Звіт

Автор: Пумня О., КІТ118Б

Дата: 01.02.2020

Лабораторна робота №9

ПАРАМЕТРИЗАЦІЯ В JAVA

**Мета.** Вивчення принципів параметризації в Java. Розробка параметризованих класів та методів.

**Вимоги:**

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.

ОПИС ПРОГРАМИ

**Опис змінних**

GenericList<ScheduerEvent> container1 – Контейнер, параметризований класом

ObjectOutputStream oos – Для використання серіалізації

ObjectInputStream ois – Для використання серіалізації

FileOutputStream fos – Для використання XML-енкодера

FileInputStream fis – Для використання XML-декодера

**Ієрархія та структура класів**

class Pumnya09 – Точка входу в програму

class GenericList – Реалізація параметризованого контейнеру

ТЕКСТ ПРОГРАМИ

**Текст файлу Pumnya09.java**

package labs.pumnya09;

import labs.pumnya07.SchedulerEvent;

import java.beans.XMLDecoder;

import java.beans.XMLEncoder;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

public final class Pumnya09 {

private Pumnya09() {

}

/\*\*

\* Точка входа, главный метод.

\* @param args - аргументы главного метода

\* @throws IOException - при неудачной

\* работе с файлами

\* @throws ClassNotFoundException - при

\* отсутствии необходимого класса

\*/

public static void main(final String[] args)

throws IOException, ClassNotFoundException {

GenericList<SchedulerEvent> container1 = new GenericList<>();

container1.pushBack(SchedulerEvent.generate(true));

container1.pushFront(SchedulerEvent.generate(false));

container1.pushBack(SchedulerEvent.generate());

System.out.println("Данные: ");

System.out.print(container1.toString());

System.out.println("Serialization...");

ObjectOutputStream oos = new ObjectOutputStream(

new FileOutputStream("DataFile.dat"));

oos.writeObject(container1);

oos.close();

System.out.println("Done!\n");

System.out.println("Deserialization...");

ObjectInputStream ois = new ObjectInputStream(

new FileInputStream("DataFile.dat"));

GenericList<SchedulerEvent> container2 =

(GenericList) ois.readObject();

ois.close();

System.out.print(container2.toString());

System.out.println("Сохраненеие в XML...");

FileOutputStream fos = new FileOutputStream("Encoded.xml");

XMLEncoder xmlEncoder = new XMLEncoder(new BufferedOutputStream(fos));

xmlEncoder.writeObject(container1);

xmlEncoder.close();

System.out.println("Done!\n");

System.out.println("Чтение из XML...");

try {

FileInputStream fis = new FileInputStream("Encoded.xml");

XMLDecoder xmlDecoder = new XMLDecoder(

new BufferedInputStream(fis));

GenericList<SchedulerEvent> container3 =

(GenericList) xmlDecoder.readObject();

xmlDecoder.close();

System.out.print(container3.toString());

} catch (IOException e) {

System.out.println(e.toString());

}

}

}

**Текст файлу GenericList.java**

package labs.pumnya09;

import java.io.Serializable;

import java.util.Comparator;

import java.util.Iterator;

import java.util.NoSuchElementException;

public class GenericList<T> implements Iterable<T>, Serializable {

/\*\* Идентификационный ключ сериализации. \*/

private static final long serialVersionUID = 29066991231285950L;

/\*\*

\* Тело для каждого узла в списке.

\* @param <T> - Уникальный тип

\*/

public static class Node<T> implements Serializable {

/\*\* Ключ сериализации. \*/

private static final long serialVersionUID = 1898343768754556332L;

/\*\* Данные узла. \*/

private T data;

/\*\* Указатель на следующий узел. \*/

private Node<T> next;

/\*\* Указатель на предыдущий узел. \*/

private Node<T> prev;

/\*\*

\* Конструкрор создания узла.

\* @param prevElem - указатель

\* на предыдущий узел

\* @param elemData - данные для создания

\* @param nextElem - указатель

\* на следующий узел

\*/

Node(final Node<T> prevElem, final T elemData, final Node<T> nextElem) {

this.data = elemData;

this.prev = prevElem;

this.next = nextElem;

}

/\*\* Конструктор по умолчанию. \*/

public Node() {

this.data = null;

this.prev = null;

this.next = null;

}

/\*\*

\* Геттер поля data.

