Третьяков І.С.

КІТ-118а

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

***2.1 Опис змінних:***

LinkedContainer<Event> linkedContainer = **new** LinkedContainer<Event>()

// об’єкт параметризованого контейнера

об’єкт класа планувальник

Event event = **new** Event(date,startTime,duration,venue,description,par1);

Scanner scan = new Scanner(System.in); // змінна для активування зчитування з консолі

***2.2 Ієрархія та структура класів.***

Main – головний клас. Містить метод main(точку входу у програму) та методи по роботі з програмою для реалізації індивідуального завдання.

interface Linked - інтерфейс контенеру

class Event - клас прикладної задачі планувальник

class LinkedContainer - параметризований клас-контейнер, котрий зберігає інформацію планувальника

**ТЕКСТ ПРОГРАМИ**

File Main.java:

package ua.khpi.oop.tretiakov09;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.TransformerException;

import java.util.Iterator;

import java.io.\*;

import java.util.LinkedList;

import java.util.Scanner;

public class Main {

public static void switcher(LinkedContainer linkedContainer, Event event, XmlWrite xmlWrite, XmlRead xmlRead) throws IOException,

ClassNotFoundException, TransformerException, ParserConfigurationException {

int choose;

do{

System.out.println("Choose action ");

Scanner in = new Scanner(System.in);

System.out.println("0. Exit");

System.out.println("1. Add elem");

System.out.println("2. Del elem");

System.out.println("3. Clear container ");

System.out.println("4. Convert to Array ");

System.out.println("5. Convert to String");

System.out.println("6. Create new element ");

System.out.println("7. Serialize ");

System.out.println("8. Deserialize ");

System.out.println("9. Xml serialize");

System.out.println("10. Xml deserialize");

choose = in.nextInt();

switch (choose) {

case 1:

System.out.println(linkedContainer.size());

linkedContainer.addLast(event);

System.out.println(linkedContainer.size());

break;

case 2:

System.out.println(linkedContainer.size());

System.out.println("Еnter index: ");

int index = in.nextInt();

linkedContainer.removeByIndex(index);

System.out.println(linkedContainer.size());

break;

case 3:

linkedContainer.clean();

System.out.println(linkedContainer.size());

break;

case 4:

// linkedContainer.toArray();

// System.out.println(linkedContainer.getElementByIndex(0));

Object []arr = linkedContainer.toArray().toArray();

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

{

System.out.println(arr[i]);

}

break;

case 5:

linkedContainer.toString();

break;

case 6:

Participant[] par2 = new Participant[2];

par2[0] = new Participant("Игнат", "Третьяков", 18);

par2[1] = new Participant("Ростислав", "Малахов", 19);

//par2[2] = new Participant("Данил", "Макаров", 17);

Scanner din = new Scanner(System.in);

Scanner cin = new Scanner(System.in);

System.out.println("Enter date");

String date = din.nextLine();

System.out.println("Enter start time(format HH:MM)");

String startTime=din.nextLine();

System.out.println("Enter duration");

int duration=cin.nextInt();

System.out.println("Enter venue");

String venue=din.nextLine();

System.out.println("Enter description");

String description=din.nextLine();

event = new Event(date,startTime,duration,venue,description,par2);

break;

case 7:

ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream("event.txt"));

objectOutputStream.writeObject(linkedContainer);

objectOutputStream.close();

break;

case 8:

ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream("event.txt"));

LinkedContainer<Event> newTravels = (LinkedContainer<Event>) objectInputStream.readObject();

objectInputStream.close();

for (Event e : newTravels) {

System.out.println(e);

}

break;

case 9:

xmlWrite.write(linkedContainer,"XML.xml");

break;

case 10:

LinkedContainer<Event> newXml = XmlRead.read("XML.xml");

for(Event e : newXml )

{

System.out.println(e);

}

//Object []array = newXml.toArray().toArray();

//for(int i=0; i<newXml.size();i++)

//{

// System.out.println(array[i]);

//}

break;

default:

System.out.println("Exit");

break;

}}while(choose!=0);

}

public static void main(String[] args) throws IOException, ClassNotFoundException, TransformerException, ParserConfigurationException {

