**Лабораторна робота №6**

**Серіалізація/десеріалізація об'єктів. Бібліотека класів користувача**

**Мета:** Тривале зберігання та відновлення стану об'єктів. Ознайомлення з принципами серіалізації/десеріалізації об'єктів. Використання бібліотек класів користувача.

**1 ВИМОГИ**

1. Реалізувати і продемонструвати тривале зберігання/відновлення раніше розробленого контейнера за допомогою серіалізації/десеріалізації.
2. Обмінятися відкомпільованим (без початкового коду) службовим класом (Utility Class) рішення задачі л.р. №3 з іншим студентом (визначає викладач).
3. Продемонструвати послідовну та вибіркову обробку елементів розробленого контейнера за допомогою власного і отриманого за обміном службового класу.
4. Реалізувати та продемонструвати порівняння, сортування та пошук елементів у контейнері.
5. Розробити консольну програму та забезпечити діалоговий режим роботи з користувачем для демонстрації та тестування рішення.
   1. **Розробник**

* П.І.Б: Заночкин. Є. Д.
* Группа: КІТ-119а
* Варіант: 7

1. **ОПИС ПРОГРАМИ**
   1. **Було використано наступні засоби:**

FileOutputStream fos = new FileOutputStream("Serial.ser"); ObjectOutputStream oos = new ObjectOutputStream(fos); oos.writeObject(container); oos.close() – серіалізація;

FileInputStream fis = new FileInputStream("Serial.ser"); ObjectInputStream ois = new ObjectInputStream(fis); MyContainer temp = (MyContainer) ois.readObject(); ois.close() – десеріалізація;

* 1. **Ієрархія та структура класів**

Було створено 3 класи:

* public class Main – містить метод main;
* public class MyContainer – клас, що містить методи для роботи з контейнером;
* public class MyIterator – клас, що містить методи для роботи з ітератором.

Також було використано мій хелпер клас, що був розроблений в лабораторній роботі №3.

* 1. **Важливі фрагменти програми**

package ua.khpi.oop.zanochkyn06;

import java.io.Serializable;

import java.util.Iterator;

public class MyContainer implements Serializable

{

private String[] arrayStr;

private int size;

/\*

\* Конструктор класу MyContainer

\*/

public MyContainer(String... str)

{

if (str.length != 0)

{

size = str.length;

arrayStr = new String[size];

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

arrayStr[i] = str[i];

}

}

/\*

\* Метод toString, який повертає вміст контейнера у вигляді рядка

\*/

public String toString()

{

StringBuilder sb = new StringBuilder();

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

sb.append(arrayStr[i] + " ");

return sb.toString();

}

/\*

\* Метод add, який додає рядок в кінець масиву

\*/

public void add(String string)

{

String newArr[] = new String[size + 1];

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

newArr[i] = arrayStr[i];

newArr[size] = string;

size++;

arrayStr = newArr;

}

/\*

\* Метод clear, який очищує контейнер

\*/

public void clear()

{

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

arrayStr[i] = null;

size = 0;

}

/\*

\* Метод remove, який видаляє перший випадок вказаного елемента з контейнера

\*/

boolean remove(String string)

{

boolean flag = false;

int pos = 0;

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

if(arrayStr[i].equals(string))

{

flag = true;

pos = i;

break;

}

if (flag)

{

String newArr[] = new String[size - 1];

for (int i = 0; i < pos; i++)

newArr[i] = arrayStr[i];

for (int i = pos, j = pos + 1; j < size; i++, j++)

newArr[i] = arrayStr[j];

size--;

arrayStr = newArr;

}

return flag;

}

/\*

\* Метод toArray, який повертає масив, що містить всі елементи у контейнері

\*/

public Object[] toArray()

{

Object[] arr = new Object[size];

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

arr[i] = arrayStr[i];

return arr;

}

/\*

\* Метод size, який повертає кількість елементів у контейнері

\*/

public int size()

{

return size;

}

/\*

\* Метод contains, який повертає true, якщо контейнер містить вказаний елемент

\*/

public boolean contains(String string)