\* @return data

\*/

public T getData() {

return this.data;

}

/\*\*

\* Геттер поля next.

\* @return next

\*/

public Node<T> getNext() {

return this.next;

}

/\*\*

\* Геттер поля prev.

\* @return prev

\*/

public Node<T> getPrev() {

return this.prev;

}

/\*\*

\* Сеттер поля data.

\* @param d - data

\*/

public void setData(final T d) {

this.data = d;

}

/\*\*

\* Сеттер поля next.

\* @param n - next

\*/

public void setNext(final Node<T> n) {

this.next = n;

}

/\*\*

\* Сеттер поля prev.

\* @param p - prev

\*/

public void setPrev(final Node<T> p) {

this.prev = p;

}

}

/\*\* Указатель на первый узел списка. \*/

private Node<T> first;

/\*\* Указатель на последний узел списка. \*/

private Node<T> last;

/\*\* Количество узлов в списке. \*/

private int size;

/\*\*

\* Геттер поля size.

\* @return размер списка

\*/

public int getSize() {

return this.size;

}

/\*\*

\* Геттер поля first.

\* @return first

\*/

public Node<T> getFirst() {

return first;

}

/\*\*

\* Геттер поля last.

\* @return last

\*/

public Node<T> getLast() {

return last;

}

/\*\*

\* Сеттер поля size.

\* @param s - size

\*/

public void setSize(final int s) {

this.size = s;

}

/\*\*

\* Сеттер поля first.

\* @param f - first

\*/

public void setFirst(final Node<T> f) {

this.first = f;

}

/\*\*

\* Сеттер поля last.

\* @param l - last

\*/

public void setLast(final Node<T> l) {

this.last = l;

}

/\*\* Конструктор создания списка. \*/

public GenericList() {

this.size = 0;

}

/\*\*

\* Возврат данных первого узла.

\* @return данные первого узла

\*/

public T first() {

Node<T> f = this.first;

if (f == null) {

throw new NoSuchElementException();

} else {

return f.data;

}

}

/\*\*

\* Возврат данных последнего узла.

\* @return данные последнего узла

\*/

public T last() {

Node<T> l = this.last;

if (l == null) {

throw new NoSuchElementException();

} else {

return l.data;

}

}

/\*\*

\* Добавление узла в начало списка.

\* @param data - данные

\*/

public void pushFront(final T data) {

Node<T> f = this.first;

Node<T> newNode = new Node<T>(null, data, f);

this.first = newNode;

if (f == null) {

this.last = newNode;

} else {

f.prev = newNode;

}

this.size++;

}

/\*\*

\* Добавление узла в конец списка.

\* @param data - данные

\*/

public void pushBack(final T data) {

Node<T> l = this.last;

Node<T> newNode = new Node<T>(l, data, null);

this.last = newNode;

if (l == null) {

this.first = newNode;

} else {

l.next = newNode;

}

this.size++;

}

/\*\*

\* Добавление узла после указанного

\* (если он существует).

\* @param data - данные

\* @param curr - узел

\*/

private void pushBefore(final T data, final Node<T> curr) {

Node<T> pred = curr.prev;

Node<T> newNode = new Node<T>(pred, data, curr);

curr.prev = newNode;

if (pred == null) {

this.first = newNode;

} else {

pred.next = newNode;

}

this.size++;

}

/\*\*

\* Вставка узла в определённое место.

\* @param index - индекс

\* @param data - данные

\*/

public void insert(final int index, final T data) {

this.checkPositionIndex(index);

if (index == this.size) {

this.pushBack(data);

} else {

this.pushBefore(data, this.getNodeByIndex(index));

}

}

/\*\* Удаление первого узла. \*/

public void popFront() {

Node<T> f = this.first;

if (f == null) {

throw new NoSuchElementException();

} else {

T element = f.data;

Node<T> next = f.next;

f.data = null;

f.next = null;

this.first = next;

if (next == null) {

this.last = null;

} else {

next.prev = null;

}

this.size--;

}

}

/\*\* Удаление последнего узла. \*/

public void popBack() {

Node<T> l = this.last;

if (l == null) {

throw new NoSuchElementException();

} else {

T element = l.data;

Node<T> prev = l.prev;

l.data = null;

l.prev = null;

this.last = prev;

if (prev == null) {

this.first = null;

} else {

prev.next = null;

}

this.size--;

}

}

/\*\*

\* Удаление определённого объекта

\* (если он существует).

