Тема 10. Обробка параметризованих контейнерів∗

Мета:

* Розширення функціональності параметризованих класів.

**1 ВИМОГИ**

**1.1 Розробник**

Інформація про розробника:

- Гряник Георгій Володимиович

- КІТ-119Д;

- 6 варіант.

**1.2 Загальне завдання**

Використовуючи програму рішення завдання [лабораторної роботи №9](https://oop-khpi.gitlab.io/%D0%B7%D0%B0%D0%B2%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F/task09/):

1. Розробити параметризовані методи ([Generic Methods](https://docs.oracle.com/javase/tutorial/java/generics/methods.html)) для обробки колекцій об'єктів згідно [прикладної задачі](https://oop-khpi.gitlab.io/%D0%B7%D0%B0%D0%B2%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F/task10/#_3).
2. Продемонструвати розроблену функціональність (створення, управління та обробку власних контейнерів) в діалоговому та автоматичному режимах.
   * Автоматичний режим виконання програми задається параметром командного рядка **-auto**. Наприклад, java ClassName -auto.
   * В автоматичному режимі діалог з користувачем відсутній, необхідні данні генеруються, або зчитуються з файлу.
3. Забороняється використання алгоритмів з [Java Collections Framework](https://docs.oracle.com/javase/8/docs/technotes/guides/collections/).

1.3 Задача

### **Поліцейська картотека**

Поліцейська картотека. Сортування за прізвищем, за датою народження, за датою останнього позбавлення волі, за датою останнього звільнення.

2 ОПИС ПРОГРАМИ

2.1 Засоби ООПРозробити параметризовані методи (Generic Methods) для обробки колекцій , автоматизація процесу за бажанням користувача

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. Клас «Console\_program\_auto» - клас розроблений для автоматизації програмию.

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

**\* @author <Георгiй>**

**\***

**\*/**

**public** **class** Helper {

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

**public** **static** ContainerList<PoliceFile> WaysToAddData(ContainerList<PoliceFile> A) /// Спосб додавання елементів

{

**int** a;

System.***out***.print("\n 1: З клавiатири\n 2: 3 Файлу\n Ваш вiбiр: ");

a=*in*.nextInt();

**switch** (a)

{

**case** 1:

*add\_from\_keyboard*(A);

**break**;

**case** 2:

A=Console\_program.*serializator*.deserializtionXML();

**break**;

}

**return** A;

}

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

**private** **static** ContainerList<PoliceFile> add\_from\_keyboard(ContainerList<PoliceFile> A )

{

**int** a;

System.***out***.print("\n Скільки ви плануєте додати в'язнів: ");

**int** n=*in*.nextInt();

System.***out***.print("\n Додати:\n 1: В початок \n 2: В кінець \n 3: ? \nВаш вiбiр: ");

a=*in*.nextInt();

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

{

PoliceFile add\_from\_keyboard=**new** PoliceFile();

add\_from\_keyboard.EL(add\_from\_keyboard,**false**);

**switch** (a)

{

**case** 1:

A.addOnBack(add\_from\_keyboard);

**break**;

**case** 2:

A.add(add\_from\_keyboard);

**break**;

}System.***out***.print("\n!записанo!\n");

} System.***out***.print("\nВашi данi успiшно записанi\n");

**return** A;

}

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

**public** **static** **void** show(ContainerList<PoliceFile> List)

{

PoliceFile PF=**new** PoliceFile();

PF.printTableHead();

**for** (**var** L:List) {

L.show();

}

}

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

**public** **static** **void** posuk(ContainerList<PoliceFile> List)//// пошуку

{

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

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

**return** ;

}

String text=**new** String();

System.***out***.print("Шукати:\n"+

" 1. Імя\n"+

" 2. по-Батькові\n"+

" 3. Прізвище\n"+

" 4. Index\n"+

"in: ");

**int** a=*in*.nextInt();

**int** i=0;

**switch**(a)

{

**case** 1:

System.***out***.print("Введіть Імя : ");

text=*in*.next();

**for** (**var** L:List)

{

**if** (L.getName().equals(text))

{

System.***out***.print("Позиція вашаго елемента: "+i+"\n");

L.printTableHead();

L.show();

**return** ;

}i++;

}

System.***out***.print("Ваш елемент незнайденo");