{

for (String str : arrayStr)

if (str.equals(string))

return true;

return false;

}

/\*

\* Метод containsAll, який повертає true, якщо контейнер містить всі елементи з зазначеного у параметрах

\*/

public boolean containsAll(MyContainer container)

{

if (size == 0 || container.size() == 0 || size < container.size())

return false;

int count = 0;

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

for (int j = 0; j < container.size(); j++)

if(arrayStr[i].equals(container.arrayStr[j]))

{

count++;

break;

}

if(count == container.size())

return true;

else

return false;

}

/\*

\* Метод printContainer, який виводить у консоль зміст контейнера

\*/

public void printContainer()

{

for(String str : arrayStr)

System.out.println(str);

System.out.println();

}

/\*

\* Метод sortLength, який сортує контейнер за довжиною

\*/

public void sortLength(int option)

{

switch(option)

{

case 1:

for(int i = 0; i < size - 1; i++)

for(int j = 0; j < size - 1; j++)

if(arrayStr[j].length() > arrayStr[j+1].length())

{

String temp = arrayStr[j];

arrayStr[j] = arrayStr[j+1];

arrayStr[j+1] = temp;

}

break;

case 2:

for(int i = 0; i < size - 1; i++)

for(int j = 0; j < size - 1; j++)

if(arrayStr[j].length() < arrayStr[j+1].length())

{

String temp = arrayStr[j];

arrayStr[j] = arrayStr[j+1];

arrayStr[j+1] = temp;

}

break;

}

}

/\*

\* Метод sortAlphabet, який сортує контейнер за алфавітом

\*/

public void sortAlphabet(int option)

{

switch(option)

{

case 1:

for(int i = 0; i < size - 1; i++)

for(int j = 0; j < size - 1; j++)

if(arrayStr[j].compareTo(arrayStr[j+1]) > 0)

{

String temp = arrayStr[j];

arrayStr[j] = arrayStr[j+1];

arrayStr[j+1] = temp;

}

break;

case 2:

for(int i = 0; i < size - 1; i++)

for(int j = 0; j < size - 1; j++)

if(arrayStr[j].compareTo(arrayStr[j+1]) < 0)

{

String temp = arrayStr[j];

arrayStr[j] = arrayStr[j+1];

arrayStr[j+1] = temp;

}

break;

}

}

/\*

\* Метод findElement, який знаходить позицію, на якій знаходится елемент у контейнері

\*/

public int findElement(String string)

{

int pos = 0;

for(String str : arrayStr)

{

if(str.equals(string))

return pos;

pos++;

}

return -1;

}

/\*

\* Метод compareElements, який порівнює елементи в контейнері

\*/

public int compareElements(int pos1, int pos2)

{

if(pos1 > size || pos2 > size)

return -1;

if(arrayStr[pos1 - 1].equals(arrayStr[pos2 - 1]))

return 1;

else

return 0;

}

/\*

\* Метод iterator, який повертає ітератор відповідно до Interface Iterable.

\*/

public Iterator<String> iterator()

{

return new MyIterator<String>();

}

public class MyIterator<String> implements Iterator

{

int index;

/\*

\* Method that returns true if the iteration has more elements

\*/

@Override

public boolean hasNext()

{

return index < size;

}

/\*

\* Method that returns the next element in the iteration

\*/

@Override

public Object next()

{

return arrayStr[index++];

}

/\*

\* Method that removes from the underlying collection the last element returned by this iterator

\*/

@Override

public void remove()

{

MyContainer.this.remove(arrayStr[index - 1]);

