Лабораторна робота №7

Колекції: ArrayList. Порівняння об’єктів. Застосування JSON для серіалізації

Тема: Розробка та реалізація програм із використанням колекцій виду ArrayList для зберігання даних та файлового введення-виведення.

Мета роботи: отримати навички створення та реалізації програм, що використовують ArrayList для зберігання та опрацювання даних та реалізують операції введення-виведення із файлами.

ArrayList

Список представляє впорядковану послідовність значень, де якесь значення може зустрічатися більше одного разу.

ArrayList - одна з реалізацій списку, побудована поверх масиву, яка здатна динамічно зростати та зменшуватися під час додавання / видалення елементів. Елементи могли бути легко доступні за їх індексами, починаючи з нуля. Ця реалізація має такі властивості:

1. Довільний доступ займає час O(1)
2. Додавання елемента займає постійний час O(1)
3. Вставка / видалення займає час O(n)
4. Пошук займає O(n) часу для невідсортованого масиву та O(log n) для відсортованого

Створення ArrayList

ArrayList має кілька конструкторів, і ми їх розглянемо далі.

По-перше, зауважимо, що ArrayList є загальним класом, тому ви можете його параметризувати будь-яким типом, що вам заманеться, і компілятор забезпечить, наприклад, що ви не зможете додавати значення Integer всередину колекції рядків. Крім того, вам не потрібно перетворювати елементи, отримуючи їх із колекції.

По-друге, корисно використовувати загальний інтерфейс List як тип змінної, оскільки він відокремлює його від конкретної реалізації.

Default No-Arg Constructor

```
List<String> list = new ArrayList<>();
assertTrue(list.isEmpty());
```

Просте створення порожнього екземпляру ArrayList

Constructor, що приймає Initial Capacity

```
List<String> list = new ArrayList<>(100);
```

Тут ви вказуєте початкову довжину базового масиву. Це може допомогти вам уникнути зайвої зміни розміру під час додавання нових елементів.

Constructor, що приймає Collection

```
Collection<Integer> numbers = Set.of(1,2,3,4,5);

List<Integer> list = new ArrayList<>(numbers);
assertEquals(5, list.size());
assertTrue(numbers.containsAll(list));
```

Зверніть увагу, що кожен елемент екземпляра Collection використовується для заповнення базового масиву.

Додавання елементів у ArrayList

Ви можете вставити елемент або в кінці, або в певному положенні:

```
List<Long> list = new ArrayList<>();

list.add(1L);
list.add(2L);
list.add(1, 3L);

assertThat(List.of(1L, 3L, 2L), equalTo(list));
```

Ви також можете вставити колекцію або кілька елементів одночасно:

```
List<Long> list = new ArrayList<>();
list.add(1L);
list.add(2L);

list.addAll(List.of(7L, 8L, 111L));

assertThat(ArraysAsList(1L, 2L, 7L, 8L, 111L), equalTo(list));
```

Проход (итерация) по ArrayList

Доступні два типи ітераторів: Iterator та ListIterator.

У той час як перший дає вам можливість пройти по списку в одному напрямку, другий дозволяє вам пройти його в обидві сторони.

Тут ми розглянемо лише ListIterator:

```
List<Integer> list = new ArrayList<>(List.of(0,1,2,3,4,5,6,7,8,9));

ListIterator<Integer> it = list.listIterator(list.size());
List<Integer> result = new ArrayList<>(list.size());
while (it.hasPrevious()) {
    result.add(it.previous());
}

Collections.reverse(list);
assertThat(result, equalTo(list));
```

Ви також можете шукати, додавати або видаляти елементи за допомогою ітераторів.

