Лабораторна робота №11

Виконав:

ст. групи КН-107

Древницький Ю.А.

Прийняв :

Асистент

кафедри СШІ

Швороб І.Б.

Обов’язковий розділ - Зауваження

Тема:

Параметризація в Java. Обробка параметризованих контейнерів

Мета:

- Вивчення принципів параметризації в *Java* .
- Розробка параметризованих класів та методів.
- Розширення функціональності параметризованих класів.

Вимоги:

1. Створити власний клас-контейнер, що параметризується (Generic Type), (docs.oracle.com/javase/tutorial/java/generics/types.html) на основі зв’язних списків для реалізації колекції domain-об’єктів з лабораторної роботи №10 (Прикладні задачі. Список №2. 20 варіантів)

2. Для розроблених класів-контейнерів забезпечити можливість використання їх об’єктів у циклі foreach в якості джерела даних.

3. Забезпечити можливість збереження та відновлення колекції об’єктів:

1) за допомогою стандартної серіалізації;

2) не використовуючи протокол серіалізації.

4. Продемонструвати розроблену функціональність: створення контейнера, додавання елементів, видалення елементів, очищення контейнера,перетворення у масив, перетворення у рядок, перевірку на наявність елементів.

5. Забороняється використання контейнерів (колекцій) з Java Collections Framework - docs.oracle.com/javase/8/docs/technotes/guides/collections/

6. Розробити параметризовані методи (Generic Methods - docs.oracle.com/javase/tutorial/java/generics/methods.html) для обробки колекцій об’єктів

згідно (Прикладні задачі. Список №2. 20 варіантів).

7. Продемонструвати розроблену функціональність (створення, управління та обробку власних контейнерів) в діалоговому та автоматичному режимах.

a. Автоматичний режим виконання програми задається параметром командного рядка -auto . Наприклад, java ClassName -auto .

b. В автоматичному режимі діалог з користувачем відсутній, необхідні данні генеруються, або зчитуються з файлу

Розробник:

Студент академічної групи КН-107

Древницький Юрій Анатолійович

Варіант 18

Розв'язок:

```
package company;

import java.io.Serializable;
import java.util.Iterator;

import static java.lang.String.valueOf;

public class Listt<E> implements LinkedRealize<E>,Iterable<E>,DescIterator<E>,Serializable {

   private int size = 0;
   private Node<E> firstNode;
   private Node<E> lastNode;

   public Listt()
   {
      lastNode = new Node<>(null,firstNode,null);
      firstNode = new Node<>(null,null,lastNode);
   }

   @Override
```

```
public void addFirst(E e) {
   Node<E> next = firstNode;
   next.setCurrentElement(e);
   firstNode = new Node<>(null,null,next);
   next.setPrevElement(firstNode);
   size++;

}

@Override
public void addLast(E e) {
   Node<E> prev = lastNode;
   prev.setCurrentElement(e);
   lastNode = new Node<>(null,prev,null);
   prev.setNextElement(lastNode);
   size++;

}

@Override
public void delEl(int counter) {
   Node<E> target = firstNode.getNextElement();
   Node<E> preTarget = firstNode.getNextElement();
   Node<E> nextTarget = firstNode.getNextElement();

   //finding
   for (int i = 0; i < counter; i++)
   {
      target = getElement(target);
   }
   preTarget = target.getPrevElement();
   nextTarget = target.getNextElement();

   //deleting
   preTarget.setNextElement(nextTarget);
   nextTarget.setPrevElement(preTarget);

   size--;
}

@Override
public int getSize() {
   return size;
}


@Override
public E getElementByIndex(int counter) {
   Node<E> target = firstNode.getNextElement();
```

```
    for (int i = 0; i < counter; i++)
    {
        target = getElement(target);
    }

    return target.getCurrentElement();
}
private Node<E> getElement(Node<E> current)
{

    return current.getNextElement();
}


@Override
public String toStringg(Listt listt) {

    String sum = "";

    for (int i=0; i<size; i++)
    {
        sum = sum.concat(valueOf(listt.getElementByIndex(i)));
    }

    return sum;
}

@Override
public void clear(Listt listt) {

    firstNode.nextElement = lastNode;
    lastNode.prevElement = firstNode;

    size = 0;
}

@Override
public Object[] toArray() {

    Node<E> target = firstNode.getNextElement();

    Object[] listElements = new Object[size];
    for (int i = 0; i < size; i++)
    {
        listElements[i] = target.getCurrentElement();
        target = getElement(target);
    }
```

```
    return listElements;
}

@Override
public boolean contains(String string) {

    Node<E> target = firstNode.getNextElement();

    for (int i = 0; i < size; i++)
    {
      if ( valueOf(target.getCurrentElement()).equals(string) )
      {
          return true;
      }

       target = getElement(target);
    }

    return false;
}

@Override
public boolean containsAll(Listt listt) {

    Node<E> target = firstNode.getNextElement();

    for (int i = 0; i < size; i++)
    {
      if ( !target.getCurrentElement().equals(listt.getElementByIndex(i)) )
      {
          return false;
      }

      target = getElement(target);
    }


    return true;
}


@Override
public Iterator<E> iterator() {
    return new Iterator<E>() {
      int counter = 0;
      @Override
      public boolean hasNext() {
          return (counter<size);
      }
```

```
        @Override
        public E next() {
            return getElementByIndex(counter++);
        }

    };
}

@Override
public Iterator<E> descendingIterator() {
    return new Iterator<E>() {
        int counter = size -1;

        @Override
        public boolean hasNext() {
            return (counter >= 0);
        }

        @Override
        public E next() {
            return getElementByIndex(counter--);
        }
    };
}


/**
 * NODE
 */
private class Node<E> implements Serializable {

    private E currentElement;
    private Node<E> nextElement;
    private Node<E> prevElement;

    private Node(E currentElement, Node<E> prevElement, Node<E> nextElement) {
        this.currentElement = currentElement;
        this.prevElement = prevElement;
        this.nextElement = nextElement;

    }


    /**
     * getters
     */
    public E getCurrentElement() {
        return currentElement;
```

```
    }

    public Node<E> getPrevElement() {
       return prevElement;
    }

    public Node<E> getNextElement() {
       return nextElement;
    }

    /**
     * setters
     */
    public void setCurrentElement(E currentElement) {
       this.currentElement = currentElement;
    }

    public void setPrevElement(Node<E> prevElement) {
       this.prevElement = prevElement;
    }

    public void setNextElement(Node<E> nextElement) {
       this.nextElement = nextElement;
    }
  }

}
```

```
package company;


public interface LinkedRealize<E> {

   void  addFirst(E e);
   void  addLast(E e);
   void delEl(int counter);
   int getSize();
   E getElementByIndex(int counter);

   // lab 9 specials

   String toStringg(Listt listt);
   void clear(Listt listt);
   Object[] toArray();
   boolean contains(String string);
```

```
    boolean containsAll(Listt listt);
}
```

```
package company;

import java.util.Iterator;

public interface DescIterator<E> {

    public Iterator<E> descendingIterator();



}
```

Висновок:

Отже, я навчився розробляти параметризовані контейнери, класи та методи за допомогою дженериків.