НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА ПРОГРАМУВАННЯ»

«Об’єктно-орієнтоване програмування»

Звіт з лабораторної роботи №9
Тема: «Параметризація в Java»

Виконав:
ст. гр. 1.КІТ102.8а
Міщенко Д.С.

Перевірив:
Пугачов Р.В.

Харків – 2019

Лабораторна робота №9
Параметризація в Java

Мета

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

1 ВИМОГИ

1.1 Розробник

1. Міщенко Дмитро Сергійович
2. КІТ-118в
3. Варіант 13

1.2 Загальне завдання

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.**

Прикладні задачі

13. Готель. Замовлення номера: паспортні дані; дати поселення та виселення; номер (клас, число місць); причина поселення (кількість не обмежена).

2 ОПИС ПРОГРАМИ

2.1 Засоби ООП

Поліморфізм,інкапсуляція

2.2 Ієрархія та структура даних

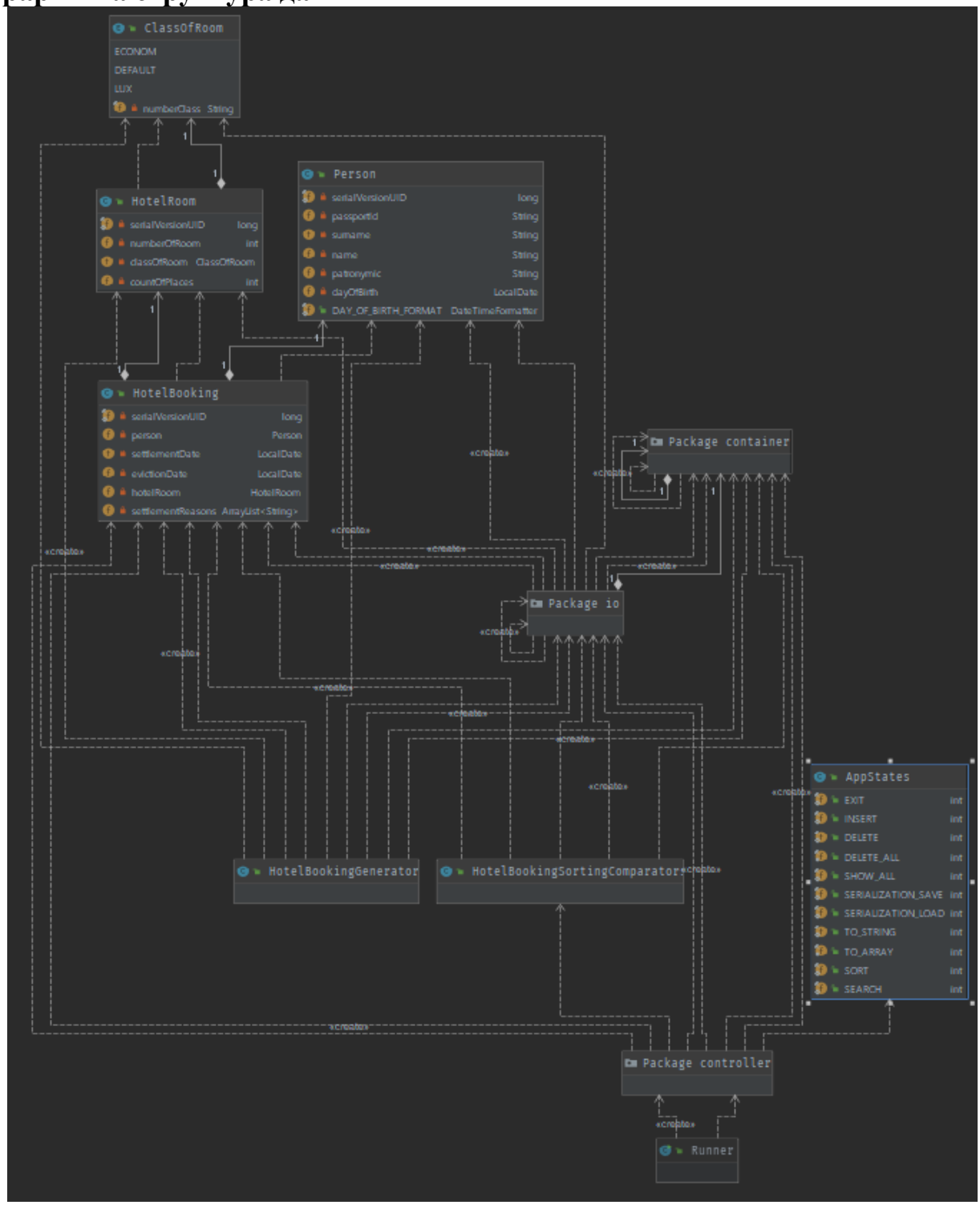


