Тема: Параметризація в Java
Мета: Вивчення принципів параметризації в Java. Розробка параметризованих класів та методів.

1 ВИМОГИ

1.1 Розробник

Інформація про розробника:
- Когутенко Олександр Олексійович;
- КІТ-119Д;
- 11 варіант.

1.2 Загальне завдання

1. Створити власний клас-контейнер, що параметризується (Generic Type), на основі зв'язних списків для реалізації колекції domain-об’єктів лабораторної роботи №7.
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.

2 ОПИС ПРОГРАМИ

2.1 Засоби ООП

Використовується наслідування, інтерфейс, поліморфізм.

2.2 Ієрархія та структура класів

Використовую 10 классів: Array, ArrayIterator, HelperClass, InteractiveConsole, Start, SaveArray, Date, Shops, ConsoleFile, HelperClassLink.

- Array використовую як інтерфейс для класу контейнеру.
- ArrayIterator використовую як особисту реалізацію ітератора.
- HelperClass допоміжний класс для розрахунків.
- InteractiveConsole клас для налагодженого спілкування програми з користувачем та методами нестандартних протоколів серіалізації.
- Start клас який має точку входу у програму.
- SaveArray клас контейнер який має все необхідні методи маніпулятори.
- Date клас використовуется для збереження дати.

- Shops клас приклодної галузі.
- ConoleFile клас за допомогою якого користувач може наблюдати за відображенням вмісту каталогів.
- HelperClassLink використовуеться за для реалізації зв'язного списку з методами стандартних методів серіалізації.