}

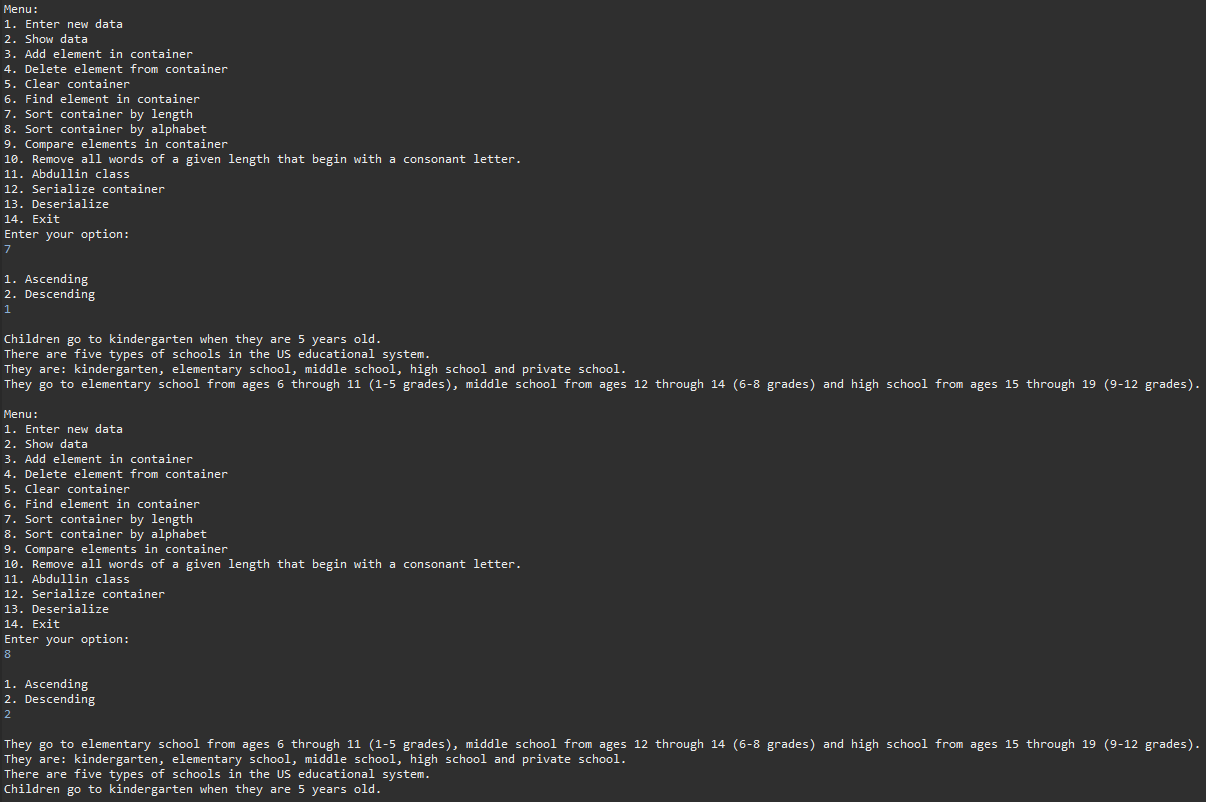
}

}

1. **Результати роботи програми**

