1

Автор:Горянський І.,1.КІТ101.8б

Лабораторна робота № 9

ПАРАМЕТРИЗАЦІЯ В JAVA

**1. ЗАВДАННЯ ДО РОБОТИ**

**Загальне завдання:**

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/).

**1.1 Опис змінних:**

**private** T currElem; поточний елемент списку

**private** Node<T> prevElem; попередній елемент списку

**private** Node<T> nextElem; наступний елемент списку

**private** Node<T> fstNode;

**private** Node<T> lstNode;

**private** **int** size = 0; змінна розміру списку

Текст програми:

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.IOException;

**import** java.text.ParseException;

**public** **class** Main {

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

LinkedContainer<String>stringLinked1 = **new** LinkedContainer<>();

LinkedContainer<Directory>dirList = **new** LinkedContainer<>();

Directory dir = **new** Directory();

System.***out***.println("Добавляем элемеент");

stringLinked1.addLast("faf");

**for**(String s : stringLinked1) {

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

}

System.***out***.println("Выводим размер");

System.***out***.println(stringLinked1.size());

System.***out***.println("Очистка контейнера");

stringLinked1.clear();

System.***out***.println(stringLinked1.size());

**for**(String s : stringLinked1) {

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

}

System.***out***.println("Запись с файла");

// stringLinked1.readAll();

**for**(String s : stringLinked1) {

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

}

System.***out***.println("Удаляем элемент");

stringLinked1.DelAll();

System.***out***.println(stringLinked1.size());

System.***out***.println("Добавляем ноый элемент");

stringLinked1.addLast("faf");

stringLinked1.addLast("ffad");

stringLinked1.addLast("bvca");

stringLinked1.addLast("fafafa");

**for**(String s : stringLinked1) {

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

}

System.***out***.println("Выводим в файл");

//stringLinked1.toFileAll();

System.***out***.println("Проверяем на наявность элементов:");

stringLinked1.checkEmptiness();

System.***out***.println("Генерируем данные для прикладной задачи:");

stringLinked1.clear();

//dir.generateStore();

//dirList.add(dir);

dirList.FromFile();

**for**(Directory s : dirList) {

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

}

//dirList.toFile();

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.ParseException;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.NoSuchElementException;

**public** **class** LinkedContainer<T> **implements** Linked<T>, Iterable<T>, DescendingIterator<T> {

// конструктор инициализации

**private** **class** Node<T> {

**private** T currElem; // текущий

**private** Node<T> prevElem; // предыдущий

**private** Node<T> nextElem; // следущий

// конструктор копирования

Node(T currElem, Node<T> prevElem, Node<T> nextElem) {

**this**.currElem = currElem;

**this**.prevElem = prevElem;

**this**.nextElem = nextElem;

}

// геттеры и сеттеры

**public** T getCurrentElem() {

**return** currElem;

}

**public** **void** setCurrentElem(T currentElem) {

**this**.currElem = currentElem;

}

**public** Node<T> getPrevElem() {

**return** prevElem;

}

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

**this**.prevElem = prevElem;

}

**public** Node<T> getNextElem() {

**return** nextElem;

}

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

**this**.nextElem = nextElem;

}

}

**private** Node<T> fstNode;

**private** Node<T> lstNode;

**private** **int** size = 0;

**public** LinkedContainer() {

fstNode = **new** Node<T>(**null**, **null**, lstNode);

lstNode = **new** Node<T>(**null**, fstNode, **null**);

fstNode = **new** Node<T>(**null**, **null**, lstNode);

}

@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** Iterator<T> descendingIterator() {

**return** **new** Iterator<T>() {

**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** **void** clear() {

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

Node<T> next = x.nextElem;

x.currElem = **null**;

x.nextElem = **null**;

x.prevElem = **null**;

x = next;

}

fstNode = **null**;

lstNode = **null**;

lstNode = **new** Node<>(**null**, fstNode, **null**);

fstNode = **new** Node<>(**null**, **null**, lstNode);

size = 0;

