

Тема: Розробка власних контейнерів. Ітератори

Мета:

- Набуття навичок розробки власних контейнерів.
- Використання ітераторів.

1 ВИМОГИ

1.1 Розробник

Інформація про розробника:

- Федюкіна Поліна Олегівна
- КІТ-119Д;
- 23 варіант.

1.2 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.

1.3 Задача

Поновити л.р.3

2 ОПИС ПРОГРАМИ

2.1 Засоби ООП

Створення власного класу контейнеру для зберігання даних під час роботи. Для коректної роботи

2.2 Ієрархія та структура класів

Клас “Helper” виконує роль допоміжного класу який виконує неосновні завдання наприклад : виведення результату або перевірка символів на відповідність. Клас-контейнер «Container» зберігає всі дані в масиві та надає доступ до даних .

Методи класу : додавання , видалення , пошук, кількість елементів.

Ітератор «Iterator» - засіб послідовного доступу до вмісту контейнера; він є інтелектуальним вказівником, що «знає» як отримати доступ до елементів контейнера;

2.3 Важливі фрагменти програми

```
public class Container {  
    static int size = 0;  
  
    String[] m_data = new String[256];  
  
    public String toString()  
    {  
        if(size == 0) return null;  
  
        String temp=new String();  
        for(int i = 0; i < size; i++)  
            temp += m_data[i];  
        return temp;  
    }  
  
    public void add(String string)  
    {  
        if (size + 1 >= 255) return;  
        String temp=new String();  
        for (int i=0;i<string.length();i++)  
        {  
            if((char)string.charAt(i)!=32)  
                temp+=string.charAt(i);  
            else{  
                m_data[size++]=temp+" ";  
                temp=new String();  
            }  
        }  
        m_data[size++]=temp+" ";  
    }  
  
    public void clear()  
    {  
        while (size!=0)  
            iterator().remove();  
    }  
}
```

```

public boolean remove (String string)
{
    if (size ==0) return false;
    for (int i=0;i<size;i++)
    if (m_data[i]==string)
    {
        for (; i < size-1; i++)
            m_data[i]=m_data[i+1];
        m_data[--size] = null;
        return true;
    }

    return false;
}

public int size()
{
    return size;
}
public boolean contains(String string)
{
    if (size ==0) return false;
    for (int i=0;i<size;i++)
        if (m_data[i]==string)
            return true;

    return false;
}
Object[] toArray()
{
    return m_data;
}
boolean containsAll(Container container)
{
    if(container.size() == this.size())
        if(container.toString() == this.toString())
            return true;
        return false;
}

public Iterator<String> iterator()
{
    return new m_Iterator();
}

public class m_Iterator implements Iterator<String>
{

    int index = 0;
    public boolean hasNext() {
        if(index<size)
            return true;
        return false;
    }
    public String next()
    {
        return m_data [index++];
    }
    public void remove()
    {
        for (int i=index; i < size-1; i++)

```

```

        m_data[i]=m_data[i+1];
        m_data[--size] = null;

    }

}

public interface Iterator<E> {

    public boolean hasNext();
    public String next();
    public void remove();
}

```

3 ВАРИАНТИ ВИКОРИСТАННЯ

Після введення даних данні записуються у контейнер за допомогою команди add. Після в функцію редагування передаються дані контейнеру за допомогою команди toString(). Відредаговані данні перезаписуються у контейнер. Результат виводиться на екран звичайним способом.

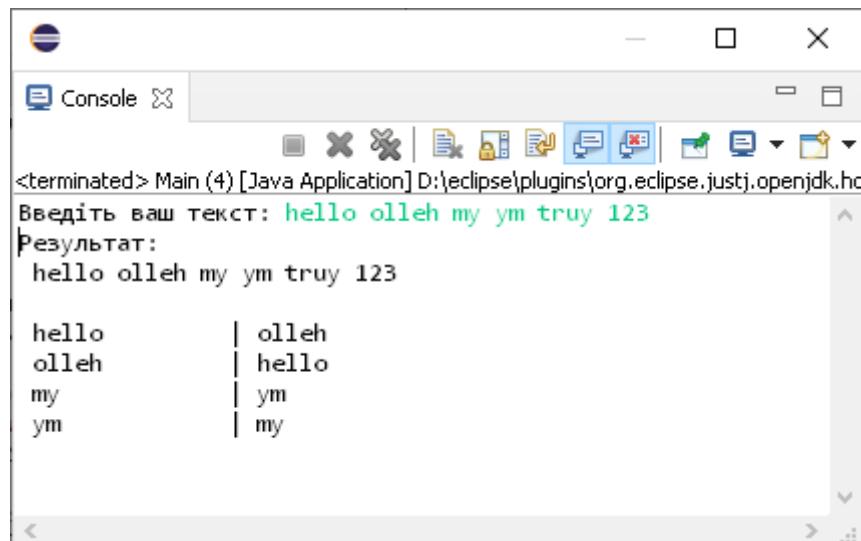


Рисунок 1 – результат редагування тексту

ВИСНОВКИ

Під час виконання лабораторної роботи було набуто навички з розробки власних контейнерів та створення і використання ітераторів.