Звіт

Автор: Пумня О., КІТ101.8б

Дата: 07.10.2019

Лабораторна робота №5

РОЗРОБКА ВЛАСНИХ КОНТЕЙНЕРІВ. ІТЕРАТОРИ

**Мета.** Набуття навичок розробки власних контейнерів та використання ітераторів.

**Вимоги:**

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.

1. В класі ітератора відповідно до Interface Iterator реалізувати методи:

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

1. Продемонструвати роботу ітератора за допомогою циклів while и for each.
2. Забороняється використання контейнерів (колекцій) і алгоритмів з Java Collections Framework.

ЗАВДАННЯ ДО РОБОТИ

Переробити попередню роботу так, щоб вона відповідала вимогам, описаним вище.

ОПИС ПРОГРАМИ

**Опис змінних**

MyContainer container; // об’єкт створеного класу MyContainer

MyIterator iter; // об’єкт створеного класу MyIterator

**Ієрархія та структура класів**

**class** Pumnya05 – точка входу в програму.

**class** MyContainer – розроблений клас-контейнер.

**class** MyIterator – inner-клас класу MyContainer.

ТЕКСТ ПРОГРАМИ

**Текст файлу Pumnya05.java**

package labs.pumnya05;

import labs.pumnya05.MyContainer.MyIterator;

public final class Pumnya05 {

private Pumnya05() {

}

/\*\*

\* An entry point - main method.

\* @param args - arguments of main method

\*/

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

// Creating container

MyContainer container = new MyContainer();

// Adding values

container.add("Germany");

container.add("France");

container.add("Italy");

container.add("Spain");

container.add("Russia");

// 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());

// Creating second container

MyContainer container2 = new MyContainer();

// Add values into the second container

container2.add("Germany");

container2.add("France");

container2.add("Italy");

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

// Using contains() method

System.out.println(container.contains("Ukraine"));

// Using containsAll() method

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

container2.add("Ukraine");

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

// Using remove() method

container2.remove("Ukraine");

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

// Creating second iterator

MyIterator iter2 = container2.iterator();

// Using iterator's methods

for (String s : container2) {

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

}

System.out.println();

if (iter2.hasNext()) {

System.out.println(iter2.next());

}

iter2.remove();

for (String s : container2) {

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

}

System.out.println();

if (iter2.hasNext()) {

System.out.println(iter2.next());

}

iter2.remove();

for (String s : container2) {

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

}

}

}

**Текст файлу MyContainer.java**

package labs.pumnya05;

import org.jetbrains.annotations.NotNull;

import java.io.Serializable;

import java.util.Arrays;

import java.util.Iterator;

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 Pumnya Alexander

\*/

public class MyContainer implements Iterable<String>, Serializable {

/\*\* Identifying key for serialization. \*/

private static final long serialVersionUID = 6126733392129125019L;

/\*\* Holds the elements of a container. \*/

protected 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 string - string to initialize a new container element

\*/

public void add(final String string) {

if (buffer == null) {

buffer = new String[1];

buffer[0] = string;

} else {

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

buffer[buffer.length - 1] = string;

}

}

/\*\*

\* Returns last element.

\* @return last element of container

\*/

public String last() {

return buffer[buffer.length - 1];

}

/\*\*

\* 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 string - string to specify the element to remove

\*/

public boolean remove(final String string) {

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(string)) {

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 string - string to find in a container

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

\*/

public boolean contains(final String string) {

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

return false;

}

for (String i : buffer) {

if (i.equals(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

\*/

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

\*/

@NotNull

@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 Pumnya Alexander

\*/

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 buffer - array of container elements

\*/

MyIterator(final String[] buffer) {

lowerBound = -1;

higherBound = buffer.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];

}

}

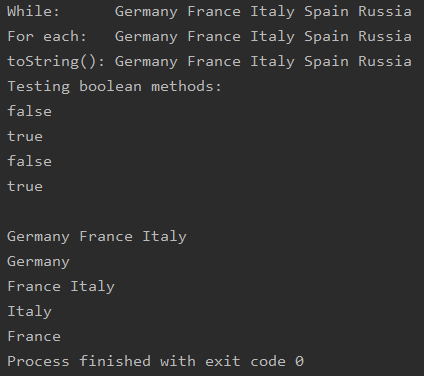
higherBound--;

}

}

}

ВАРІАНТИ ВИКОРИСТАННЯ

  
Рисунок 1 – Результат роботи програми

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

ВИСНОВКИ

При виконанні лабораторної роботи набуто практичних навичок щодо створення власних класів контейнерів та ітераторів. Досліджено принцип роботи контейнерів та ітераторів.