Лабораторна робота №5.

РОЗРОБКА ВЛАСНИХ КОНТЕЙНЕРІВ. ІТЕРАТОРИ

**Мета роботи:**

* Набуття навичок розробки власних контейнерів.
* Використання ітераторів.

**1. Завдання / вимоги до лабораторної роботи:**

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.

**1.1 Розробник:**

*студент Литвин Ігнатій Ігоревич; КІТ-26А; Варіант №7*

**2. Розробка програми**

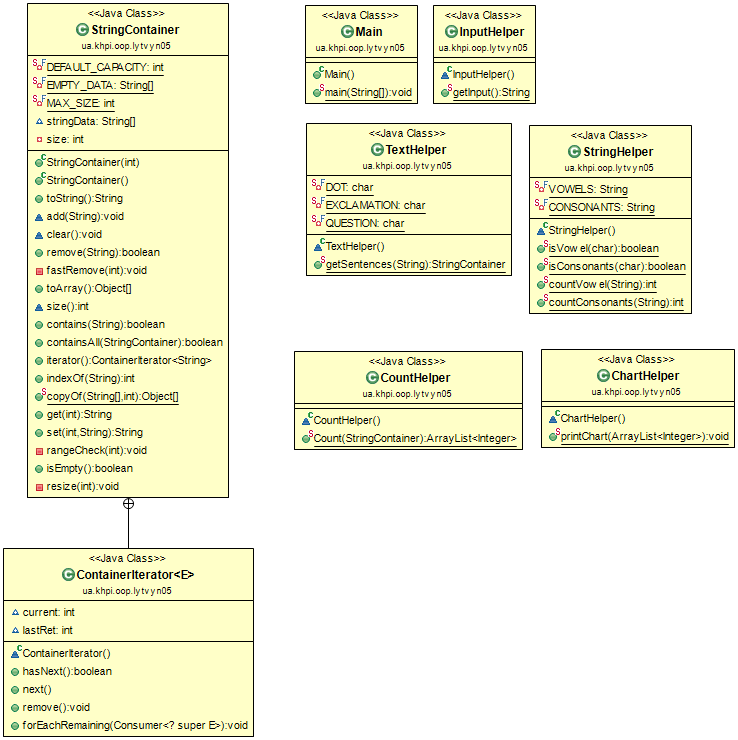
**2.1 Ієрархія та структура класів**

Рисунок 1 « Ієрархія та структура класів»

**2.2 Опис програми**

Програма реалізована у вигляді консольного вікна з послідовним виконанням завдання.

Основне призначення: використовуючи введений текст, визначати та виводити у вигляді таблиці, яких літер (голосних чи приголосних) більше в кожному реченні тексту.

Програма працює лише з текстом написаним на латинкою. Для обробки даних використовуються класи-утиліти. Регулярних вирази не використовуються при виконанні завдання.

Для збереження початкових даних завдання л.р. №3 у вигляді масиву рядків з можливістю додавання, видалення і зміни елементів було розроблено клас-контейнер, що ітерується. У контейнері було реалізовані 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.
* **public int indexOf(String string)** повертає індекс першого входження вказаного елемента в цей контейнер, або -1, якщо цей контейнер не містить елемента.
* **public String get(int index)** повертає елемент у зазначеній позиції в цьому контейнері
* **public String set(int index, String string)** замінює елемент у вказаному положенні в цьому контейнері на вказаний елемент.
* **private void rangeCheck(int index)** перевіряє, чи вказаний індекс знаходиться в діапазоні.
* **public boolean isEmpty()** перевіряє чи не містить контейнер елементів
* **private void resize(int minCapacity)** збільшує ємніть, щоб забезпечити щонайменше число елементів, вказані аргументом мінімальної ємності.

**2.3 Важливі фрагменти програми**

/\*\*

\* Контейнер для зберігання елементів типу String

\*

\* **@author** student Lytvyn I.I. KIT-26A

\*/

@SuppressWarnings("serial")

**public** **class** StringСontainer **implements** Iterable<String>, Serializable {

/\*\*

\* Початкова ємність за замовчуванням.

\*/

**private** **static** **final** **int** ***DEFAULT\_CAPACITY*** = 10;

/\*\*

\* Пустий порожній екземпляр масиву, який використовується для пустих

\* екземплярів.

\*/

**private** **static** **final** String[] ***EMPTY\_DATA*** = {};

/\*\*

\* Максимальний розмір масиву.