\* @param o - объект удаления

\* @return true, если удачно

\*/

public boolean remove(final Object o) {

Node<T> x;

if (o == null) {

for (x = this.first; x != null; x = x.next) {

if (x.data == null) {

this.unlink(x);

return true;

}

}

} else {

for (x = this.first; x != null; x = x.next) {

if (o.equals(x.data)) {

this.unlink(x);

return true;

}

}

}

return false;

}

/\*\*

\* Удаление по определённому индексу.

\* @param index - индекс

\*/

public void remove(final int index) {

this.checkElementIndex(index);

this.unlink(this.getNodeByIndex(index));

}

private boolean isElementIndex(final int index) {

return index >= 0 && index < this.size;

}

private boolean isPositionIndex(final int index) {

return index >= 0 && index <= this.size;

}

private String outOfBoundsMsg(final int index) {

return "Index: " + index + ", Size: " + this.size;

}

private void checkElementIndex(final int index) {

if (!this.isElementIndex(index)) {

throw new IndexOutOfBoundsException(this.outOfBoundsMsg(index));

}

}

/\*\*

\* Проверка индекса на выход за пределы.

\* @param index - индекс

\*/

private void checkPositionIndex(final int index) {

if (!this.isPositionIndex(index)) {

throw new IndexOutOfBoundsException(this.outOfBoundsMsg(index));

}

}

/\*\*

\* Отделяет узел от списка.

\* @param x - узел

\*/

private void unlink(final Node<T> x) {

Node<T> next = x.next;

Node<T> prev = x.prev;

if (prev == null) {

this.first = next;

} else {

prev.next = next;

x.prev = null;

}

if (next == null) {

this.last = prev;

} else {

next.prev = prev;

x.next = null;

}

x.data = null;

this.size--;

}

/\*\*

\* Получение узла по индексу.

\* @param index - индекс

\* @return узел

\*/

private Node<T> getNodeByIndex(final int index) {

Node<T> x;

int i;

if (index < this.size >> 1) {

x = this.first;

for (i = 0; i < index; ++i) {

x = x.next;

}

} else {

x = this.last;

for (i = this.size - 1; i > index; --i) {

x = x.prev;

}

}

return x;

}

/\*\*

\* Получение данные узла по индексу.

\* @param index - индекс

\* @return данные узла

\*/

public T getByIndex(final int index) {

this.checkElementIndex(index);

return this.getNodeByIndex(index).data;

}

/\*\*

\* Получить индекс узла.

\* @param o - узел

\* @return - индекс

\*/

public int indexOf(final Object o) {

int index = 0;

Node<T> x;

if (o == null) {

for (x = this.first; x != null; x = x.next) {

if (x.data == null) {

return index;

}

++index;

}

} else {

for (x = this.first; x != null; x = x.next) {

if (o.equals(x.data)) {

return index;

}

++index;

}

}

return -1;

}

/\*\*

\* Провеока на содержание узла в списке.

\* @param o - объект

\* @return true, если узел

\* находится в списке

\*/

public boolean contains(final Object o) {

return indexOf(o) >= 0;

}

/\*\* Очистка списка. \*/

public void clear() {

Node<T> next;

for (Node<T> x = this.first; x != null; x = next) {

next = x.next;

x.data = null;

x.next = null;

x.prev = null;

}

this.first = null;

this.last = null;

this.size = 0;

}

/\*\*

\* Переопределение функции toString().

\* @return строку

\*/

@Override

public String toString() {

StringBuilder builder = new StringBuilder();

if (this.getSize() == 0) {

builder.append("Список пуст!\n");

} else {

for (Node<T> x = this.first; x != null; x = x.next) {

builder.append(x.data.toString()).append("\n");

}

}

return builder.toString();

}

/\*\*

\* Преображает контейнер в массив.

\* @return массив элементов

\*/

public Object[] toArray() {

Object[] result = new Object[this.size];

int i = 0;

for (Node<T> x = this.first; x != null; x = x.next) {

result[i++] = x.data;

}

return result;

}

/\*\*

\* Метод, предназначеный

\* для создания итератора.

