Тема 9. Параметризація в Java∗

Мета:

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

**1 ВИМОГИ**

**1.1 Розробник**

Інформація про розробника:

- Гряник Георгій Володимирович

- КІТ-119Д;

- 6 варіант.

**1.2 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.gitlab.io/%D0%B7%D0%B0%D0%B2%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F/task07/).
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/).

1.3 Задача

### **Поліцейська картотека**

Дані про злочинця: П.І.Б.; дата народження; дата судимостей ; дата останнього позбавлення волі; дата останнього звільнення.

2 ОПИС ПРОГРАМИ

2.1 Засоби ООПРозробка класу Серіалізаціі/десеріалізаціі, клас контейнер-список, конструктори типів даних створенi користувачем, helper-клас, клас-файл.  
2.2 Ієрархія та структура класівКлас “PoliceFile ” – описує поліцейську картотеку з можливістю додавати та виводити дані класу. Клас описує дані про злочинця відповідно до завданн. Клас «Date” – опису формат часу :день, місяць, рік . Створений для ергономічного запису дат відомостей про злочинця. Клас « Console\_program» - клас керування програми , створений щоб надавати користувачеві можливість керувати програмою. Клас «СontainerList» - клас-контейнер створений для зберігання даних у список. Реалізовано додавання, видалення та інші можливості для керування даними. Клас «Serializator» - клас розроблений для збереження даних контейнеру у файл. При цьому зберігання прохоже у звичайний файл та файл типу .xml. При цьому в класі реалізовано методи для відновлення даних як із звичайного файлу так із .xml файлу. Клас Console\_File розроблений для роботи із файлами розміщені в директоріях. Цей клас забезпечує можливість користувачеві обирати файл та перемінятися між директоріями. Клас «Helper» - реалізація допоміжних методів які реалізують допоміжні дії в основній програмі.

2.3 Важливі фрагменти програми **/\*\***

**\* @author <Георгiй>**

**\***

**\*/**

**import** java.io.Serializable;

**public** **class** Element<T **extends** Serializable> {

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

**public** T num;

**public** Element next;

**public** Element(T n) {

num = n;

next = **null**;

}

**public** Element() {}

}

////////////////////////////////////////////////////////////////////

**import** java.io.Serializable;

**import** java.lang.reflect.Array;

**import** java.util.Optional;

**import** java.util.Scanner;

**import** java.util.Arrays;

**import** java.util.Iterator;

**import** java.util.NoSuchElementException;

**public** **class** ContainerList<T **extends** Serializable> **implements** List<T>,Iterable<T> {

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

**public** Element<T> top = **null** ;

**public** Element<T> current = **null**;

**public** Element<T> last = **null**;

**public** Element<T> previous = **null**;

///////////////////////////////////////////////////////

**public** ContainerList() { }

///////////////////////////////////////////////////

**public** ContainerList(T tab[]) {

**for** (T a : tab) {

**this**.add(a);

}

}

////////////////////////////////////////////////////

@Override

**public** **void** addOnBack(T n) {

Optional<T> optional = Optional.*empty*();

optional = Optional.*ofNullable*(n);

**if** (optional.isPresent()) {

current = **new** Element<>(n);

current.next = top;

top = current;

}

}

///////////////////////////////////////////////////////

@Override

**public** **void** add(T n) {

Optional<T> optional = Optional.*empty*();

optional = Optional.*ofNullable*(n);

**while** (optional.isPresent()) {

current = **new** Element<>(n);

**if** (top == **null**) {

top = current;

} **else** {

last.next = current;

}

last = current;

**break**;

}

}

////////////////////////////////////////////////////////

@Override

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

**int** size = 0;

current = top;

**while** (current != **null**) {

size++;

current = current.next;

}

**return** size;

}

/////////////////////////////////////////////////////////

@Override

**public** T getNode(**int** n) {

**int** size = **this**.size();

Element<T> searchingNumber = top;

Element<T> target = **null**;

T t = **null**;

**if** (n <= size() && searchingNumber != **null**) {

**for** (**int** a = 0; a < n; a++) {

target = searchingNumber;

searchingNumber = searchingNumber.next;

}

} **else** {

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

}

**if** (target != **null**) {

t = target.num;