\*/

**private** **static** **final** **int** ***MAX\_SIZE*** = Integer.***MAX\_VALUE*** - 8;

/\*\*

\* Буферний масив, в якому зберігаються контейнерні елементи. Ємність

\* контейнера - це довжина цього масиву.

\*/

**transient** String[] stringData;

/\*\*

\* Розмір контейнера (кількість елементів, що містяться в ньому).

\*/

**private** **int** size;

/\*\*

\* Створює порожній контейнер із вказаною початковою ємністю.

\*

\* **@param** initialCapacity

\* Початкова ємність контейнера

\* **@throws** IllegalArgumentException

\* якщо вказана початкова ємність є негативною

\*/

**public** StringСontainer(**int** initialCapacity) {

**if** (initialCapacity > 0) {

**this**.stringData = **new** String[initialCapacity];

} **else** **if** (initialCapacity == 0) {

**this**.stringData = ***EMPTY\_DATA***;

} **else** {

**throw** **new** IllegalArgumentException(

"Illegal Capacity: " + initialCapacity);

}

}

/\*\*

\* Створює порожній контейнер з початковою ємністю.

\*/

**public** StringСontainer() {

**this**.stringData = ***EMPTY\_DATA***;

}

/\*\*

\* Конвертує контейнер до String.

\*/

@Override

**public** String toString() {

**return** getClass().getName() + "@" + Integer.*toHexString*(hashCode());

}

/\*\*

\* Додає вказаний елемент до кінця цього контейнера.

\*

\* **@param** string

\* елемент, що слід додати

\*/

**void** add(String string) {

resize(size + 1);

stringData[size++] = string;

}

/\*\*

\* Видаляє всі елементи з цього контейнера.

\*/

**void** clear() {

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

stringData[i] = **null**;

}

size = 0;

}

/\*\*

\* Видаляє перше входження вказаного елемента з цього контейнер, якщо є.

\* Якщо контейнер не містить елемент, він не змінився.

\*

\* **@param** string

\* елемент, який слід видалити з цього контейнера, якщо він

\* присутній

\* **@return** <tt>true</tt> якщо цей контейнер містить вказаний елемент

\*/

**public** **boolean** remove(String string) {

**if** (string == **null**) {

**for** (**int** index = 0; index < size; index++)

**if** (stringData[index] == **null**) {

fastRemove(index);

**return** **true**;

}

} **else** {

**for** (**int** index = 0; index < size; index++)

**if** (string.equals(stringData[index])) {

fastRemove(index);

**return** **true**;

}

}

**return** **false**;

}

/\*

\* Приватний метод видалення, який пропускає перевірку межі та не повертає

\* видаленого значення

\*/

**private** **void** fastRemove(**int** index) {

**int** numMoved = size - index - 1;

**if** (numMoved > 0)

System.*arraycopy*(stringData, index + 1, stringData, index,

numMoved);

stringData[--size] = **null**;

}

/\*\*

\* Повертає масив, що містить всі елементи у цьому контейнері в правильній

\* послідовності (від першого до останнього елемента).

\*

\* **@return** масив, що містить всі елементи у цьому контейнері в правильній

\* послідовності

\*/

**public** Object[] toArray() {

**return** *copyOf*(stringData, size);

}

/\*\*

\* Повертає кількість елементів у цьому контейнері.

\*

\* **@return** кількість елементів у цьому контейнері.

\*/

**int** size() {

**return** size;

}

/\*\*

\* Повертає <tt>true</tt> якщо цей контейнер містить вказаний елемент.

\*

\* **@param** string

\* елемент, наявність якого в цьому контейнері має бути

\* перевірено

\* **@return** <tt>true</tt> if this list contains the specified element

\*/

**public** **boolean** contains(String string) {

**return** indexOf(string) >= 0;

}

/\*\*

\* Повертає <tt>true</tt> якщо цей контейнер містить вказані елементи

\* отриманий контейнер

\*

\* **@param** conteiner

\* контейнер, наявність яких елементів буде перевірено

\* **@return** <tt>true</tt> якщо цей контейнер містить вказані елементи

\*/

**public** **boolean** containsAll(StringСontainer container) {

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

**if** (!contains(container.get(i)))

**return** **false**;

**return** **true**;

}

@Override

**public** ContainerIterator<String> iterator() {

**return** **new** ContainerIterator<String>();

}

/\*\*

\* Повертає індекс першого входження вказаного елемента в цей контейнер, або

\* -1, якщо цей контейнер не містить елемента.