****

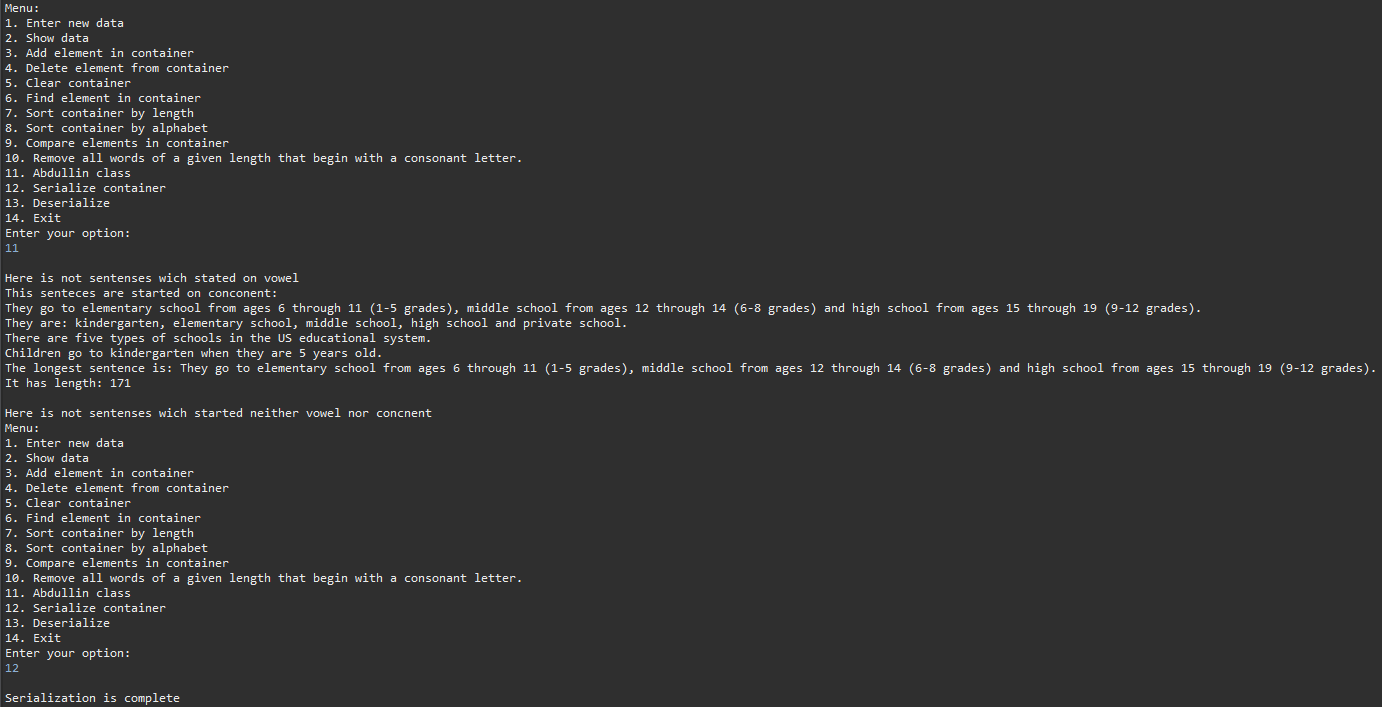
а)

****

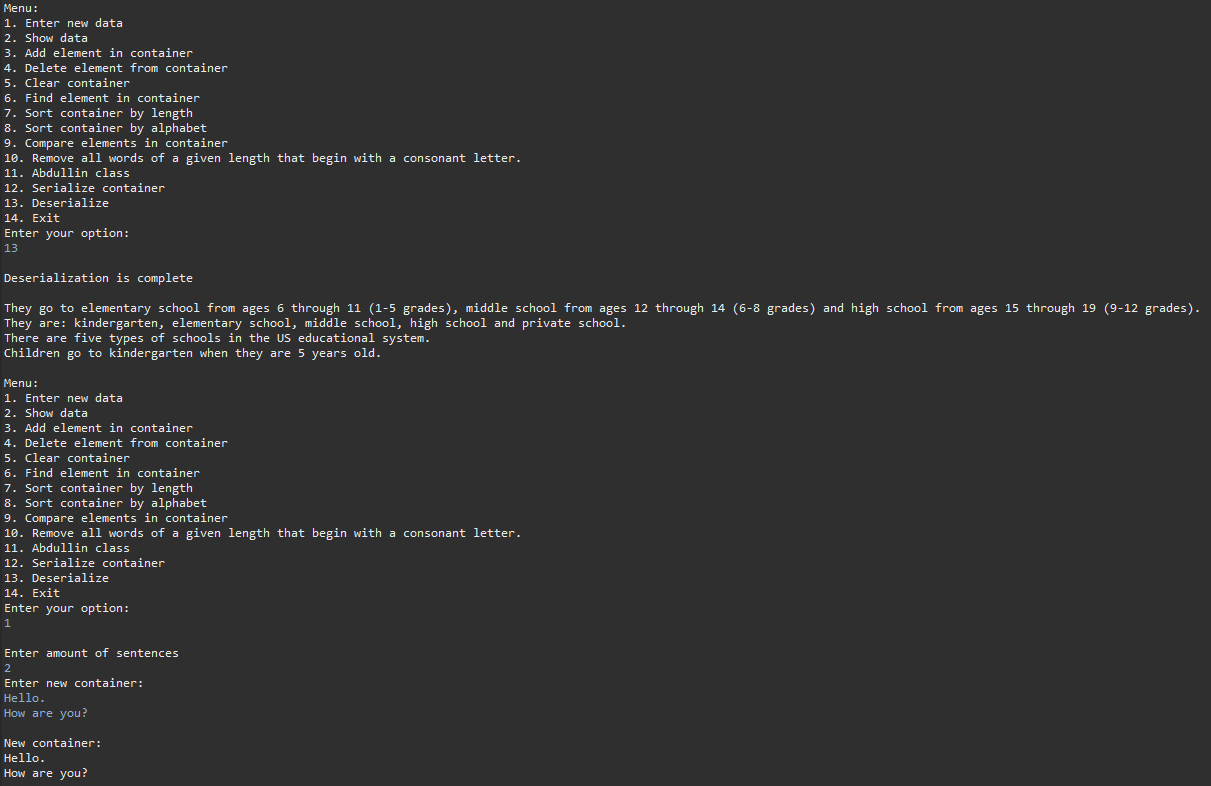
б)