Рисунок 1 – Ієрархія классів

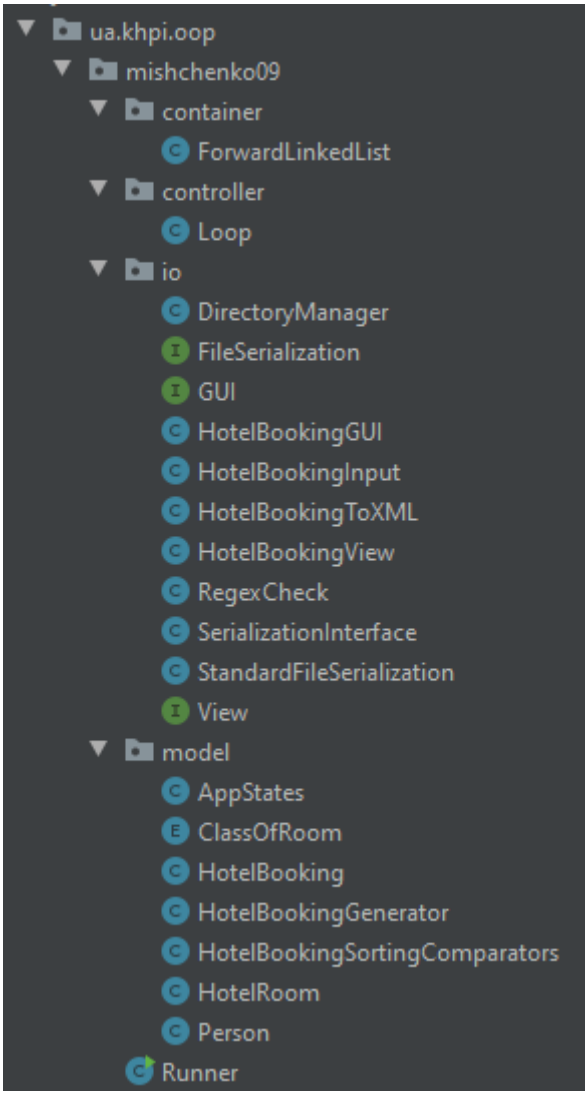


Рисунок 2 – Структура пакету

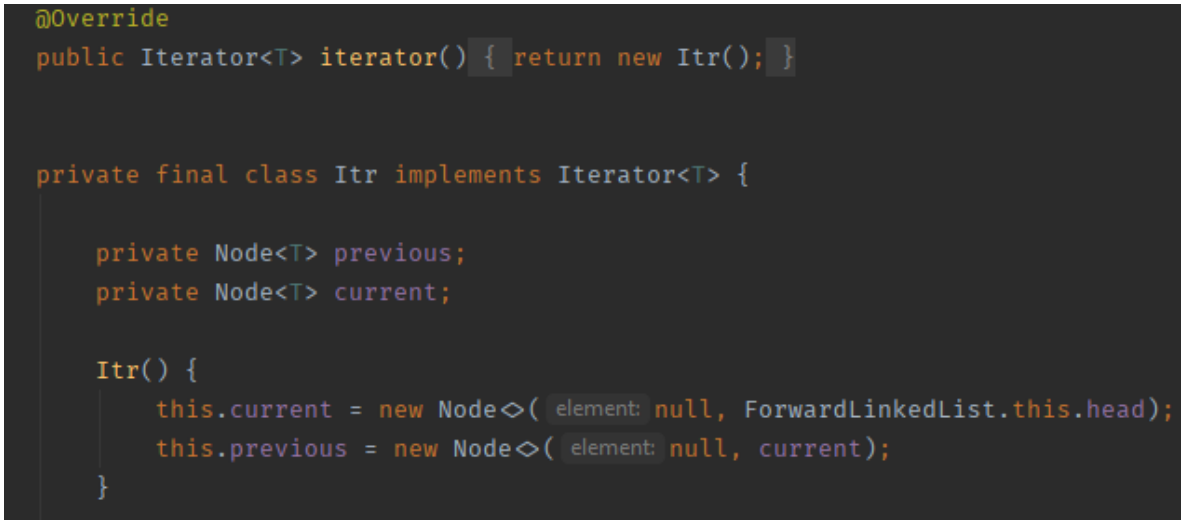
2.3 Важливі фрагменти програми

Створено контейнер на основі зв’язних списків(**LinkedList**) (рис. 1)

```
public class ForwardLinkedList<T> implements Serializable, Iterable<T> {

    private static final long serialVersionUID = 1L;
    private Node<T> head;
    private Node<T> tail;
    private int size;
```

Рисунок 3 – Клас-контейнер ForwardLinkedList



```
@Override
public Iterator<T> iterator() { return new Itr(); }


private final class Itr implements Iterator<T> {

    private Node<T> previous;
    private Node<T> current;

    Itr() {
        this.current = new Node<>( element: null, ForwardLinkedList.this.head);
        this.previous = new Node<>( element: null, current);
    }
```

