Звіт

Лабораторна робота №5

РОЗРОБКА ВЛАСНИХ КОНТЕЙНЕРІВ. ІТЕРАТОРИ

Мета: Набуття навичок розробки власних контейнерів. Використання ітераторів.

ВИМОГИ

Розробник:

- Чугунов Вадим Юрійович;

- КІТ-119а;

- Варіант №24.

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

Опис змінних

TaskContainer container; // об’єкт створеного класу TaskContainer
TaskIterator iter; // об’єкт створеного класу TaskIterator

Ієрархія та структура класів

Class Main – точка входу в програму;

Class TaskIterator – inner-клас класу TaskContainer;

Class TaskContainer – розроблений клас-контейнер.

Class TestHelper – утилітарний клас, який має в собі Split метод.

ТЕКСТ ПРОГРАМИ

Текст файлу Main.java

```
package ua.oop.khpi.chugunov05;


import ua.oop.khpi.chugunov05.TaskContainer.TaskIterator;


public class Main {


    /**
     * The main method.
     *
     * @param args - arguments of main method
     */
```

```
public static void main(final String[] args) {
    // Container
   TaskContainer TestContainer1 = new TaskContainer();
   String word2 = "Task to count";
    String word1 =  "Task to count words";

   //Arrays of our splitting strings
    String[] array = TestHelper.SplitString(word1); // split method
    String[] array2 = TestHelper.SplitString(word2);

    // Our Values to ADD ("Task to count words")
   TestContainer1.addElemOfArray(array);

    // Create OUR Iterator
    TaskIterator iter = TestContainer1.iterator();
    System.out.println("\n================================================");

    // For Each loop printing values
    System.out.print("For each loop:\t\t   ");
    for (String s : TestContainer1) {
        System.out.print(s + " ");
    }
    System.out.println("|");
    System.out.println();

    System.out.print("While loop:\t\t\t   ");
    // While loop printing values
    while (iter.hasNext()) {
        System.out.print(iter.next() + " ");
    }
    System.out.println("|");
    System.out.println();

    //  toString() method
    System.out.println("Method toString():\t   " + TestContainer1.toString()+"|");
```

```
// The second container
TaskContainer TestContainer2 = new TaskContainer();
// Adding values to container ("count words")
TestContainer2.addElemOfArray(array2);

System.out.println("\n================================================");
System.out.println("The Method that returns bollean (TRUE/FALSE):");

// ContainsAll() method
System.out.println("Answer is "+TestContainer1.containsAll(TestContainer2));
// Contains() method
System.out.println("Answer is "+TestContainer2.contains("count"));
TestContainer2.add("words");
System.out.println("Answer is "+TestContainer1.containsAll(TestContainer2));
// Remove() methods
TestContainer2.remove("words");



System.out.println("\n================================================");

// Second iterator
TaskIterator iter2 = TestContainer2.iterator();
//  Iterator's methods
for (String s : TestContainer2) {
    System.out.print(s + ' ');
}
System.out.println();
if (iter2.hasNext()) {
    System.out.println(iter2.next());
}
iter2.remove();
for (String s : TestContainer2) {
    System.out.print(s + ' ');
}
System.out.println();
if (iter2.hasNext()) {
```

```
            System.out.println(iter2.next());
        }
        iter2.remove();
        for (String s : TestContainer2) {
            System.out.print(s + ' ');
        }
        if(iter2.hasNext()) {
             System.out.print("\nYEAH!!!Have next!");
        } else {
             System.out.print("\nNOPE!!!Haven't next!");
        }
        System.out.println("\n================================================");

    }

}
```