2.3 Важливі фрагменти програми

```
package ua.khpi.oop.kogutenko09;

import java.beans.XMLDecoder;
import java.beans.XMLEncoder;
import java.io.*;
import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 * The type Helper class link.
 *
 * @param <T> the type parameter
 */
public class HelperClassLink<T> implements Iterable<T> {
	private Node firstElem;
	private Node lastElem;
	private int N;

    /**
     * Instantiates a new Helper class link.
     */
    public HelperClassLink() {
		firstElem = null;
		lastElem = null;
		N = 0;
	}
////////////////////////////////////////////////////////
	private class Node {
		private T data;
		private Node next;

		/**
		 * Instantiates a new Node.
		 *
		 * @param data the data
		 * @param next the next
		 */
		public Node(T data, Node next)
```

```
        {
            this.data = data;
            this.next = next;
        }

        /**
         * Instantiates a new Node.
         */
        public Node()
        {
        }

        /**
         * Data of elem t.
         *
         * @return the t
         */
        public T dataOfElem()
        {
            return data;
        }
    }
////////////////////////////////////////////////////////
    private class LinkedListIterator implements Iterator<T> {
        private Node current = firstElem;

        public T next() {
            if (!hasNext())
            {
                throw new NoSuchElementException();
            }
            T item = current.data;
            current = current.next;
            return item;
        }

        public boolean hasNext()
        {
            return current != null;
        }

        public void remove()
        {
            throw new UnsupportedOperationException();
        }
```

```
    }

  /**
   * Get t.
   *
   * @param index the index
   * @return the t
   */
////////////////////////////////////////////////////
    public T get(int index) {
        System.out.println("\n\"get\" from HelperClassLink\n");
        if (index < 0 || index >= N) {
            throw new IndexOutOfBoundsException();
        }
        Node result = firstElem;
        for (int i = 0; i < index; i++) {
            result = result.next;
        }

        return result.data;
    }

  /**
   * Add.
   *
   * @param item the item
   */
  public void add(T item) {
        System.out.println("\n\"add\" from HelperClassLink\n");
        if (item == null)
        {
            throw new NullPointerException("The first argument for addLast() is
null.");
        }
        if (!isEmpty())
        {
            Node prev = lastElem;
            lastElem = new Node(item, null);
            prev.next = lastElem;
        }
        else
            {
                lastElem = new Node(item, null);
                firstElem = lastElem;
            }
```

```
        N++;
    }

/**
 * Remove boolean.
 *
 * @param index the index
 * @return the boolean
 */
public boolean remove(int index) {
        System.out.println("\n\"remove\" from HelperClassLink\n");
        if (index < 0 || index > N - 1) {
            throw new IllegalArgumentException();
        }
        if (index == 0) {
            firstElem = firstElem.next;
        } else {
            Node node = findNodeBeforeByIndex(index);
            Node tmp = findByIndex(index);
            node.next = tmp.next;
        }
        N--;
        return false;
    }

/**
 * Remove element boolean.
 *
 * @param element the element
 * @return the boolean
 */
public boolean removeElement(T element) {
        System.out.println("\n\"removeElement\" from HelperClassLink\n");

        if (N == 0) {
            return false;
        } else if (N == 1) {
            firstElem = null;
            lastElem = null;
            N = 0;
            return true;
        }

        Node nodeBefore = findNodeBefore(element);
```

```
        if (nodeBefore.data == null) {
            firstElem = firstElem.next;
            N--;
            return true;
        } else if (nodeBefore != null) {
            if (lastElem.data == element) {
                nodeBefore.next = null;
                lastElem = nodeBefore;
            } else {
                nodeBefore.next = nodeBefore.next.next;
            }
            N--;
            return true;
        }
        return false;
    }

/**
 * Size int.
 *
 * @return the int
 */
public int size(){
        System.out.println("\n\"size\" from HelperClassLink\n");
         return N;
    }

/**
 * Is empty boolean.
 *
 * @return the boolean
 */
public boolean isEmpty() {
        System.out.println("\n\"isEmpty\" from HelperClassLink\n");
        return N == 0;
    }

public Iterator<T> iterator(){
        System.out.println("\n\"iterator\" from HelperClassLink\n");
        return new LinkedListIterator();
    }

@Override
public String toString() {
        System.out.println("\n\"toString\" from HelperClassLink\n");