\* @return итератор

\*/

@Override

public Iterator<T> iterator() {

return new Iterator<>() {

private int currentPos = 0;

@Override

public boolean hasNext() {

return currentPos < size;

}

@Override

public T next() {

if (this.hasNext()) {

return getByIndex(currentPos++);

} else {

throw new NoSuchElementException();

}

}

};

}

public void sort(Comparator<T> comp){

if (this.getSize() == 0) {

System.out.println("Nothing to sort!");

} else {

if (this.getFirst() == null) {

System.out.println("Error!");

} else {

Node<T> current = this.getFirst();

Node<T> index = null;

T temp;

while(current != null) {

index = current.next;

while(index != null) {

if(comp.compare(current.data, index.data) > 0) {

temp = current.data;

current.data = index.data;

index.data = temp;

}

index = index.next;

}

current = current.next;

}

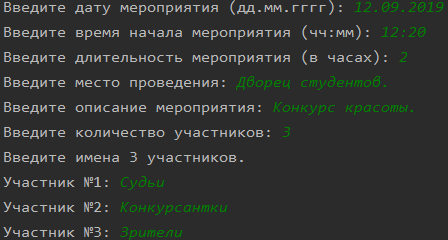
}

}

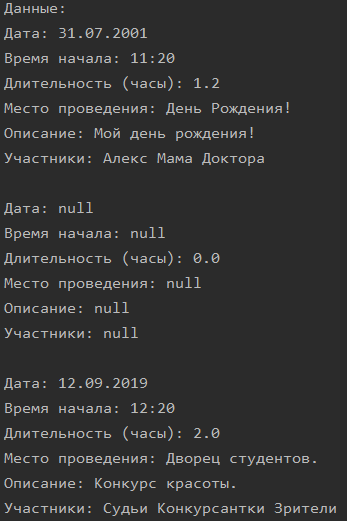
}

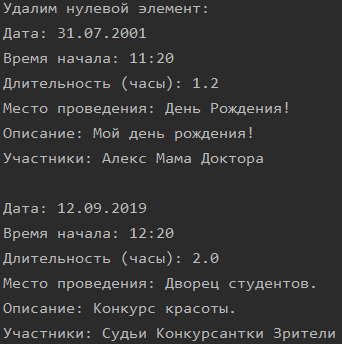
}

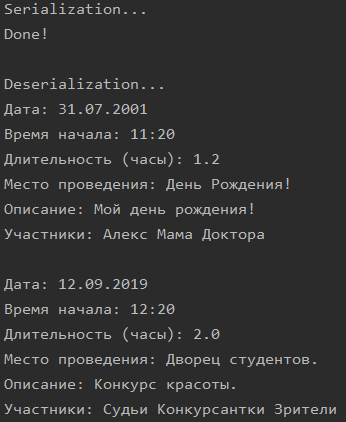
ВАРІАНТИ ВИКОРИСТАННЯ

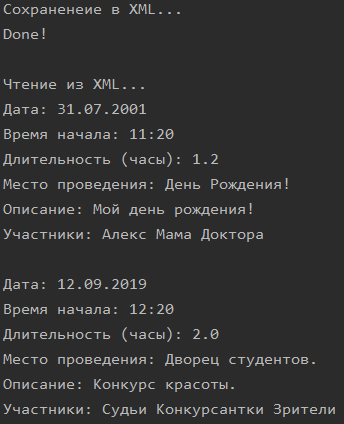
  
Рисунок 1 – Введення даних для заходу

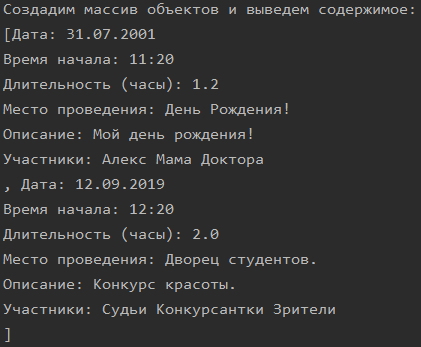
Додатково створюється ще два захода, їх дані виводяться на екран:

  
Рисунок 2 – Дані заходів

  
Рисунок 3 – Видалення елементу

  
Рисунок 4 – Використання серіалізації

  
Рисунок 5 – Використання XML

  
Рисунок 6 – Використання методу toArray()

ВИСНОВКИ

При виконанні лабораторної роботи набуто практичних навичок розробки параметризованих контейнерів. Створено контейнер, що реалізує таку динамічну структуру даних як двусвязний список, та його основні методи.