****

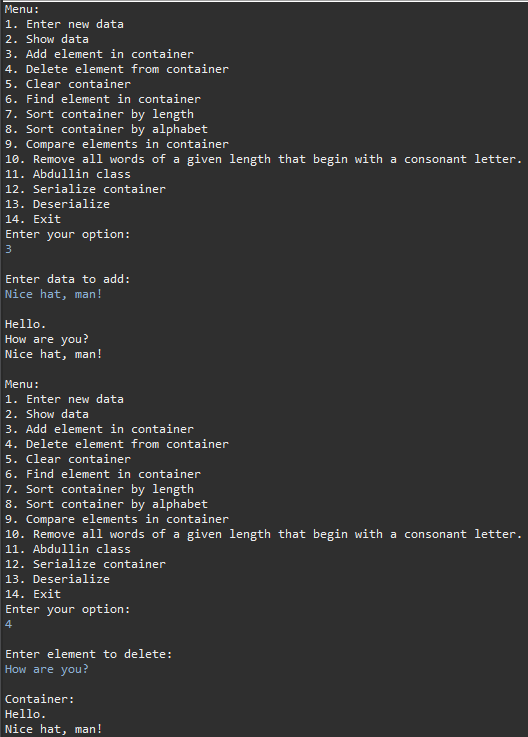
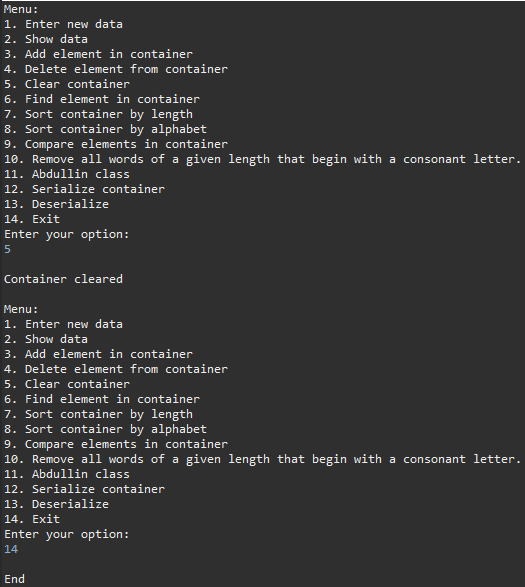
в)

****

г)

****

ґ)

** **

д) е)

Рисунок 6.1 – Результат роботи програми у середовищі Eclipse

**Висновок**

Під час виконання лабораторної роботи було набуто навичок роботи з серіалізацією\десеріалізацієюм та з розробки бібліотеки класів користувача у середовищі Eclipse IDE.