

## РОЗРОБКА ВЛАСНИХ КОНТЕЙНЕРІВ. ІТЕРАТОРИ

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

### ВИМОГИ

#### Розробник:

- Веремчук Дарина Анатоліївна;
- КІТ-119д;
- Варіант №5.

#### 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:

<code>MyContainer container;</code>	<code>// об'єкт створеного класу MyContainer</code>
<code>MyIterator iter;</code>	<code>// об'єкт створеного класу MyIterator</code>

### Ієрархія та структура класів:

**class** `Veremchuk05` – точка входу в програму;

**class** `MyContainer` – розроблений клас-контейнер;

**class** `MyIterator` – внутрішній клас класу `MyContainer`.

## ТЕКСТ ПРОГРАМИ

### Текст класу **Veremchuk05**:

```
package ua.oop.khpi.veremchuk05;

import ua.oop.khpi.veremchuk05.MyContainer.MyIterator;

import ua.oop.khpi.veremchuk03.Helper;

import org.w3c.dom.ls.LSOutput;

public class Veremchuk05 {

    private Veremchuk05() {

    }

    /**
     * An entry point - main method.
     * @param args - arguments of main method
     */

    public static void main(final String[] args) {

        // Creating container

        MyContainer container = new MyContainer();

        //Source strings for processing with Helper class

        String text = "Hello, my name is Darina";

        String text2 = "Hello, my name";

        String[] lines = Helper.DivString(text); // alternative to split method

        String[] lines2 = Helper.DivString(text2);

        StringBuilder builder = new StringBuilder();

        container.add(lines);

        // Creating iterator
```

```

MyIterator iter = container.iterator();

System.out.print("While:      ");

// Printing values by a while loop
while (iter.hasNext()) {
    System.out.print(iter.next() + " ");
}

System.out.println();

// Printing values by a for each loop
System.out.print("For each:  ");
for (String s : container) {
    System.out.print(s + " ");
}

System.out.println();

// Using toString() method
System.out.println("toString(): " + container.toString()+ "\n");

// Creating second container
MyContainer container2 = new MyContainer();

// Add values into the second container
container2.add(lines2);


System.out.println("Testing boolean methods:");

// Using contains() method
System.out.println(container.contains("my"));

// Using containsAll() method
System.out.println(container.containsAll(container2));

container2.add("Nastya");

System.out.println(container.containsAll(container2));

// Using remove() method

```

```
container2.remove("Nastya");

System.out.println(container.containsAll(container2) + "\n");

// Creating second iterator

MyIterator iter1 = container.iterator();

// Using iterator's methods

for (String s : container) {

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

}

System.out.println();

if (iter1.hasNext()) {

    System.out.println(iter1.next());

}

iter1.remove();

for (String s : container) {

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

}

System.out.println();

if (iter1.hasNext()) {

    System.out.println(iter1.next());

}

iter1.remove();

for (String s : container) {

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

}

System.out.println();

if (iter1.hasNext()) {

    System.out.println(iter1.next());

}

iter1.remove();
```

```

        iter1.toString();
    }
}

```

### Текст класу **MyContainer**:

```

package ua.oop.khpi.veremchuk05;

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

/**
 * Class MyContainer.
 * Contains the range of methods to manipulate a container.
 * Class is iterable - can be iterated element by element.
 *
 * @author Veremchuk Darina
 */
public class MyContainer implements Iterable<String> {

    /** Holds the elements of a container. */
    private String[] buffer = null;

    /**
     * Method concatenates all container elements into a string.
     * @return container in a string
     */
    @Override

```

```

public String toString() {

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

        return null;

    } else {

        StringBuilder builder = new StringBuilder();

        for (String i : buffer) {

            builder.append(i).append(' ');

        }

        return builder.toString();

    }

}

/**
 * Method for adding elements to a container.
 *
 * @param str - string to initialize a new container element
 */
public void add(final String str) {

    if (buffer == null) {

        buffer = new String[1];

        buffer[0] = str;

    } else {

        buffer = Arrays.copyOf(buffer, buffer.length + 1);

        buffer[buffer.length - 1] = str;

    }

}

/**
 * Method for adding elements of string array to a container.
 *
 * @param str - string array
 */
public void add(final String[] str) {

```

```

        for (String i : str) {

            this.add(i);

        }

    }

    /**
     * Method for resetting a container!
     */

    public void clear() {

        buffer = new String[0];

    }

    /**
     * Method for removing an exact 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 (buffer == null || buffer.length == 0) {

            return false;

        }

        String[] newBuffer = new String[buffer.length - 1];

        int index;

        for (index = 0; index < buffer.length; index++) {

            if (buffer[index].equals(str)) {

                break;

            } else if (index == buffer.length - 1) {

                return false;

            }

        }