}

@Override

**public** **void** add(**final** T t) {

addLast(t);

}

@Override

**public** **void** toFile() {

**try** {

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

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

file.createNewFile();

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

Directory temp;

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

pw.println(size);

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

temp = (Directory) getElementByIndex(i);

pw.println(temp.getName());

pw.println(temp.getAddress());

pw.println(temp.getSpecial());

pw.println(temp.getPhone());

pw.println(temp.getTime());

pw.println(temp.getTime1());

}

pw.close();

} **catch** (IOException e) {

System.***out***.print("Error: " + e);

}

}

@Override

**public** **void** readAll() {

**try** (BufferedReader br = **new** BufferedReader(

**new** FileReader("C:\\Users\\User\\eclipse-workspace\\laba 9\\File.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** FromFile() **throws** ParseException {

BufferedReader br = **null**;

Directory temp = **new** Directory();

**try** {

br = **new** BufferedReader(**new** FileReader("C:\\Users\\User\\eclipse-workspace\\laba 9\\File.txt"));

String Line;

Line = br.readLine();

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

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

Line = br.readLine();

temp.setName(Line);

Line = br.readLine();

temp.setAddress(Line);

Line = br.readLine();

temp.setSpecial(Line);

Line = br.readLine();

temp.setPhone(Line);

Line = br.readLine();

temp.setTime(Line);

Line = br.readLine();

temp.setTime1(Line);

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

}

} **catch** (IOException e) {

System.***out***.print("Error: " + e);

} **finally** {

**try** {

br.close();

} **catch** (IOException e) {

System.***out***.print("Error: " + e);

}

}

}

**public** Object[] toArray() {

Object[] result = **new** Object[size];

**int** i = 0;

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

result[i++] = temp.currElem;

}

**return** result;

}

**public** String toString() {

**return** Arrays.*toString*(toArray());

}

@Override

**public** **void** addLast(**final** T obj) {

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

prev.setCurrentElem(obj); // установка значения

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

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

size++; // увелечение размера списка

}

@Override

**public** **void** addFirst(**final** T obj) {

Node<T> next = fstNode;

next.setCurrentElem(obj);

fstNode = **new** Node<>(**null**, **null**, next);

next.setPrevElem(fstNode);

size++;

}

@Override

**public** **int** size() {

**return** size;

}

@Override

**public** T getElementByIndex(**final** **int** index) {

Node<T> target = fstNode.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();

}

@Override

**public** **void** DelAll() {

fstNode = **null**;

lstNode = **null**;

lstNode = **new** Node<>(**null**, fstNode, **null**);

fstNode = **new** Node<>(**null**, **null**, lstNode);

size = 0;

}

@Override

**public** **void** toFileAll() {

**try** {

File file = **new** File("C:\\Users\\User\\eclipse-workspace\\laba 9\\File1.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***.print("Error: " + e);

}

}

@Override

**public** **void** checkEmptiness() {

**if** (size() == 0) {

System.***out***.println("Conteiner is empty");

} **else** **if** (size() > 0) {

System.***out***.println("Conteiner is not empty");

}

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.ParseException;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.NoSuchElementException;

**public** **class** LinkedContainer<T> **implements** Linked<T>, Iterable<T>, DescendingIterator<T> {

// конструктор инициализации

**private** **class** Node<T> {

**private** T currElem; // текущий

**private** Node<T> prevElem; // предыдущий

**private** Node<T> nextElem; // следущий

// конструктор копирования

Node(T currElem, Node<T> prevElem, Node<T> nextElem) {

**this**.currElem = currElem;

**this**.prevElem = prevElem;

**this**.nextElem = nextElem;

}

// геттеры и сеттеры

**public** T getCurrentElem() {

**return** currElem;

}

**public** **void** setCurrentElem(T currentElem) {

**this**.currElem = currentElem;

}

**public** Node<T> getPrevElem() {

**return** prevElem;

}

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

**this**.prevElem = prevElem;

}

**public** Node<T> getNextElem() {

**return** nextElem;

}

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

**this**.nextElem = nextElem;

}

}

**private** Node<T> fstNode;

**private** Node<T> lstNode;

**private** **int** size = 0;

**public** LinkedContainer() {

fstNode = **new** Node<T>(**null**, **null**, lstNode);

lstNode = **new** Node<T>(**null**, fstNode, **null**);

fstNode = **new** Node<T>(**null**, **null**, lstNode);

