + payment + "\n" + "data = " + data + "\n" + "position = "

+ position + "\n" +"education = " + education + "\n" +"department =" + department;

}

}

РОБОТА ПРОГРАМИ

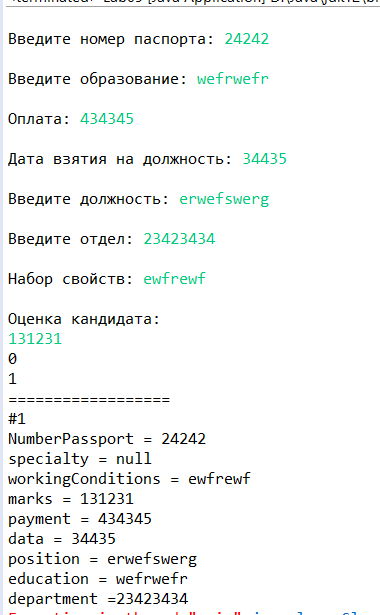


Рисунок 2

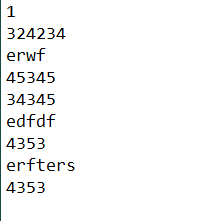


Рисунок 3- робота з файлами

Висновок

Створили власний клас-контейнер, що параметризується ([Generic Type](https://docs.oracle.com/javase/tutorial/java/generics/types.html)), на основі [зв’язних списків](https://en.wikipedia.org/wiki/Linked_list) для реалізації колекції domain-об’єктів [лабораторної роботи №7](https://oop-khpi.github.io/" \l "task_07).Для розроблених класів-контейнерів забезпечили можливість використання їх об’єктів у циклі [foreach](https://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html) в якості джерела даних.Забезпечили можливість збереження та відновлення колекції об’єктів: 1) за допомогою стандартної серіалізації; 2) не використовуючи протокол серіалізації.Продемонстрували розроблену функціональність: створення контейнера, додавання елементів, видалення елементів, очищення контейнера, перетворення у масив, перетворення у рядок, перевірку на наявність елементів.