```

```
        StringBuilder s = new StringBuilder();
        for (T item : this)
            s.append(item + " ");
        return s.toString();
    }

/**
 * Print list.
 */
public void printList(){
        System.out.println("\n\"printList\" from HelperClassLink\n");
        String str = "";
        for(T item : this)
        {
            str += item.toString();
        }
        System.out.println(str);
    }

private Node findByIndex(int index) {
        System.out.println("\n\"findByIndex\" from HelperClassLink\n");
        if (index < 0 || index > N - 1) {
            throw new IndexOutOfBoundsException();
        }
        int tmpIndex = 0;
        if (firstElem == null) {
            throw new IndexOutOfBoundsException();
        }

        if (index == 0) {
            return firstElem;
        }

        Node node = firstElem;
        while (node.next != null) {
            node = node.next;
            tmpIndex++;
            if (tmpIndex == index) {
                return node;
            }
        }
        throw new IndexOutOfBoundsException();
    }
private Node findNodeBefore(T value) {
        System.out.println("\n\"findNodeBefore\" from HelperClassLink\n");
```

```
        if (firstElem.data == value) {
            Node res = new Node(firstElem.data, firstElem.next);
            return res;
        }

        Node node = firstElem;
        while (node.next != null) {
            if (node.next == value) {
                return node;
            }
            node = node.next;
        }
        return null;
    }

  private Node findNodeBeforeByIndex(int index) {
        System.out.println("\n\"findNodeBeforeByIndex\" from
HelperClassLink\n");
        if (index <= 0 || index > N - 1) {
            return null;
        }

        int count = 0;
        Node node = firstElem;
        while (node.next != null) {
            if (count == index - 1) {
                return node;
            }
            count++;
            node = node.next;
        }
        return null;
    }

  /**
   * Serialization xml.
   */
  public void serializationXML(){
        System.out.println("\n\"serializationXML\" from HelperClassLink\n");
            //File file = ConsoleFile.MenuFillOut();
        try{
            XMLEncoder encoder = new XMLEncoder(
                new BufferedOutputStream(
                    new FileOutputStream( ConsoleFile.MenuFillOut(".xml"))));
```

```
            encoder.writeObject(this.size());

            for(T shop : this) {
                encoder.writeObject(shop);
                //encoder.writeObject(shop.getDescription());
            }

            encoder.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

/**
 * Serialization bin.
 */
public void serializationBIN(){
        File file = ConsoleFile.MenuFillOut(".bin");///pathname
        try {
            FileOutputStream fos = new FileOutputStream(file);
            ObjectOutputStream oos = new ObjectOutputStream(fos);

            oos.writeObject(this.size());
            System.out.println("size :" + this.size());
            for (T el : this)
            {
                oos.writeObject(el);
            }
        }
        catch(FileNotFoundException e)
        {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }

/**
 * Deseriallztion bin.
 */
public void deserializationBIN(){
        File file = ConsoleFile.MenuFillIn(".bin");///pathname
```

```
        try {
            FileInputStream fis = new FileInputStream(file);///pathname
            ObjectInputStream ois = new ObjectInputStream(fis);
            Integer count = ois.readInt();
            for(int i = 0; i < count; i++)
            {
                this.add((T) ois.readObject());
            }

        }
        catch(FileNotFoundException e) {e.printStackTrace();}
        catch (IOException e) {e.printStackTrace();    }
        catch (ClassNotFoundException e) {e.printStackTrace(); }
    }

/**
 * Deserializtion xml.
 */
public void deserializationXML(){
        try{
            XMLDecoder decoder = new XMLDecoder(
                    new BufferedInputStream(
                            new FileInputStream(ConsoleFile.MenuFillIn(".xml"))
                    )
            );

            int count = (int) decoder.readObject();

            for(int i = 0; i < count; i++)
            {
                T shops = (T)decoder.readObject();
                Object obj = decoder.readObject();
                //shops.setDescription((Map<String, String>) obj);
                this.add(shops);
            }
            decoder.close();

        }
        catch(FileNotFoundException e) {e.printStackTrace();}
    }



public Array<T> toOwnArray() {
    Array<T> save = new SaveArray<>();
```

```
    for(int i = 0; i > N; i++) {
        save.add(this.get(i));
    }
    return save;
  }
}
```