\*/

**public** **int** indexOf(String string) {

**if** (string == **null**) {

**for** (**int** i = 0; i < size; i++)

**if** (stringData[i] == **null**)

**return** i;

} **else** {

**for** (**int** i = 0; i < size; i++)

**if** (string.equals(stringData[i]))

**return** i;

}

**return** -1;

}

**public** **static** Object[] copyOf(String[] original, **int** newLength) {

Object[] copy = (String[]) **new** Object[newLength];

System.*arraycopy*(original, 0, copy, 0,

Math.*min*(original.length, newLength));

**return** copy;

}

/\*\*

\* Повертає елемент у зазначеній позиції в цьому контейнері.

\*

\* **@param** index

\* індекс повернутого елементу

\* **@return** елемент у зазначеній позиції в цьому контейнері

\* **@throws** IndexOutOfBoundsException

\*/

**public** String get(**int** index) {

rangeCheck(index);

**return** stringData[index];

}

/\*\*

\* Замінює елемент у вказаному положенні в цьому контейнері з вказаний

\* елемент.

\*

\* **@param** index

\* індекс елемента, який потрібно замінити

\* **@param** string

\* елемент, який зберігатиметься у вказаній позиції

\* **@return** елемент, що був раніше в зазначеній позиції

\* **@throws** IndexOutOfBoundsException

\*/

**public** String set(**int** index, String string) {

rangeCheck(index);

String oldValue = stringData[index];

stringData[index] = string;

**return** oldValue;

}

/\*\*

\* Перевіряє, чи вказаний індекс знаходиться в діапазоні. Якщо ні, кидає

\* відповідний runtime exception. Цей метод "не" перевіряє, чи є індекс

\* негативним.

\*/

**private** **void** rangeCheck(**int** index) {

**if** (index >= size)

**throw** **new** IndexOutOfBoundsException(

"Index: " + index + ", Size: " + size);

}

/\*\*

\* Повертає <tt> true </ tt> якщо цей контейнер не містить елементів.

\*

\* **@return** <tt>true</tt> якщо цей контейнер не містить елементів

\*/

**public** **boolean** isEmpty() {

**return** size == 0;

}

/\*\*

\* Збільшує ємніть, щоб забезпечити щонайменше число елементів, вказані

\* аргументом мінімальної ємності.

\*

\* **@param** minCapacity

\* бажана мінімальна ємність

\*/

**private** **void** resize(**int** minCapacity) {

**if** (stringData == ***EMPTY\_DATA***) {

minCapacity = Math.*max*(***DEFAULT\_CAPACITY***, minCapacity);

}

**if** (minCapacity - stringData.length > 0) {

**int** oldCapacity = stringData.length;

**int** newCapacity = oldCapacity + (oldCapacity >> 1);

**if** (newCapacity - minCapacity < 0) {

newCapacity = minCapacity;

}

**if** (newCapacity - ***MAX\_SIZE*** > 0) {

**if** (minCapacity < 0) // Переповнення

**throw** **new** OutOfMemoryError();

newCapacity = (minCapacity > ***MAX\_SIZE***) ? Integer.***MAX\_VALUE***

: ***MAX\_SIZE***;

}

String[] newData = **new** String[minCapacity];

System.*arraycopy*(stringData, 0, newData, 0, size);

stringData = newData;

}

}

/\*\*

\* Ітератор для контейнеру StringСontainer

\*

\* **@author** student Lytvyn I.I. KIT-26A

\*/

**class** ContainerIterator<E> **implements** Iterator<E> {

**int** current; // Індекс елементу, що повернеться

**int** lastRet = -1; // Індекс останього елементу, що повертався; -1 якщо

// такий відсутній

@Override

**public** **boolean** hasNext() {

**return** current != size;

}

@SuppressWarnings("unchecked")

@Override

**public** E next() {

**int** i = current;

**if** (i >= size)

**throw** **new** NoSuchElementException();

Object[] elementData = StringСontainer.**this**.stringData;

current = i + 1;

**return** (E) elementData[lastRet = i];

}

@Override

**public** **void** remove() {

**if** (lastRet < 0)

**throw** **new** IllegalStateException();

**try** {

StringСontainer.**this**.fastRemove(lastRet);

current = lastRet;

lastRet = -1;