//создание своего линкед листа

LinkedContainer<Event> linkedContainer = new LinkedContainer<Event>();

XmlWrite xmlWrite = new XmlWrite();

XmlRead xmlRead = new XmlRead();

//создание объекта

Participant par1[] = new Participant[2];

par1[0] = new Participant("Роман","Жолин",25);

par1[1] = new Participant("Евгений","Розумовский",24);

Scanner din = new Scanner(System.in);

Scanner cin = new Scanner(System.in);

System.out.println("Enter date");

String date = din.nextLine();

System.out.println("Enter start time(format HH:MM)");

String startTime=din.nextLine();

System.out.println("Enter duration");

int duration=cin.nextInt();

System.out.println("Enter venue");

String venue=din.nextLine();

System.out.println("Enter description");

String description=din.nextLine();

Event event = new Event(date,startTime,duration,venue,description,par1);

switcher(linkedContainer,event,xmlWrite, xmlRead);

}

}

LinkedContainer.java:

package ua.khpi.oop.tretiakov09;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.LinkedList;

public class LinkedContainer<E> implements Linked<E>, Iterable<E>, Serializable {

private int size = 0;

private Node<E> first;

private Node<E> last;

public LinkedContainer() {

last = new Node<E>(first,null,null);

first = new Node<E>(null,null,last);

}

@Override

public void addLast(E e) {

Node<E> prev = last;

prev.setItem(e);

last = new Node<E>(prev,null, null);

prev.setNext(last);

size++;

}

@Override

public void addFirst(E e) {

Node<E> next = first;

next.setItem(e);

first = new Node<E>(null, null, next);

next.setPrev(first);

size++;

}

@Override

public int size() {

return this.size;

}

@Override

public E getElementByIndex(int index) {

Node<E> target = first.getNext();

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

if (target == null) return null;

target = target.getNext();

}

return target.getItem();

}

@Override

public void removeByIndex(int index) {

Node<E> target = first.getNext();

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

if (target == null) return;

target = target.getNext();

}

Node<E> PrevRemoved = target.prev;

Node<E> NextRemoved = target.next;

PrevRemoved.next = NextRemoved;

NextRemoved.prev = PrevRemoved;

target.setItem(null);

target.setPrev(null);

target.setNext(null);

size--;

}

@Override

public void clean() {

Node<E> target = first.getNext();

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

target.setItem(null);

target = target.getNext();

}

last = new Node<E>(first,null, null);

first = new Node<E>(null, null, last);

size = 0;

}

ArrayList<E> toArray(){

ArrayList<E> result = new ArrayList<E>();

Node<E> target = first.getNext();

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

result.add(target.getItem());

target = target.getNext();

}

return result;

}

@Override

public String toString() {

StringBuilder builder = new StringBuilder();

Node<E> target = first.getNext();

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

builder.append(target.getItem().toString());

target = target.getNext();

}

return builder.toString();

}

boolean isEmpry(){

if(first.next == last){

return true;

}else {

return false;

}

}

@Override

public Iterator<E> iterator() {

Iterator<E> iterator = new Iterator<E>() {

int counter = 0;

@Override

public boolean hasNext() {

return counter < size;

}

@Override

public E next() {

return getElementByIndex(counter++);

}

};

return iterator;

}

private static class Node<E> implements Serializable {

E item;

Node<E> next;

Node<E> prev;

Node(Node<E> prev, E element, Node<E> next) {

this.item = element;

this.next = next;

this.prev = prev;

}

public E getItem() {

return item;

}

public void setItem(E item) {

this.item = item;

}

public Node<E> getNext() {

return next;

}

public void setNext(Node<E> next) {

this.next = next;

}

public Node<E> getPrev() {

return prev;

}

public void setPrev(Node<E> prev) {

this.prev = prev;

}

}

}

Event.java :