**break**;

**case** 2:

System.***out***.print("Введіть по-Батькові : ");

text=*in*.next();

**for** (**var** L:List)

{

**if** (L.getSurname().equals(text))

{

System.***out***.print("Позиція вашаго елемента: "+i+"\n");

L.printTableHead();

L.show();

**return** ;

}i++;

}

System.***out***.print("Ваш елемент незнайденo");

**break**;

**case** 3:

System.***out***.print("Введіть прізвище : ");

text=*in*.next();

**for** (**var** L:List)

{

**if** (L.getLastname().equals(text))

{

System.***out***.print("Позиція вашаго елемента: "+i+"\n");

L.printTableHead();

L.show();

**return** ;

}i++;

}

System.***out***.print("Ваш елемент незнайденo");

**break**;

**case** 4:

System.***out***.print("Введіть index : ");

a=*in*.nextInt();

**try**

{

System.***out***.print("\n\n");

List.getNode(a).show();

}

**catch**(Exception e) { System.***out***.print("Ваш елемент незнайденo");}

**break**;

}

}

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

**public** **static** **void** UpdateData(ContainerList<PoliceFile> List ){

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

**if** (size==0)

{

System.***out***.print("\n\n Список пустий\n\n");

**return**;

}

System.***out***.print("Введть індекс: ");

**int** index=*in*.nextInt();

**if**(index<1||index>size)

{

System.***out***.print("\nТакого злочинця незнайдено\n");

**return**;

}

PoliceFile PF=List.getNode(index);

PF.show();

List.Substitute(PF.EL(PF,**true**), index);

}

**public** **static** **void** romove(ContainerList<PoliceFile> List) {

**if** (List.size()==0)

{

System.***out***.print("\n\n Список пустий\n\n");

**return**;

}

System.***out***.print("\nВведіть номер плануєе видалити?: ");

**int** n=*in*.nextInt();

List.removeNode(n);

}

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

**public** **static** **void** Sort(ContainerList<PoliceFile> List)//// помилку пошуку

{

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

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

System.***out***.print("Cписок пустий");

**return** ;

}

System.***out***.print("Сортувати:\n"+

" 1. Ім`я\n"+

" 2. по-Батькові\n"+

" 3. Прізвище\n"+

" 4. датою народження\n"+

" 5. датою останнього позбавлення волі\n"+

" 6. датою останнього звільнення\n"+

"in: ");

**int** a=*in*.nextInt();

**switch**(a)

{

**case** 1:

List.sort(**new** Comparator<PoliceFile>() {

@Override

**public** **int** compare(PoliceFile o1, PoliceFile o2) {

**return** *comparison* (o1.getName(), o2.getName());

}

});

**break**;

**case** 2:

List.sort(**new** Comparator<PoliceFile>() {

@Override

**public** **int** compare(PoliceFile o1, PoliceFile o2) {

**return** *comparison*(o1.getSurname(), o2.getSurname()) ;

}

});

**break**;

**case** 3:

List.sort(**new** Comparator<PoliceFile>() {

@Override

**public** **int** compare(PoliceFile o1, PoliceFile o2) {

**return** *comparison*(o1.getLastname(), o2.getLastname());

}

});

**break**;

**case** 4:

List.sort(**new** Comparator<PoliceFile>() {

@Override

**public** **int** compare(PoliceFile o1, PoliceFile o2) {

**return** *comparison*(o1.getDateOfBirth(), o2.getDateOfBirth());

}

});

**break**;

**case** 5:

List.sort(**new** Comparator<PoliceFile>() {

@Override

**public** **int** compare(PoliceFile o1, PoliceFile o2) {

**return** *comparison*(o1.getDateOfLastImprisonment(), o2.getDateOfLastImprisonment());

}

});

**break**;

**case** 6:

List.sort(**new** Comparator<PoliceFile>() {

@Override

**public** **int** compare(PoliceFile o1, PoliceFile o2) {

**return** *comparison*(o1.getDateOfLastreLease(), o2.getDateOfLastreLease());

}

});

**break**;

}

System.***out***.print(" \*сортування завершено ");