} **catch** (IndexOutOfBoundsException ex) {

**throw** **new** RuntimeException(ex.toString());

}

}

@SuppressWarnings("unchecked")

@Override

**public** **void** forEachRemaining(Consumer<? **super** E> consumer) {

Objects.*requireNonNull*(consumer);

**final** **int** size = StringСontainer.**this**.size;

**int** i = current;

**if** (i >= size) {

**return**;

}

**final** Object[] elementData = StringСontainer.**this**.stringData;

**if** (i >= elementData.length) {

**throw** **new** ConcurrentModificationException();

}

**while** (i != size) {

consumer.accept((E) elementData[i++]);

}

current = i;

lastRet = i - 1;

}

}

}

1. **РЕЗУЛЬТАТ РОБОТИ**

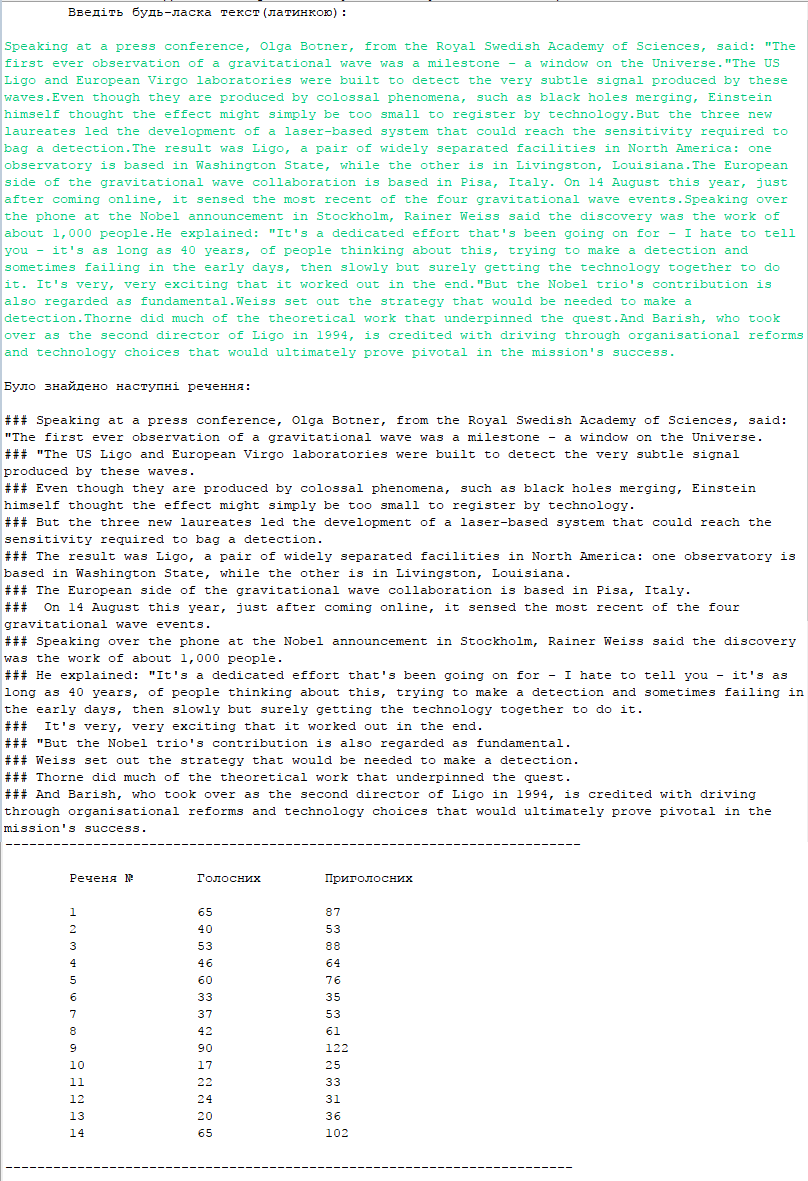
Для налагодження роботи програми було успішно проведено її тестування.

Рисунок 2 "Результат роботи програми"

**ВИСНОВКИ**

*Створено і налагоджено програму, що повністю виконую поставлене індивідуальне завдання та відповідає вимогам.*

*Було отримано і вдосконалено навички у розробці власних контейнерів та у використані ітераторів.*