Лабораторна робота №5

**РОЗРОБКА ВЛАСНИХ КОНТЕЙНЕРІВ. ІТЕРАТОРИ**

**Мета:** Набуття навичок розробки власних контейнерів. Використання ітераторів.

**ВИМОГИ**

**Розробник:**

- Косінов Владислав Дмитрович;

- КІТ-120б;

- Варіант №8.

**Загальне завдання:**

1) Розробити клас-контейнер, що ітерується для збереження початкових даних завдання л.р. №3 у вигляді масиву рядків з можливістю додавання, видалення і зміни елементів.

2) В контейнері реалізувати та продемонструвати наступні методи:

* String toString() повертає вміст контейнера у вигляді рядка;
* void add(String string) додає вказаний елемент до кінця контейнеру;
* void clear() видаляє всі елементи з контейнеру;
* boolean remove(String string) видаляє перший випадок вказаного елемента з контейнера;
* Object[] toArray() повертає масив, що містить всі елементи у контейнері;
* int size() повертає кількість елементів у контейнері;
* boolean contains(String string) повертає true, якщо контейнер містить вказаний елемент;
* boolean containsAll(Container container) повертає true, якщо контейнер містить всі елементи з зазначеного у параметрах;
* public Iterator<String> iterator() повертає ітератор відповідно до Interface Iterable.

3) В класі ітератора відповідно до Interface Iterator реалізувати методи:

* public boolean hasNext();
* public String next();
* public void remove().

4) Продемонструвати роботу ітератора за допомогою циклів while и for each.

5) Забороняється використання контейнерів (колекцій) і алгоритмів з Java Collections Framework.

**ОПИС ПРОГРАМИ**

**Ієрархія та структура класів:**

**class** Main – точка входу в програму;

**class** aContainer – розроблений клас-контейнер

**ТЕКСТ ПРОГРАМИ**

Текст класу **Main**:

public class Main {

    public static void main(String[] args) {

        aContainer container1 = new aContainer();

        String string1 = new String();

        //Container's methods

        container1.add("asd");

        container1.add("dsa");

        container1.add("zxc");

        container1.add("\*cxz\*");

        container1.add("reverse");

        System.out.println("\nmethod size()");

        System.out.println("Size: " + container1.size());

        System.out.println("\nmethod toString()");

        System.out.println("toString: " + container1.toString());

        String[] s1 = container1.toArray();

        System.out.println("\nmethod toArray()");

        for (String s : s1) {

            System.out.println(s);

        }

        System.out.println("\nmethod contains()");

        System.out.println(container1.contains("Big"));

        System.out.println("\nmethod containsAll()");

        aContainer container2 = new aContainer();

        container2.add("asd");

        container2.add("dsa");

        container2.add("zxc");

        container2.add("\*cxz\*");

        container2.add("reverse");

        System.out.println("Should return true");

        System.out.println("Result: " + container1.containsAll(container2));

        System.out.println("\nmethod remove()");

        container1.remove("dsa");

        for (String s : container1) {

            System.out.println(s);

        }

        System.out.println("\nmethod clear()");

        container1.clear();

        for (String s : container1) {

            System.out.println(s);

        }

        // Using iterator's methods

        System.out.println("\nIterator's methods\n");

        aContainer.aIterator iterator = container2.iterator();

        for (String s : container2) {

            System.out.print(s + ' ');

        }

        System.out.println();

        if (iterator.hasNext()) {

            System.out.println(iterator.next());

        }

        iterator.remove();

        for (String s : container2) {

            System.out.print(s + ' ');

        }

        System.out.println();

        if (iterator.hasNext()) {

            System.out.println(iterator.next());

        }

        iterator.remove();

        for (String s : container2) {

            System.out.print(s + ' ');

        }

    }

}

Текст класу **aContainer**:

import java.util.Arrays;

import java.util.Iterator;

import java.util.NoSuchElementException;

import java.util.Objects;

/\*\*

 \* Class aContainer.

 \* Contains the range of methods to manipulate a container.

 \* Class is iterable - can be iterated element by element.

 \*/

public class aContainer implements Iterable<String> {

    /\*\*

     \* Holds the elements of a container.

     \*/

    private String[] values;

    /\*\*

     \* Constructor for making new string

     \*/

    public aContainer() {

        values = new String[0];

    }

    /\*\*

     \* Method concatenates all container elements into a string.

     \*

     \* @return container in a string

     \*/

    public String toString() {

        StringBuilder string = new StringBuilder(new String());

        for (String s : values)

            string.append(s + " ");

        return string.toString();

    }

    /\*\*

     \* Method for adding elements to a container.

     \*

     \* @param string - string to initialize a new container element

     \*/

    public boolean add(String string) {

        try {

            String[] temp = values;

            values = new String[temp.length + 1];

            System.arraycopy(temp, 0, values, 0, temp.length);

            values[values.length - 1] = string;

            return true;

        } catch (ClassCastException ex) {

            ex.printStackTrace();

        }

        return false;

    }

    /\*\*

     \* Method for resetting a container.

     \*/

    public void clear() {

        for (int i = 0; i < values.length; i++) {

            values[i] = null;

        }

    }

    /\*\*

     \* Method for removing an exact element by string criteria.