}

@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** Iterator<T> descendingIterator() {

**return** **new** Iterator<T>() {

**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** **void** clear() {

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

Node<T> next = x.nextElem;

x.currElem = **null**;

x.nextElem = **null**;

x.prevElem = **null**;

x = next;

}

fstNode = **null**;

lstNode = **null**;

lstNode = **new** Node<>(**null**, fstNode, **null**);

fstNode = **new** Node<>(**null**, **null**, lstNode);

size = 0;

}

@Override

**public** **void** add(**final** T t) {

addLast(t);

}

@Override

**public** **void** toFile() {

**try** {

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

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

file.createNewFile();

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

Directory temp;

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

pw.println(size);

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

temp = (Directory) getElementByIndex(i);

pw.println(temp.getName());

pw.println(temp.getAddress());

pw.println(temp.getSpecial());

pw.println(temp.getPhone());

pw.println(temp.getTime());

pw.println(temp.getTime1());

}

pw.close();

} **catch** (IOException e) {

System.***out***.print("Error: " + e);

}

}

@Override

**public** **void** readAll() {

**try** (BufferedReader br = **new** BufferedReader(

**new** FileReader("C:\\Users\\User\\eclipse-workspace\\laba 9\\File.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** FromFile() **throws** ParseException {

BufferedReader br = **null**;

Directory temp = **new** Directory();

**try** {

br = **new** BufferedReader(**new** FileReader("C:\\Users\\User\\eclipse-workspace\\laba 9\\File.txt"));

String Line;

Line = br.readLine();

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

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

Line = br.readLine();

temp.setName(Line);

Line = br.readLine();

temp.setAddress(Line);

Line = br.readLine();

temp.setSpecial(Line);

Line = br.readLine();

temp.setPhone(Line);

Line = br.readLine();

temp.setTime(Line);

Line = br.readLine();

temp.setTime1(Line);

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

}

} **catch** (IOException e) {

System.***out***.print("Error: " + e);

} **finally** {

**try** {

br.close();

} **catch** (IOException e) {

System.***out***.print("Error: " + e);

}

}

}

**public** Object[] toArray() {

Object[] result = **new** Object[size];

**int** i = 0;

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

result[i++] = temp.currElem;

}

**return** result;

}

**public** String toString() {

**return** Arrays.*toString*(toArray());

}

@Override

**public** **void** addLast(**final** T obj) {

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

prev.setCurrentElem(obj); // установка значения

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

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

size++; // увелечение размера списка

}

@Override

**public** **void** addFirst(**final** T obj) {

Node<T> next = fstNode;

next.setCurrentElem(obj);

fstNode = **new** Node<>(**null**, **null**, next);

next.setPrevElem(fstNode);

size++;

}

@Override

**public** **int** size() {

**return** size;

}

@Override

**public** T getElementByIndex(**final** **int** index) {

Node<T> target = fstNode.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();

}

@Override

**public** **void** DelAll() {

fstNode = **null**;

lstNode = **null**;

lstNode = **new** Node<>(**null**, fstNode, **null**);

fstNode = **new** Node<>(**null**, **null**, lstNode);

size = 0;

}

@Override

**public** **void** toFileAll() {

**try** {

File file = **new** File("C:\\Users\\User\\eclipse-workspace\\laba 9\\File1.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***.print("Error: " + e);

}

}

@Override

**public** **void** checkEmptiness() {

**if** (size() == 0) {

System.***out***.println("Conteiner is empty");