Текст файлу TaskContainer.java

```
package ua.oop.khpi.chugunov05;

import java.util.NoSuchElementException;
import java.util.Iterator;
import java.util.Arrays;



/**
 * This is class TaskContainer.
 * Class is iterable - can be iterated element by element.
 *
 * @author Chugunov Vadim
 */
public class TaskContainer implements Iterable<String> {
    /** Holding the elements. */
    private String[] buf = null;

    /**
```

```
  *The  Method that  concatenates all container elements into a string.
  * @return container in a string
  */
 @Override
 public String toString() {
     if (buf == null || buf.length == 0) {
         return null;
     } else {
         StringBuilder builder = new StringBuilder();
         for (String i : buf) {
             builder.append(i).append(' ');
         }
         return builder.toString();
     }
 }

 /**
 *  The override to add method for adding an elem of string array.
 *  @param str - string array
 */
public void addElemOfArray(final String[] str) {
  for (String i : str) {
      this.add(i);
    }
}

 /**
  * The Method that adding elem to the container.
  *  @param str - string to initialize a new container element
  */
 public void add(final String str) {
     if (buf == null) {
          buf = new String[1];
          buf[0] = str;
     } else {
          buf = Arrays.copyOf(buf, buf.length + 1);
```

```
          buf[buf.length - 1] = str;
    }
}
/**
 * The Method that resetting a container.
 */
public void clear() {
   buf = new String[0];
}
/**
 * The Method that removes element by string criteria.
 * @return false if removing cannot be done(no elements in container)
 *         true if element has been found and successfully deleted
 * @param str - string to specify the element to remove
 */
public boolean remove(final String str) {
    if (buf == null || buf.length == 0) {
        return false;
    }
    String[] newBuf = new String[buf.length - 1];
    int index;
    for (index = 0; index < buf.length; index++) {

        if (buf[index].equals(str)) {
            break;
        } else if (index == buf.length - 1) {
            return false;
        }
    }
    int j = 0;
    for (int k = 0; k < buf.length; k++) {
        if (k == index) {
            continue;
        }
        newBuf[j++] = buf[k];
    }
```

```
    buf = Arrays.copyOf(newBuf, newBuf.length);
    return true;
}
/**
 *The  Method that converts container to an array.
 * @return an array of container elements
 */
public String[] toArray() {
    if (buf == null) {
        return null;
    }
    return Arrays.copyOf(buf, buf.length);
}
/**
 *The  Method that receives the size of container.
 * @return current container size
 */
public int size() {
    if (buf == null) {
        return 0;
    }
    return buf.length;
}
/**
 * The Method that checks a container elements.
 * @param str - string to find in a container
 * @return true if contains, false if does not contain
 */
public boolean contains(final String str) {
    if (buf == null || buf.length == 0) {
        return false;
    }
    for (String i : buf) {
        if (i.equals(str)) {
            return true;
        }
```

```
    }
    return false;
}
/**
 *The  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
 */
public boolean containsAll(final TaskContainer container) {
    if (buf == null || buf.length == 0) {
        return false;
    }
    int equation = 0;
    String[] toCompare;
    toCompare = container.toArray();
    for (int i = 0; i < container.size(); i++) {
        if (this.contains(toCompare[i])) {
            equation++;
        }
    }
    return equation == container.size();
}
/**
 * The Method that creat's a iterator.
 * @return a new iterator to a Container object
 */
@Override
public TaskIterator iterator() {
    return new TaskIterator(buf);
}
/**
 * This is class TaskIterator.
 * Contains two fields of lower/higher bound of a container.
 * Contains methods for iterating over a container,
 * checking the existence of the next element and removing.
```

```
 * @author Chugunov Vadim
 */
public class TaskIterator implements Iterator<String> {
    private int lowerB;
    private int higherB;

    /**
     * Defines values of lower and higher bound.
     * @param buf - array of container elements
     */
    TaskIterator(final String[] buf) {
        lowerB = -1;
        higherB = buf.length-1;
    }
    /**
     * The Method that checks the existence of the next element.
     * @return true if the next element exists
     * false if it doesn't exist
     */
    @Override
    public boolean hasNext() {
        return lowerB < higherB;
    }
    /**
     * The Method that  for moving further through the container.
     * @return (current!) iterated element
     */
    @Override
    public String next() {
        if (!this.hasNext()) {
            throw new NoSuchElementException();
        } else {
            lowerB++;
            return buf[lowerB];
        }
    }
```

```
        /**
         *The Method that for removing the current element from iteration.
         */
        @Override
        public void remove() {
            String[] copyBuf = Arrays.copyOf(buf,
                                             buf.length);
            buf = new String[buf.length - 1];
            int j = 0;
            for (int i = 0; i < copyBuf.length; i++) {
                if (i != lowerB) {
                    buf[j++] = copyBuf[i];
                }
            }
            higherB--;
        }
    }
}
```