} **else** {

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

}

**return** t;

}

//////////////////////////////////////////////////////////

@Override

**public** **void** removeNode(**int** n) {

**if** (n <= 0 || n > size()) {

System.***out***.println("Список пустий");

} **else** {

**if** (n == 1) {

top = top.next;

}

current = **null**;

current = top;

**if** (current != **null**) {

**for** (**int** a = 1; a < n; a++) {

previous = current;

current = current.next;

}

}

**if** (n > 1) {

previous.next = current.next;

}

}

}

///////////////////////////////////////////////////////////

@Override

**public** **void** remove()

{

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

{

System.***out***.println("Список пустий");

**return**;

}

top=**null**;

}

////////////////////////////////

**public** String toString()

{

String s=**new** String();

current=top;

**while** (current != **null**) {

s+=current.num.toString();

current = current.next;

}

**return** s;

}

////////////////////////////////

**public** **void** Substitute(T el,**int** index)

{

current=top;

**for** (**int** i=1;i<index;i++)

{

current=current.next;

}

current.num=el;

}

////////////////////////////////

@SuppressWarnings({"unchecked"})

**public** Object[] СonverToArray()

{

**int** size=**this**.size();

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

current=top;

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

{

mas[i]=(T)current.num;

current=current.next;

}

**return** mas;

}

/////////////////////////////////

// Iterator

**public** Iterator<T> iterator() {

**return** **new** MyLinkedListIterator();

}

**private** **class** MyLinkedListIterator **implements** Iterator<T> {

**private** Element<T> curr;

**public** MyLinkedListIterator() {

**this**.curr = top; // голова списка

}

**public** **boolean** hasNext() {

**return** **this**.curr != **null**;

}

**public** T next() {

**if** (!**this**.hasNext()) {

**throw** **new** NoSuchElementException();

}

T value = curr.num; // значение в текущем узле

curr = curr.next; // следующий узел

**return** value;

}

**public** **void** remove() {

**throw** **new** UnsupportedOperationException();

}

}

}////////////////////////////////////////////////////////////////////

**public** **class** Console\_File {

**public** **static** Scanner *in* = **new** Scanner(System.***in***);

**public** **static** **int** dialogOut() **throws** IOException, InterruptedException

{

//for (int i = 0; i < 50; ++i) System.out.println();

System.***out***.print("\n\n Оберіть команду:"

+"\n\*1 - Місце знаходження"

+"\n\*2 - Файли в директорії"

+"\n\*3 - Вийти із директорії"

+"\n\*4 - Вибрати файл"

+"\n\*5 - Створити директорію "

+"\n\*6 - Перейти за адресою: "

+"\n\*7 - Save файл"

+"\n\n ваша команда: ");

**return** *in*.nextInt();

}

**public** **static** File MenuFillOut() ///функціє проводить координування по можливостям програм

{

File file= **new** File("C:/Users/дом/Documents");

**while**(**true**)///нескінченний цикл який дозволяє працювати програмі

{

**try** {

**int** k=*dialogOut*();

**switch**(k)///пошук введеної команди

{

**case** 1:**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

System.***out***.println("\n Шлях: "+file.getPath()); //getAbsolutePath

**break**;

**case** 2:

{

**int** a=0;

**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

System.***out***.print("Файли:"+file.getPath()+"\n");

**for** (File file2 : file.listFiles())

{

**if** (a%5==0) System.***out***.print("\n");a++;

System.***out***.printf("%-25s ",file2.getName());

} ;///////////

}

**break**;

**case** 3:**for** (**int** i = 0; i < 50; ++i) System.***out***.println();file=file.getParentFile();

**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

**break**;

**case** 4:

**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

{System.***out***.print("Ведіть назву файлу:");

String s = *in*.nextLine();

file=**new** File(file.getAbsolutePath()+"/"+*in*.nextLine());

}

**if** (file.isFile()==**true**)**return** file;

**if**(file.isDirectory()==**true**);

**else** {

System.***out***.print("незнайдено");

file=file.getParentFile();

}

**break**;

**case** 5:

{**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

System.***out***.print("Ведіть назву директрії: ");

String s = *in*.nextLine();

File dir=**new** File(file.getPath()+"/"+*in*.nextLine());