3 ВАРІАНТИ ВИКОРИСТАННЯ

```
1 / input       -    input from file
2 / show        -    show information about shops
3 / add         -    add one shop
4 / remove      -    remove one shop
5 / switch      -    switch to another list
0 / exit        -    exit and save data

user@user: 1


create collection from...choice command:
 1 - location
 2 - directory files
 3 - go to another directory
 4 - out of directory
 5 - choice file to create collection
 6 - go to... :
```

Рисунок 9.1 - початок роботи

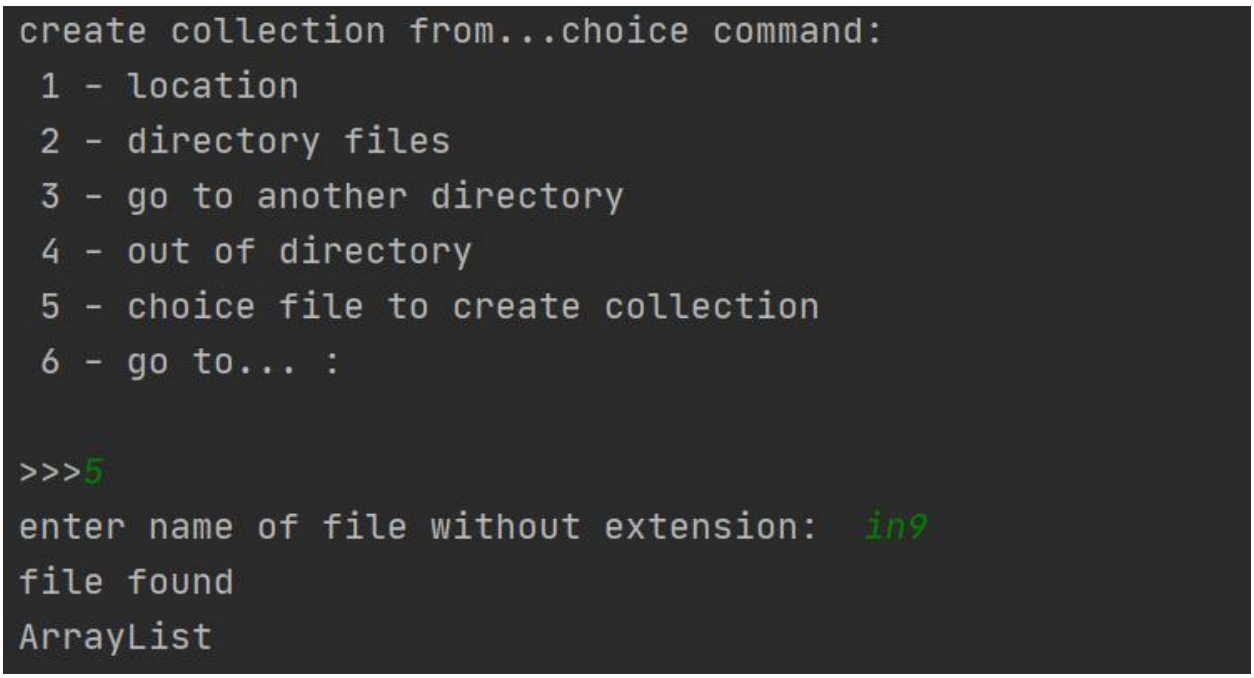

Рисунок 9.2 - створення списку через серыалызацію

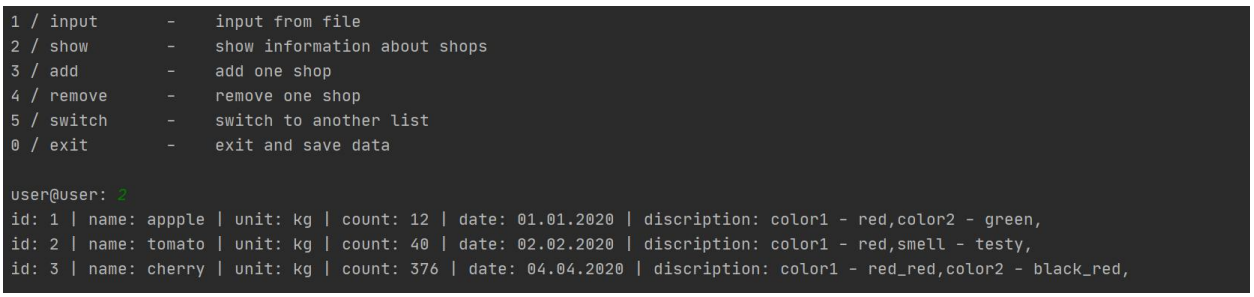

Рисунок 9.3 - перевирка списку через вивід на екран

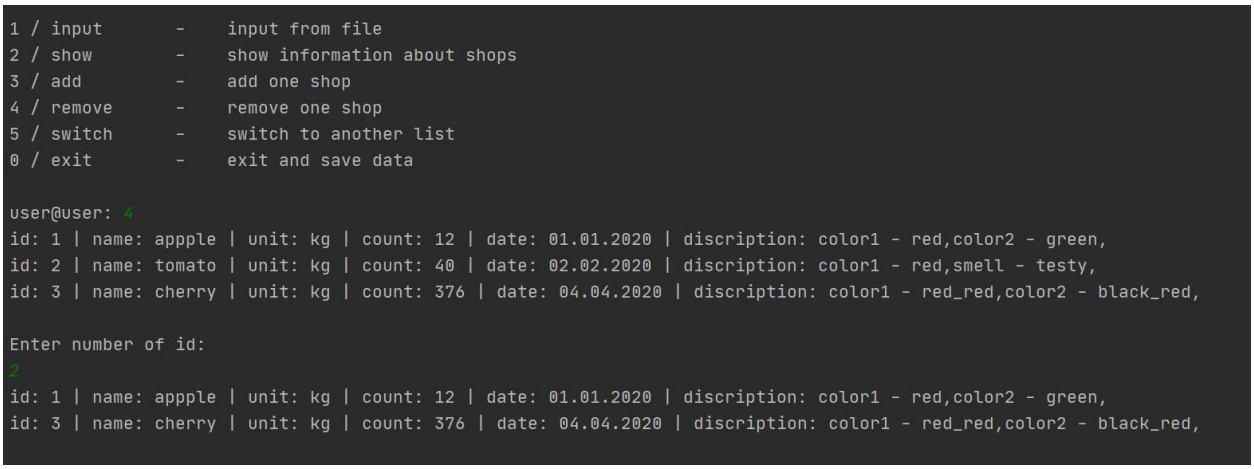

Рисунок 9.4 - видалення зі списку елемента

ВИСНОВКИ

Вивчив принципи параметризації в Java. Розробив параметризовані класи та методи.