**package** ua.khpi.oop.tretiakov09;

**import** java.io.Serializable;

**public** **class** Event **implements** Serializable {

**private** String data; // Дата

**private** String startTime; // Время начала

**private** **int** duration; // Продолжительность

**private** String venue; // Место проведения

**private** String description; // Описание

**private** Participant[] participants; // Участники

**public** Event(String data, String startTime, **int** duration,

String venue, String description, Participant[] participants) {

**this**.data = data;

**this**.startTime = startTime;

**this**.duration = duration;

**this**.venue = venue;

**this**.description = description;

**this**.participants = participants;

}

**public** Event() {

}

**public** String getData() {

**return** **this**.data;

}

**public** **void** setData(String data) {

**this**.data = data;

}

**public** String getStartTime() {

**return** **this**.startTime;

}

**public** **void** setStartTime(String startTime) {

**this**.startTime = startTime;

}

**public** **int** getDuration() {

**return** **this**.duration;

}

**public** **void** setDuration(**int** duration) {

**this**.duration = duration;

}

**public** String getVenue() {

**return** **this**.venue;

}

**public** **void** setVenue(String venue) {

**this**.venue = venue;

}

**public** String getDescription() {

**return** **this**.description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** Participant[] getParticipants() {

**return** **this**.participants;

}

**public** **void** setParticipants(Participant[] participants) {

**this**.participants = participants;

}

@Override

**public** String toString() {

StringBuffer buf = **new** StringBuffer();

buf.append("Дата: " + **this**.data + "\n");

buf.append("Время начала: " + **this**.startTime + "\n");

buf.append("Продолжительность: " + **this**.duration + "\n");

buf.append("Место проведения: " + **this**.venue + "\n");

buf.append("Описание: " + **this**.description + "\n");

buf.append("Участники: ");

**for**(**int** i = 0; i < **this**.participants.length; i++) {

buf.append(**this**.participants[i].toString());

}

**return** buf.toString();

}

}

Linked.java:

**package** ua.khpi.oop.tretiakov09;

**public** **interface** Linked<E> {

**void** addLast(E e);

**void** addFirst(E e);

**int** size();

E getElementByIndex(**int** index);

**void** removeByIndex(**int** index);

**void** clean();

}

Participant.java:

**package** ua.khpi.oop.tretiakov09;

**import** java.io.Serializable;

**public** **class** Participant **implements** Serializable {

**private** String name;

**private** String surname;

**private** **int** age;

**public** Participant(String name, String surname, **int** age){

**this**.name = name;

**this**.surname = surname;

**this**.age = age;

}

**public** Participant(){

}

**public** String getName() {

**return** **this**.name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getSurname() {

**return** **this**.surname;

}

**public** **void** setSurname(String surname) {

**this**.surname = surname;

}

**public** **int** getAge() {

**return** **this**.age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

StringBuffer buf = **new** StringBuffer();

buf.append("Имя: " + **this**.name + "\n");

buf.append("Фамилия: " + **this**.surname + "\n");

buf.append("Возраст: " + **this**.age + "\n");

**return** buf.toString();

}

}

**ВАРІАНТИ ВИКОРИСТАННЯ**

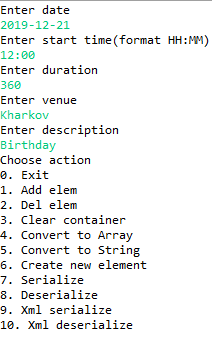


Рис. 9.1 – Результат роботи програми

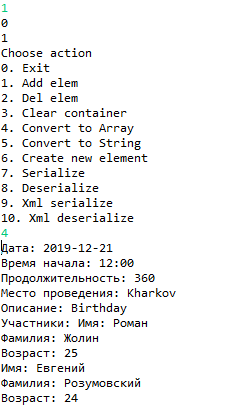


Рис. 9.2 – Результат роботи програми

Програму можна використовувати задля створення бази даних. Завдяки параметризації зв’язного списка, базу даних можна використати для будь-яких типів даних. Переважно у нашому варіанті - планувальник, в якому представляються різноманітні заходи. Також для вибору доступно багато інших можливостей.

**ВИСНОВКИ**

При виконанні лабораторної роботи набуто практичних навичок щодо розробки параметризованих класів. Завдяки цієї можливості в JAVA, можливо створювати колекції та інші класи на основі будь-яких типів.