System.***out***.println("Створення директорії: "+dir.mkdirs());

}

**break**;

**case** 6:

{**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

{

System.***out***.print("Ведіть адресу: ");

String s = *in*.nextLine();

s = *in*.nextLine();

String s2=**new** String();

**for** (**int** i=0;i<s.length();i++)

{

**if** (s.charAt(i)=='\\') s2+="/";

**else** s2+=s.charAt(i);

}

file=**new** File(s2);

}

}

**break**;

**case** 7:

{**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

System.***out***.print("Ведіть назву файлу:");

String s = *in*.nextLine();

**return** **new** File(file.getAbsolutePath()+"/"+*in*.nextLine());

}

}

}**catch**(Exception e) { **for** (**int** i=0;i<10;i++)System.***out***.println(" EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR ");

System.***out***.print("\n\n\nТрапилась помилка. Але тепер все добре!!\n\n");

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

}

}

**public** **static** **int** dialogIn() **throws** IOException, InterruptedException

{

//for (int i = 0; i < 50; ++i) System.out.println();

System.***out***.print("\n\n Оберіть команду:"

+"\n\*1 - Місце знаходження"

+"\n\*2 - Файли в директорії"

+"\n\*3 - Перейти в директорію"

+"\n\*4 - Вийти із директорії"

+"\n\*5 - Відкрити файл"

+"\n\*6 - Перейти за адресою: "

+"\n\n ваша команда: ");

**return** *in*.nextInt();

}

**public** **static** File MenuFillIn() ///функціє проводить координування по можливостям програм

{

File file= **new** File("C:/Users/дом/Documents");

**while**(**true**)///нескінченний цикл який дозволяє працювати програмі

{

**try** {

**int** k=*dialogIn*();

**switch**(k)///пошук введеної команди

{

**case** 1:

**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

System.***out***.println("\n Шлях: "+file.getPath()); //getAbsolutePath

**break**;

**case** 2:

{ **for** (**int** i = 0; i < 50; ++i) System.***out***.println();

**int** a=0;

System.***out***.print("Файли:"+file.getPath()+"\n");

**for** (File file2 : file.listFiles())

{

**if** (a%4==0) System.***out***.print("\n");a++;

System.***out***.printf("%-25s ",file2.getName());

} ///////////

}

**break**;

**case** 3:

{**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

System.***out***.print("Ведіть назву файлу:");

String s = *in*.nextLine();

file=**new** File(file.getAbsolutePath()+"/"+*in*.nextLine());

}

**if** (file.isDirectory()==**false**)

{

System.***out***.print("Дирикторія незнайдена");

file=file.getParentFile();

}

**break**;

**case** 4:

**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

file=file.getParentFile();

**break**;

**case** 5:

{**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

System.***out***.print("Ведіть назву файлу:");

String s = *in*.nextLine();

file=**new** File(file.getAbsolutePath()+"/"+*in*.nextLine());

}

**if** (file.isFile()==**true**)**return** file;

**break**;

**case** 6:

{**for** (**int** i = 0; i < 50; ++i) System.***out***.println();

{

System.***out***.print("Ведіть адресу: ");

String s = *in*.nextLine();

s = *in*.nextLine();

String s2=**new** String();

**for** (**int** i=0;i<s.length();i++)

{

**if** (s.charAt(i)=='\\') s2+="/";

**else** s2+=s.charAt(i);

}

file=**new** File(s2);

}

}

**break**;

}

}**catch**(Exception e) { **for** (**int** i=0;i<10;i++)System.***out***.println(" EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR ");

System.***out***.print("\n\n\nТрапилась помилка. Але тепер все добре!!\n\n");

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

}

}

} **3 ВАРІАНТИ ВИКОРИСТАННЯ**

Створена програма працює за принципом додавання редагування видалення. Тобто програма може додати дати ваші дані про злочинця (ім’я, прізвище, по-батькові) та дати через які він потрапив у цю базу. Також програма може вивести ці дані на екран. Алгоритм додавання реалізований за алгоритмом додавання одно направленого списку. При додаванні даних програма просить користувача ввести певні дані. Якщо ви десь допустили помилку то можна завжди очистити список або відредагувати дані якогось злочинця. Алгоритм редагування подібний до додаванням за винятком того, що замість створення нового об’єкту береться готовий об’єкт та змінюється дані полів/поля.