     \*

     \* @param string string to specify the element to remove

     \* @return false if removing cannot be done(no elements in container)

     \* true if element has been found and successfully deleted

     \*/

    boolean remove(String string) {

        int pos = 0;

        for (int i = 0; i < values.length; i++) {

            if (Objects.equals(values[i], string)) {

                break;

            } else pos++;

        }

        try {

            String[] temp = values;

            values = new String[temp.length - 1];

            System.arraycopy(temp, 0, values, 0, pos);

            int elemToDestinate = temp.length - pos - 1; //позиция, в которую нужно начинать копировать кроме вырезанного

            System.arraycopy(temp, pos + 1, values, pos, elemToDestinate);

            return true;

        } catch (ClassCastException ex) {

            System.err.println("Error ClassCastException");

        }

        return false;

    }

    /\*\*

     \* Method for converting container to an array.

     \*

     \* @return an array of container elements

     \*/

    public String[] toArray() {

        return Arrays.copyOf(values, values.length);

    }

    /\*\*

     \* Method for receiving the size of container.

     \*

     \* @return current container size

     \*/

    public int size() {

        return values.length;

    }

    /\*\*

     \* Method for checking a container elements with a specified string.

     \*

     \* @param string string to find in a container

     \* @return true if contains, false if does not contain

     \*/

    boolean contains(String string) {

        for (String s : values) {

            if (Objects.equals(s, string)) {

                return true;

            }

        }

        return false;

    }

    /\*\*

     \* Method for checking the equality of two containers.

     \*

     \* @param container for comparing with another container

     \* @return true if both containers are the same

     \* false if they are different

     \*/

    boolean containsAll(aContainer container) {

        if (values == null || values.length == 0) {

            return false;

        }

        int il = 0;

        String[] toCompare;

        toCompare = container.toArray();

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

            if (this.contains(toCompare[i])) {

                il++;

            }

        }

        return il == container.size();

    }

    /\*\*

     \* Method for creating a correct iterator.

     \*

     \* @return a new iterator to a Container object

     \*/

    public aIterator iterator() {

        return new aIterator(values);

    }

    /\*\*

     \* Class aIterator.

     \* Contains two fields of lower and higher bound of a container.

     \* Constructor gets a storage field from Container and defines

     \* both bounds.

     \* Contains methods for iterating over a container,

     \* checking the existence of the next element and removing.

     \*

     \* @author Kosinov Vladyslav

     \*/

    public class aIterator implements Iterator<String> {

        /\*\*

         \* Lower bound of the container

         \*/

        private  int firstBound;

        /\*\*

         \* Upper bound of the container

         \*/

        private  int lastBound;

        /\*\*

         \* Constructor for processing the container data.

         \* Defines values of lower and higher bound.

         \*

         \* @param values - array of container elements

         \*/

        public aIterator(String[] values) {

            firstBound = -1;

            lastBound = values.length - 1;

        }

        /\*\*

         \* Returns {@code true} if the iteration has more elements.

         \* (In other words, returns {@code true} if {@link #next} would

         \* return an element rather than throwing an exception.)

         \*

         \* @return {@code true} if the iteration has more elements

         \*/

        @Override

        public boolean hasNext() {

            return firstBound < lastBound;

        }

        /\*\*

         \* Returns the next element in the iteration.

         \*

         \* @return the next element in the iteration

         \* @throws NoSuchElementException if the iteration has no more elements

         \*/

        @Override

        public String next() {

            if (!this.hasNext()) {

                throw new NoSuchElementException();

            } else {

                firstBound++;

                return values[firstBound];

            }

        }

        /\*\*

         \* Removes from the underlying collection the last element returned

         \* by this iterator (optional operation).  This method can be called

         \* only once per call to {@link #next}.

         \*/

        @Override

        public void remove() {

            try {

                String[] temp = values;

                values = new String[temp.length - 1];

                System.arraycopy(temp, 0, values, 0, firstBound);

                int elemToDestinate = temp.length - firstBound - 1; //позиция, в которую нужно начинать копировать кроме вырезанного

                System.arraycopy(temp, firstBound + 1, values, firstBound, elemToDestinate);

            } catch (ClassCastException ex) {

                System.err.println("Error");

            }

        }

    }

}

**РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ**

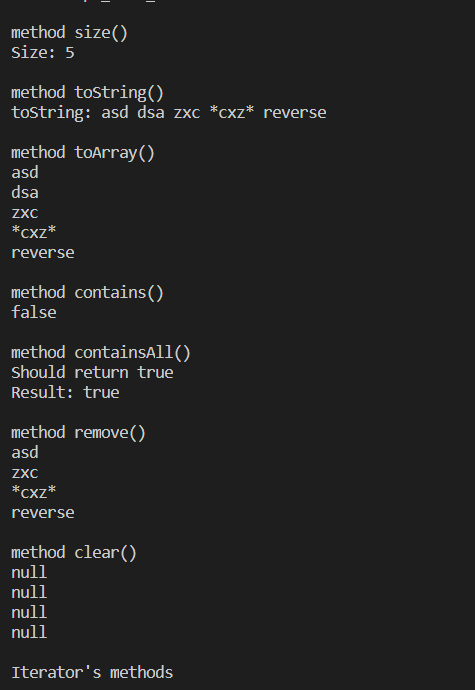


Рисунок 5.1 – Результат роботи програми

**ВАРІАНТИ ВИКОРИСТАННЯ**

Програма може використовуватись як контейнер для об’єктів типу String. Також є можливість ітерування по контейнеру.

**ВИСНОВОК**

Під час лабораторної роботи, набув навичок розробки власних контейнерів та навчився використовувати ітератори. Використав пакет import java.util.Arrays, import java.util.Iterator, import java.util.NoSuchElementException. Програма виконується без помилок.