} **else** **if** (size() > 0) {

System.***out***.println("Conteiner is not empty");

}

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.util.Scanner;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**import** java.text.ParseException;

**import** java.io.IOException;

**import** java.util.Date;

**import** java.text.SimpleDateFormat;

**public** **class** Directory {

**private** String Address;

**private** String Name;

**private** String Special;

**private** String Phone;

**private** String StartTime;

**private** String EndTime;

String str1;

String str2;

Directory(){

Address = **null**;

Name = **null**;

Special = **null**;

Phone = **null**;

StartTime = **null**;

EndTime = **null**;

}

@Override

**public** String toString() {

**return** "Address=" + Address + "\nName=" + Name + "\nSpecial=" + Special + "\nPhone=" + Phone

+ "\nStartTime=" + StartTime + "\nEndTime=" + EndTime;

}

Directory(**final** Directory obj){

Address = obj.Address;

Name = obj.Name;

Special = obj.Special;

Phone = obj.Phone;

StartTime = obj.StartTime;

EndTime = obj.EndTime;

}

**public** **void** setAddress(String Address) {

**this**.Address =Address;

}

**public** **void** setName(String Name) {

**this**.Name = Name;

}

**public** **void** setSpecial(String Special) {

**this**.Special = Special;

}

**public** **void** setPhone(String Phone) {

**this**.Phone = Phone;

}

**public** **void** setTime(String Time) {

**this**.StartTime = Time;

}

**public** **void** setTime1(String Time1) {

**this**.EndTime = Time1;

}

String getAddress() {

**return** Address;

}

String getName() {

**return** Name;

}

String getSpecial() {

**return** Special;

}

String getPhone() {

**return** Phone;

}

**public** String getTime() {

// SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

// return sdf.format(this.StartTime);

**return** StartTime;

}

**public** String getTime1() {

// SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

// return sdf.format(this.EndTime);

**return** EndTime;

}

**public** **void** generateStore() **throws** ParseException,IOException {

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

System.***out***.println("Enter name of outlet");

Name = s.nextLine();

System.***out***.println("Enter Address");

Address = s.nextLine();

System.***out***.println("Enter Specialization");

Special = s.nextLine();

Pattern pattern = Pattern.*compile*("\\d{10,12}");

String str = **null**;

Matcher m = **null**;

**do** {

System.***out***.println("Enter Phone number");

str = s.nextLine();

m = pattern.matcher(str);

}**while**(!m.matches());

Phone = str;

Pattern pattern1 = Pattern.*compile*("[0-9]{2}:[0-9]{2}");

String str1 = **null**;

Matcher mat1 = **null**;

**do** {

System.***out***.println("Enter StartTime:");

str1 =s.nextLine();

mat1 = pattern1.matcher(str1);

}**while**(!mat1.matches());

StartTime = str1;

Pattern pattern2 = Pattern.*compile*("[0-9]{2}:[0-9]{2}");

String str2 = **null**;

Matcher mat2 = **null**;

**do** {

System.***out***.println("Enter StartTime:");

str2 =s.nextLine();

mat2 = pattern2.matcher(str1);

}**while**(!mat2.matches());

EndTime = str2;

}

}

**package** ua.khpi.oop.gorianckiy09;

**import** java.util.Iterator;

**public** **interface** DescendingIterator<T> {

Iterator<T> descendingIterator();

}

**2. Діаграми**

3.Результат роботи:

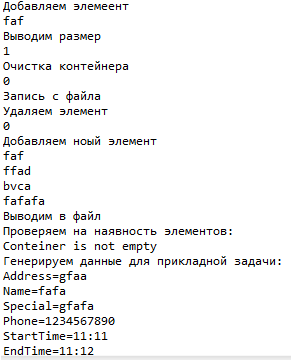


Рисунок 1 – результат виконання програми.

ВИСНОВОК

При виконанні цієї лабораторної роботи були набуті навички розробки свого власного контейнеру.