Дата реалізована окремим класом задля ергономічності програми та кращого контролювання дати. Тобто створено клас який містить ці дані і створюється поле типу дата та записується відповідні значення.

Для зручної роботи розроблено метод зберігання та відновлення даних не тільки у форматі файлу, а й у форматі xml файлу. При цьому можна злегкістю відновити збереженні дані.

Для реалізації дій меню розроблено “Helper” яке просто реалізовує методи по типу способу додавання даних та виводу.

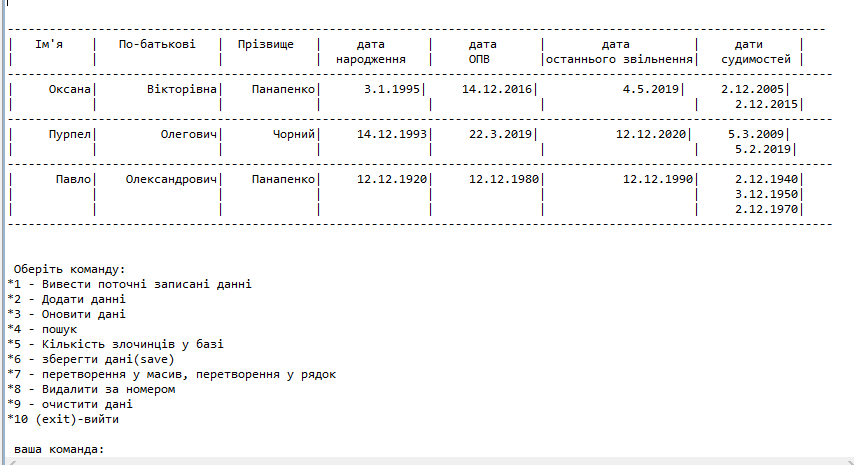


Рисунок 1. Вивод даних програми та меню

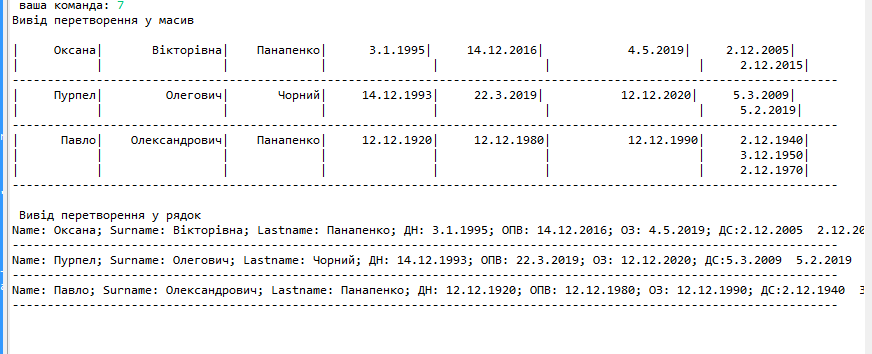
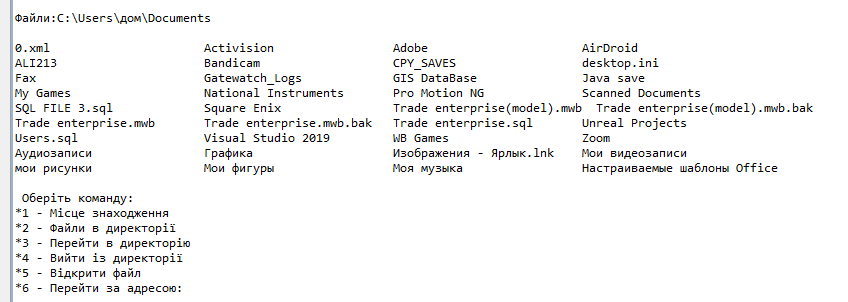


Рисунок 2 – меню відкриття файлу та список файлів у директорії

3. Вивод даних перетворених у масив та рядок

ВИСНОВКИ

Під час виконання лабораторної роботи було набуто навички роботи управління введенням/виведенням даних з використанням класів платформи Java SE та реалізації власного контейнеру списку.