}

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

**public** **static** **int** comparison(String a,String b)

{

**int** len=0;

**if**(a.length()<b.length())len=a.length();

**else** len=b.length();

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

{

**if** (a.charAt(i)>b.charAt(i)) **return** 1;

**if** (a.charAt(i)<b.charAt(i)) **return** -1;

}

**if**(a.length()<b.length())**return** -1;

**else** **if** (a.length()>b.length()) **return** 1;

**else** **return** 0;

}

**public** **static** **int** comparison(Date a,Date b) //0 a=b 1 a>b 2 a<b

{

**if**(a.getYear()==b.getYear())

{

**if**(a.getMoon()==b.getMoon())

{

**if**(a.getDay()==a.getDay()) **return** 0;

**else** **if**( a.getDay()>b.getDay()) **return** 1;

**else** **return** -1;

}

**else** **if**( a.getMoon()>b.getMoon()) **return** 1;

**else** **return** -1;

}

**else** **if**( a.getYear()>b.getYear()) **return** 1;

**else** **return** -1;

}

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

**public** **class** ContainerList<T **extends** Serializable >

**implements** List<T>,Iterable<T> {

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

**public** Element<T> head = **null** ;//head

**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);

}

}

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

……..

**public** **void** sort (Comparator<PoliceFile> c )

{

ArrayList<T> list=**new** ArrayList<>();

current = head;

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

list.add( current.num);

current = current.next;

}

list.sort((Comparator<? **super** T>) c);

clear();

**for**(**var** a:list)

{

add(a);

}

}

……..

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

**3 ВАРІАНТИ ВИКОРИСТАННЯ**

Оновлена попередню програма.

Додано можливість сортувати елементи списку та можливість автоматичного режиму програми.

На початку роботи користувач повинен подати команду. Якщо передає команду «-auto»-то програма працює в автоматичному режимі. У такому режимі програма відкриває заготовлений файли та зчитує дані. Далі просто програма виконує вивід даних сортування та інше.

В іншому випадку програма працю в стандартному режимі із користувачем.

Сортування виконане для всіх полів.

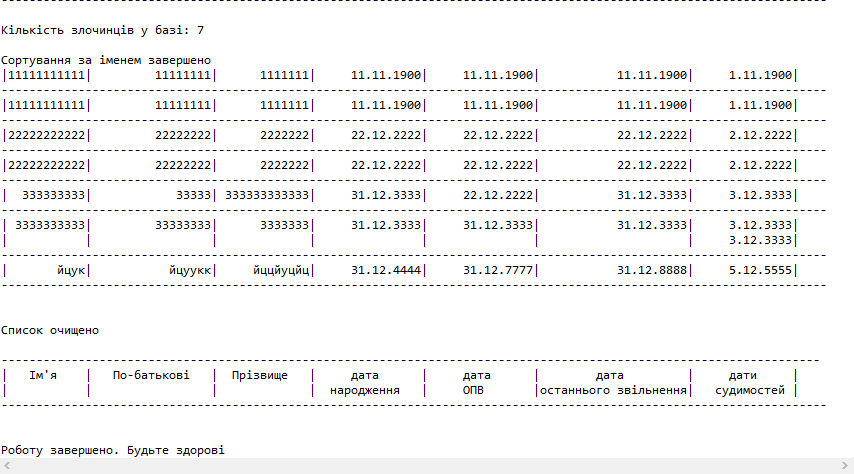


Рисунок 1. Виконання роботи програми автоматичному режимі

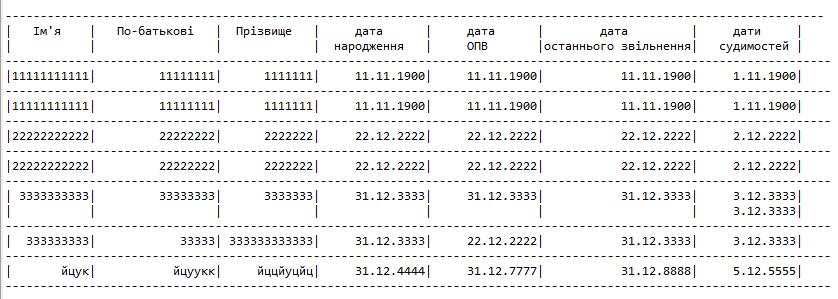


Рисунок 2 – Сортування списку за датою народження

ВИСНОВКИ

Під час виконання лабораторної роботи було набуто навички роботи створення сортування контейнера списку та демонстрування роботи програми в автоматичному режимі.