Пошук в ArrayList

Ми продемонструємо, як працює пошук за допомогою колекції:

```
List<String> list = new ArrayList<>();
for (int i=0; i<16; i++) {
    list.add(Integer.toHexString(i));
}
List<String> stringsToSearch = new ArrayList<>(list);
stringsToSearch.addAll(list); // додаємо рядки ще раз
```

Пошук в неупорядкованому списку

Для того, щоб знайти елемент, ви можете використовувати методи indexOf() або lastIndexOf(). Вони обидва приймають об'єкт і повертають значення int:

```
assertEquals(10, stringsToSearch.indexOf("a"));
assertEquals(26, stringsToSearch.lastIndexOf("a"));
```

Також можна використовувати цикл for або ітератор:

```
Iterator<String> it = stringsToSearch.iterator();
Set<String> matchingStrings = new HashSet<>(Arrays.asList("a", "c", "9"));

List<String> result = new ArrayList<>();
while (it.hasNext()) {
    String s = it.next();
    if (matchingStrings.contains(s)) {
        result.add(s);
    }
}
```

Пошук у впорядкованому списку

Якщо у вас відсортований масив, ви можете використовувати двійковий алгоритм пошуку, який працює швидше, ніж лінійний пошук:

```
List<String> copy = new ArrayList<>(stringsToSearch);
Collections.sort(copy);
int index = Collections.binarySearch(copy, "f");
assert(index>0);
```

Як результат ми отримаємо індекс елемента, який ми шукали, якщо він міститься у списку; інакше, (- (точка вставки) - 1). Точка вставки визначається як точка, в яку елемент буде вставлений у список: індекс першого елемента, більший за шуканий, або list.size (), якщо всі елементи у списку менше зазначеного елемента. Зверніть увагу, що це гарантує, що повернене значення буде >= 0 тоді і тільки тоді, коли значення знайдено.

Видалення елементів з ArrayList

Для того, щоб видалити елемент, слід знайти його індекс і лише потім виконати видалення методом remove(). Перевантажена версія цього методу, яка приймає об'єкт, шукає його та виконує видалення першого входження рівного елемента:

```
List<Integer> list = new ArrayList<>(List.of(0,1,2,3,4,5,6,7,8,9));
Collections.reverse(list);

list.remove(0);
assertThat(list.get(0), equalTo(8));

list.remove(Integer.valueOf(0));
assertFalse(list.contains(0));
```

Але будьте обережні, працюючи з такими Wrapper класами, як Integer. Для того, щоб вилучити певний елемент, спочатку слід явно упаковати int у Integer або, інакше, елемент буде видалений за його індексом.

Ви також можете використовувати Stream API (розглядатиметься у наступних роботах) для видалення кількох елементів, але ми не будемо розглядати його тут. Для цього ми використаємо ітератор:

```
Set<String> matchingStrings = HashSet<>(Arrays.asList("a", "b", "c", "d", "e", "f"));

Iterator<String> it = stringsToSearch.iterator();
while (it.hasNext()) {
    if (matchingStrings.contains(it.next())) {
        it.remove();
    }
}
```

Використання JSON для серіалізації

У лабораторній роботі основна увага приділяється на розумінні використання класу ObjectMapper з бібліотеки Jackson та способі серіалізації об’єктів Java у JSON та десеріалізації рядка JSON у об’єкти Java.

Щоб зрозуміти більше про бібліотеку Jackson загалом, Jackson Tutorial - це гарне місце для початку.

Dependencies

Для використання механізмів серіалізації/десеріалізації із використанням бібліотеки Jackson треба її підключити до проекту. Наприклад, за допомогою Maven, це можна зробити вказавши у `pom.xml` відповідну залежність:

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.2</version>
</dependency>
```

***Примітка 1**: Ця залежність також транзитивно додасть наступні бібліотеки:*

1. jackson-annotations
2. jackson-core

***Примітка 2**: версія 2.12.2 є актуальною на момент оприлюднення цього завдання (березень 2021). У майбутньому версія буде мінятись. Завжди використовуйте найновіші версії бібліотеки jackson-databind. Перевірити їхню актуальність можна у центральному сховищі Maven jackson-databind*

Читання та Запис із використанням ObjectMapper

Почнемо з основних операцій читання та запису.

Почнемо з простого API readValue ObjectMapper. Ми можемо використовувати його для синтаксичного аналізу або десериалізації вмісту JSON у об’єкт Java.

Крім того, для запису, ми можемо використовувати API writeValue для серіалізації будь-якого об’єкта Java як виводу JSON.

Ми використовуватимемо наступний клас Person з двома полями як об’єкт для серіалізації або десеріалізації у прикладах цієї роботи:

```
public class Person {
    private String name;
    private String lastName;
    // constructors, getters, setters...
}
```

Перетворення Java Object в JSON

Розглянемо перший приклад серіалізації об'єкта Java у JSON за допомогою методу writeValue класу ObjectMapper:

```
ObjectMapper objectMapper = new ObjectMapper();
Person person = new Person("Vova", "Gray");
objectMapper.writeValue(new File("target/person.json"), person);
```

В результаті роботи цього фрагмента коду, у файл `person.json` каталогу `target` буде записане наступне:

```
{"name":"Vova","lastName":"Green"}
```

Методи writeValueAsString та writeValueAsBytes класу ObjectMapper генерують JSON з об'єкта Java і повертають зформований JSON як рядок або як масив байтів:

```
String personAsString = objectMapper.writeValueAsString(person);
```

Перетворення JSON в Java Object

Розглянемо приклад перетворення рядку JSON в Java object із використанням класу ObjectMapper:

```
String json = "{ \"name\" : \"Petya\", \"lastName\" : \"Bulkin\" }";
Person person = objectMapper.readValue(json, Person.class);
```

Функція readValue() також приймає інші форми введення, такі як файл, що містить рядок JSON:

```
Person person = objectMapper.readValue(new File("src/test/resources/json_person.json"
), Person.class);
```

Перетворення JSON в Jackson JsonNode

Крім того, JSON може бути проаналізований та перетворений в об'єкт JsonNode і

використаний для отримання даних із конкретного вузла:

```
String json = "{ \"name\" : \"Vasya\", \"lastName\" : \"Pupkin\" }";
JsonNode jsonNode = objectMapper.readTree(json);
String lastName = jsonNode.get("lastName").asText();
// Output: lastName -> Pupkin
```

Створення Java List з рядку JSON Array String

Ми можемо проаналізувати JSON у формі масиву та перетворити його у список об'єктів Java за допомогою TypeReference:

```
String jsonPeopleArray =
  "[{ \"name\" : \"Vova\", \"lastName\" : \"Green\" }, { \"name\" : \"Petya\",
\"lastName\" : \"Bulkin\" }]";
List<Person> people = objectMapper.readValue(jsonPeopleArray, new TypeReference<List
<Person>>(){});
```

Розширені засоби

Однією з найбільших сильних сторін бібліотеки Jackson є надзвичайно великі можливості налаштування процесу серіалізації та десеріалізації.

Налаштування функції серіалізації або десеріалізації

Під час перетворення об'єктів JSON у класи Java, якщо рядок JSON має деякі нові поля, процес за замовчуванням призведе до виникнення Exception:

```
String jsonString = "{ \"name\" : \"Vova\", \"lastName\" : \"Green\", \"rating\" :
\"60\" }";
```

Рядок JSON у наведеному вище прикладі в процесі аналізу за замовчуванням об'єкта Java для класу Person призведе до виникнення UnrecognizedPropertyException.

За допомогою методу configure ми можемо розширити процес за замовчуванням, щоб ігнорувати нові поля:

```
objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
Person person = objectMapper.readValue(jsonString, Person.class);

JsonNode jsonNodeRoot = objectMapper.readTree(jsonString);
JsonNode jsonNodeRating = jsonNodeRoot.get("rating");
String rating = jsonNodeRating.asText();
```

Ще один варіант заснований на FAIL_ON_NULL_FOR_PRIMITIVES, який визначає, чи

дозволені нульові значення для примітивних значень:

```
objectMapper.configure(DeserializationFeature.FAIL_ON_NULL_FOR_PRIMITIVES, false);
```

Подібним чином FAIL_ON_NUMBERS_FOR_ENUM контролює, чи дозволено серіалізувати / десеріалізувати значення переліків (enum) як числа:

```
objectMapper.configure(DeserializationFeature.FAIL_ON_NUMBERS_FOR_ENUMS, false);
```

Опрацювання дат

Для того, щоб увімкнути можливість серіалізації об'єктів з пакету java.time (що з'явився у Java 8) треба зареєструвати модуль серіалізації дати/часу:

```
ObjectMapper mapper = new ObjectMapper();
mapper.registerModule(new JavaTimeModule());
```

Після цього можна використовувати об'єкт mapper для серіалізації об'єктів, що мають поля дати/часу "нових" типів.

Опрацювання колекцій

Ще однією невеликою, але корисною функцією, доступною через клас DeserializationFeature, є можливість генерувати тип колекції, який ми хочемо, з відповіді масиву JSON.

Наприклад, ми можемо генерувати результат як масив:

```
String jsonPeopleArray =
  "[{ \"name\" : \"Vova\", \"lastName\" : \"Green\" }, { \"name\" : \"Petya\",
\"lastName\" : \"Bulkin\" }]";
ObjectMapper mapper = new ObjectMapper();
mapper.configure(DeserializationFeature.USE_JAVA_ARRAY_FOR_JSON_ARRAY, true);
Person[] people = mapper.readValue(jsonPeopleArray, People[].class);
// print people
```

або як список:

```
String jsonPeopleArray =
  "[{ \"name\" : \"Vova\", \"lastName\" : \"Green\" }, { \"name\" : \"Petya\",
\"lastName\" : \"Bulkin\" }]";
ObjectMapper mapper = new ObjectMapper();
List<Person> people = mapper.readValue(jsonPeopleArray, new TypeReference<List<Person
>>(){});
// print people
```

Завдання до лабораторної роботи

1. Створити клас за завданням лабораторної роботи №2. Визначити метод, що створює порожній масив об’єктів, з максимальною кількістю 100 елементів.
2. Реалізувати зберігання даних у текстовий файл та зчитування з текстового файлу. Враховувати можливість виникнення виключень, та обробляти їх, виводячи відповідні повідомлення для користувача.
3. Реалізувати зберігання даних у JSON файл (за допомогою ObjectMapper) та зчитування з JSON файлу (за допомогою ObjectMapper).
4. Створити інтерактивне меню, за допомогою якого надати можливість користувачеві виконувати додавання нових та вилучення існуючих елементів з масивів, файлові операції введення-виведення та запити відповідно варіанту завдання.

Примітка: всі операції виведення на екран повинні бути відокремлені від операцій пошуку та фільтрації даних. Для забезпечення такої поведінки рекомендується створити окремі класи для операцій екранного введення-виведення, файлового введення-виведення та бізнес-логіки застосування.

Варіанти завдань

Варіант 1.

Student: id, Прізвище, Ім’я, По батькові, Дата народження, Адреса, Телефон, Факультет, Курс, Група.

Створити колекцію об’єктів. Вивести:

a. список студентів заданого факультету;
b. список студентів, які народились після заданого року;
c. список навчальної групи в порядку алфавіту;
d. список студентів упорядкований за алфавітом назви факультету, а для студентів одного факультету – за датою народження

Варіант 2.

Customer: id, Прізвище, Ім’я, По батькові, Дата народження, Адреса, Номер кредитної картки, Баланс рахунку (кількість грошей).

Створити колекцію об’єктів. Вивести:

a. список покупців, із вказаним іменем;
b. список покупців, у яких номер кредитної картки знаходиться в заданому інтервалі;
c. кількість та список покупців, які мають заборгованість (від’ємний баланс на карті) в порядку зростання заборгованості;

d. список покупців, упорядкований за зростанням балансу рахунку, а при рівності балансів – за номером кредитної картки

Варіант 3.

Patient: id, Прізвище, Ім’я, По батькові, Адреса, Телефон, Номер медичної карти, Діагноз.

Створити колекцію об’єктів. Вивести:

a. список пацієнтів, які мають указаний діагноз в порядку зростання номерів медичної картки;
b. список пацієнтів, номер медичної карти у яких знаходиться в заданому інтервалі;
c. кількість та список пацієнтів, номер телефона яких починається з вказаної цифри;
d. список діагнозів пацієнтів (без повторів) із вказанням кількості пацієнтів, що мають цей діагноз у порядку спадання цієї кількості

Варіант 4.

Abiturient: id, Прізвище, Ім’я, По батькові, Адреса, Телефон, Середній бал.

Створити колекцію об’єктів. Вивести:

a. список абітурієнтів із вказаним іменем, в порядку спадання середнього балу;
b. список абітурієнтів, середній бал у яких вище заданого;
c. вибрати задане число n абітурієнтів, що мають найвищий середній бал.
d. список абітурієнтів в порядку алфавіту за прізвищем, при збігу прізвищ – за іменами

Варіант 5.

Book: id, Назва, Автор, Видавництво, Рік видання, Кількість сторінок, Ціна.

Створити колекцію об’єктів. Вивести:

a. список книг заданого автора в порядку зростання року видання;
b. список книг, що видані заданим видавництвом;
c. список книг, що випущені після заданого року;
d. список авторів в алфавітному порядку

Варіант 6.

House: id, Номер квартири, Площа, Поверх, Кількість кімнат, Вулиця.

Створити колекцію об’єктів. Вивести:

a. список квартир, які мають задане число кімнат;

b. список квартир, які мають задане число кімнат та розташовані на поверсі, який знаходиться в заданому проміжку;

c. список квартир, які мають площу, що перевищує задану в порядку спадання площі. Якщо площа однакова – то в порядку зростання поверху;

d. список всіх квартир, в порядку зростання площі

Варіант 7.

Phone: id, Прізвище, Ім’я, По батькові, Номер рахунку, Час міських розмов, Час міжміських розмов.

Створити колекцію об’єктів. Вивести:

a. відомості про абонентів, у яких час міських розмов перевищує заданий;

b. відомості про абонентів, які користувались міжміським зв’язком в порядку алфавіту за прізвищем, при однакових прізвищах – за іменами, потім по-батькові;

c. відомості про абонентів чий номер рахунку знаходиться у вказаному діапазоні;

d. відомості про всіх абонентів в порядку зростання сумарного часу розмов

Варіант 8.

Car: id, Модель, Рік випуску, Ціна, Реєстраційний номер.

Створити колекцію об’єктів. Вивести:

a. список автомобілів заданої моделі в порядку зростання року випуску;

b. список автомобілів заданої моделі, які експлуатуються більше n років;

c. список автомобілів заданого року випуску, ціна яких більше вказаної;

d. список автомобілів в порядку спадання ціни. Якщо ціна однакова, то в порядку зростання року випуску

Варіант 9.

Product: id, Найменування, Виробник, Ціна, Термін зберігання, Кількість.

Створити колекцію об’єктів. Вивести:

a. список товарів для заданого найменування в порядку спадання терміну зберігання;

b. список товарів для заданого найменування, ціна яких не перевищує задану;

c. список товарів, термін зберігання яких більше заданого;

d. список товарів, впорядкований за зростанням вартості (кількість * ціна), якщо вартість однакова, то за спаданням ціни

Варіант 10.

Train: id, Пункт призначення, Номер поїзду, Час відправки, Число місць (загальних, купе, плацкарт, люкс).

Створити колекцію об’єктів. Вивести:

a. список поїздів, які прямують до заданого пункту призначення в порядку зростання часу відправки, якщо час однаковий – за зростанням номеру поїзда;
b. список поїздів, які прямують до заданого пункту призначення та відправляються після заданої години;
c. список поїздів, які відправляються до заданого пункту призначення та мають загальні місця;
d. список поїздів, які відправляються до заданого пункту призначення в порядку зростання кількості всіх місць