Рисунок 4 – Реалізований ітератор для можливості використання їх об'єктів у циклі **foreach**

```
public void write(ForwardLinkedList<HotelBooking> forwardLinkedList, String path) {

    try {
        XMLEncoder encoder = new XMLEncoder(
                new BufferedOutputStream(
                        new FileOutputStream(path)));

        encoder.setPersistenceDelegate(LocalDate.class,
                (localDate, encdr) -> {
                        return new Expression(localDate,
                                LocalDate.class,
                                methodName: "parse",
                                new Object[]{localDate.toString()});
                });

        for (HotelBooking elem : forwardLinkedList) {
            encoder.writeObject(elem);
            elem.saveListXML(encoder);
        }
        encoder.close();
    } catch (FileNotFoundException e) {
        logger.error("File not found");
    }
}
```

Рисунок 5 – Метод збереження контейнера в XML файл

```java
@SuppressWarnings("unchecked")
public ForwardLinkedList<HotelBooking> read(String path) {
    try {
        FileInputStream fileInputStream = new FileInputStream(path);
        ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
        ForwardLinkedList<HotelBooking> list = (ForwardLinkedList<HotelBooking>) objectInputStream.readObject();
        objectInputStream.close();
        logger.info("Read successful");
        return list;
    } catch (IOException | ClassNotFoundException e) {
        logger.error("Error while reading {}", e.getMessage());
    }
    return null;
}
```

Рисунок 6 – Завантаження збереженого об’єкта за допомогою стандартного протоколу серіалізації(Serializable)

```java
public T[] toArray(T[] a) {
    if (a.length < size()) {
        T[] array = (T[]) Array.newInstance(a.getClass().getComponentType(), size());
        Node<T> temp = head;
        int currentIndex = 0;
        while (currentIndex < size) {
            a[currentIndex] = (T) temp.item;
            currentIndex++;
            temp = temp.next;
        }
    }

    if (a.length > size()) {
        a[size] = null;
    }

    Node<T> temp = head;
    for (int i = 0; i < size; i++) {
        a[i] = (T) temp.item;
        temp = temp.next;
    }

    return a;

}
```

Рисунок 7 – Метод **toArray**

3 ВАРІАНТИ ВИКОРИСТАННЯ

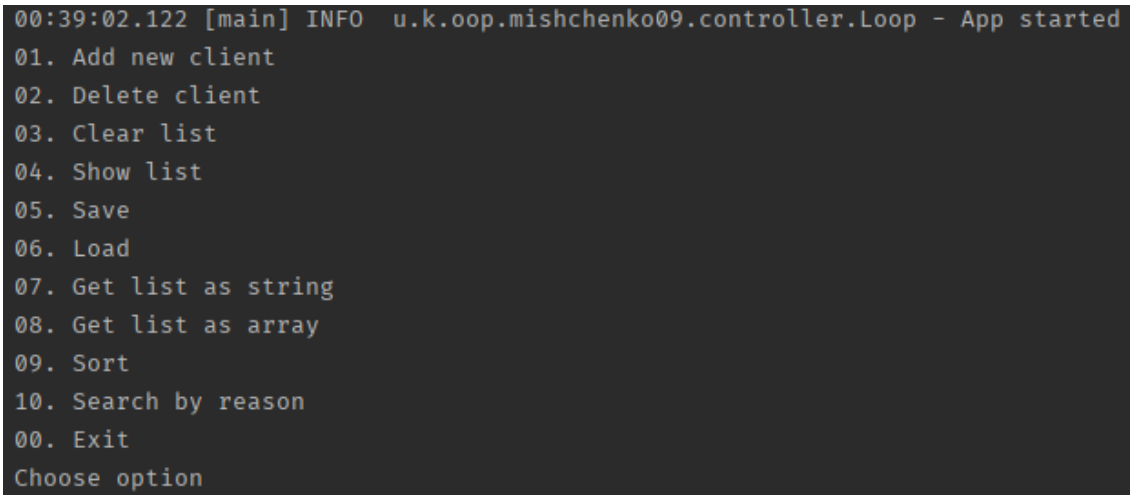


Рисунок 8 – Головне меню програми

Рисунок 9 – Виведення даних

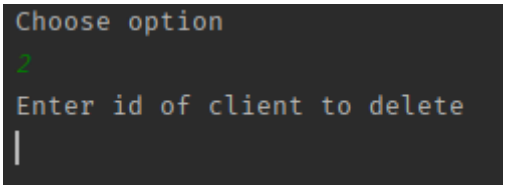


Рисунок 10 – Видалення елемента за індексом

ВИСНОВКИ

В результаті виконання лабораторної роботи були вивчені принципи параметризації в Java. Був розроблений власний параметризований клас-контейнер на основі зв’язних списків.