```



```

    }

    int j = 0;

    for (int k = 0; k < buffer.length; k++) {

        if (k == index) {

            continue;

        }

        newBuffer[j++] = buffer[k];

    }

    buffer = Arrays.copyOf(newBuffer, newBuffer.length);

    return true;

}

/**
 * Method for converting container to an array.
 * @return an array of container elements
 */
public String[] toArray() {

    if (buffer == null) {

        return null;

    }

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

}

/**
 * Method for receiving the size of container.
 * @return current container size
 */
public int size() {

    if (buffer == null) {

        return 0;

    }

```

```

        return buffer.length;
    }

    /**
     * Method for checking a container elements with a specified string.
     * @param str - string to find in a container
     * @return true if contains, false if does not contain
     */
    public boolean contains(final String str) {
        if (buffer == null || buffer.length == 0) {
            return false;
        }
        for (String i : buffer) {
            if (i.equals(str)) {
                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
     */
    public boolean containsAll(final MyContainer container) {
        if (buffer == null || buffer.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();

    }

    /**
     * Method for creating a correct iterator.
     * @return a new iterator to a Container object
     */

    @Override

    public MyIterator iterator() {

        return new MyIterator(buffer);

    }

    /**
     * Class MyIterator.
     * 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 Veremchuk Darina
     */

    public class MyIterator implements Iterator<String> {

        /** Lower bound of a container. */

        private int lowerBound;

```

```

/** Higher bound of a container. */

private int higherBound;

/**
 * Constructor for processing the container data.
 * Defines values of lower and higher bound.
 * @param buf - array of container elements
 */
MyIterator(final String[] buf) {
    lowerBound = -1;
    higherBound = buf.length-1;
}

/**
 * Method 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 lowerBound < higherBound;
}

/**
 * Method for moving further through the container.
 * @return current iterated element
 */
@Override
public String next() {
    if (!this.hasNext()) {
        throw new NoSuchElementException();
    } else {

```

```

        lowerBound++;

        return buffer[lowerBound];
    }
}

/**
 * Method for removing the current element from iteration.
 */
@Override
public void remove() {
    String[] copyBuffer = Arrays.copyOf(buffer,
                                         buffer.length);

    buffer = new String[buffer.length - 1];

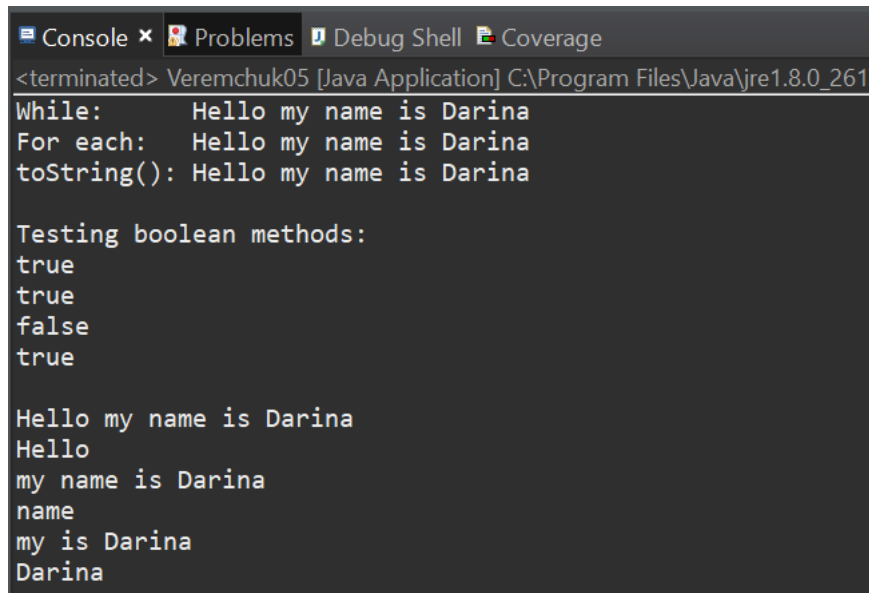
    int j = 0;

    for (int i = 0; i < copyBuffer.length; i++) {
        if (i != lowerBound) {
            buffer[j++] = copyBuffer[i];
        }
    }

    higherBound--;
}
}
}

```

## РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ

A screenshot of a Java IDE's console window. The window has tabs for 'Console', 'Problems', 'Debug Shell', and 'Coverage'. The 'Console' tab is active, showing the output of a Java application. The output text is as follows:

```
<terminated> Veremchuk05 [Java Application] C:\Program Files\Java\jre1.8.0_261
While:      Hello my name is Darina
For each:   Hello my name is Darina
toString(): Hello my name is Darina

Testing boolean methods:
true
true
false
true

Hello my name is Darina
Hello
my name is Darina
name
my is Darina
Darina
```

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, набула навичок розробки власних контейнерів та навчилася використовувати ітератори. Використала пакет `import java.util.Arrays`, `import java.util.Iterator`, `import java.util.NoSuchElementException`. Програма виконується без помилок.