Текст файлу TestHelper.java

```
package ua.oop.khpi.chugunov05;
import java.util.ArrayList;
import java.util.List;


public class TestHelper  {

        public static String[] SplitString(String text) {
        List<String> words = new ArrayList<>();
        StringBuilder builder = new StringBuilder();
        int count = 0;
        char symbol;
        while (count < text.length()) {
          for (int i = count; i < text.toCharArray().length; i++)
          {
```

```
            symbol = text.toCharArray()[i];
            if((int)symbol == 32 | (int)symbol == 33 |(int)symbol == 58|(int)symbol ==
44|(int)symbol == 46) {
                    count++;
              break;
            }
            builder.append(symbol);
            count++;
          }
          words.add(builder.toString());
          builder = new StringBuilder();
        }

        if(builder.length() != 0) {
          words.add(builder.toString());
        }
        for (int i = 0; i < words.size(); i++) {
          if(words.get(i).length() == 0) {
            words.remove(i);
            }
        }
        String[] output = new String[words.size()];
        for (int i = 0; i < words.size(); i++) {
          output[i] = words.get(i);
        }
        return output;

        }
}
```

ВАРІАНТИ ВИКОРИСТАННЯ

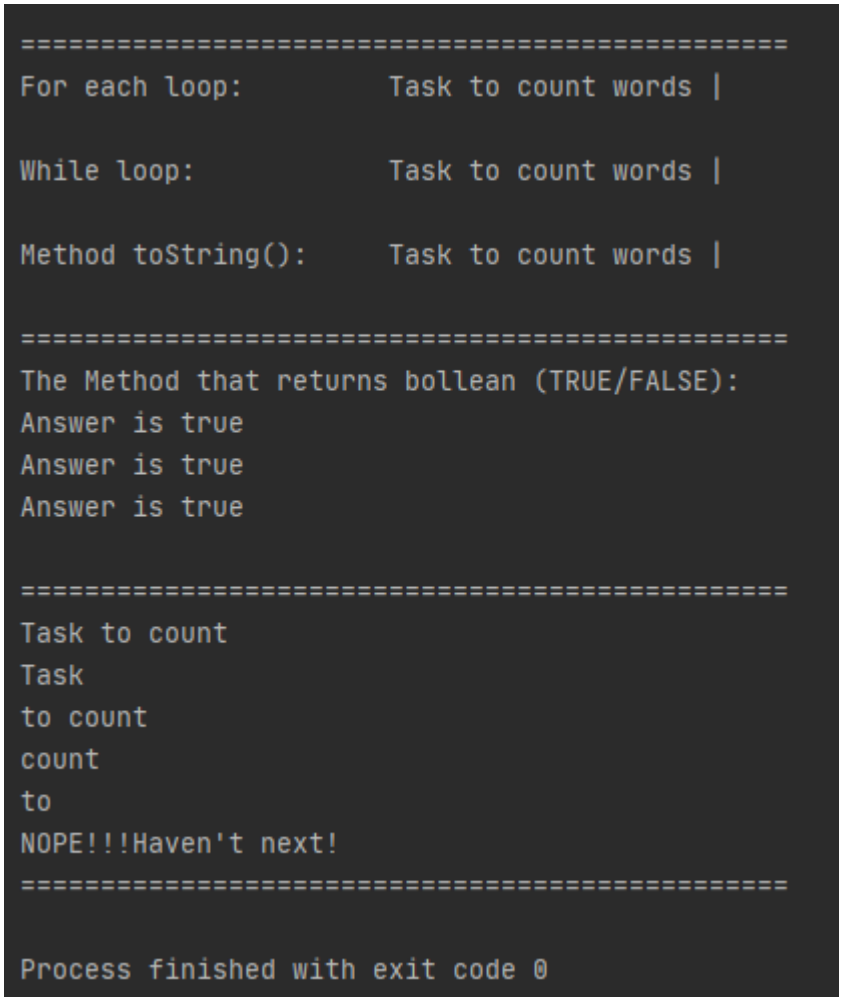

Рисунок 1 – Результат роботи програми

Дану програму можна використовувати для пошуку потрібного нам слова в тексті, та знаходження кількості повторів його в тексті. Використовувати інтерактивне меню задля простого доступу до елементів програми, також використовувати контейнер для об’єктів та ітерування.

ВИСНОВОК

Під час виконання лабораторної роботи було створено програму, яка має власний клас койнтейнер та ітератор. Реалізували та продемонстрували відповідні методи класу контейнера та ітератора.