Plan

2023-10-23

1. Predicate
2. Lambda
3. Arrays methods

Theory

▶ **English**

▼ **На русском**

Анонимные классы в Java

Что это такое?

Анонимные классы — это классы без имени, которые объявляются и создаются в одном выражении. Они используются, если нужно создать экземпляр класса для одноразового использования.

Пример:

```
public class Example {
    public static void main(String[] args) {

        Comparator<Interger> comparator = new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return o1 - o2;
            }
        };

        Arrays.sort(array, comparator);
    }

}
```

Лямбда-выражения в Java

Что это такое?

Лямбда-выражения — это сокращённый способ представления экземпляров анонимных классов, реализующих функциональный интерфейс. Функциональный интерфейс — это интерфейс с одним абстрактным методом.

Синтаксис

Базовый синтаксис лямбда-выражения выглядит следующим образом:

```
(parameters) -> expression
```

или (с телом)

```
(parameters) -> { statements; }
```

- **parameters**: параметры метода (можно опустить типы, скобки нужны, если параметров больше одного).
- **expression**: выражение, возвращающее результат.
- **statements**: блок кода, если нужно выполнить несколько операций.

Примеры

1. Без параметров:

```
() -> System.out.println("Hello, world!")
```

2. С одним параметром:

```
x -> x * x
```

3. С несколькими параметрами и телом:

```
(x, y) -> {
    int sum = x + y;
    return sum;
}
```

Применение

Лямбда-выражения часто используются в комбинации с методами из стандартной библиотеки, такими как `sort` и др.

```
        // Сортировка массива в обратном порядке
        Arrays.sort(array,(o1, o2) -> o2 - o1);
```

Замечания

- Лямбда-выражения не имеют доступа к переменным метода, которые изменяются (`non-final` или `effectively final`).
- В лямбда-выражениях нельзя определять новые переменные с именами, уже используемыми в области видимости.

Для чего это нужно?

Лямбда-выражения делают код более кратким и читаемым, особенно при работе с функциональными интерфейсами, коллекциями и потоками данных.

Предикаты в Java

Что это такое?

Предикаты — это функциональные интерфейсы, которые используются для проверки условий. Интерфейс `Predicate<T>` имеет метод `test`, который принимает объект типа `T` и возвращает `boolean`.

Пример с лямбдой:

```
Predicate<Integer> isEven = n -> n % 2 == 0; // лямбда
System.out.println(isEven.test(4)); // true
System.out.println(isEven.test(3)); // false
```

Пример с анонимным классом:

```
public class Main {
    public static void main(String[] args) {
        Predicate<Integer> isEven = new Predicate<Integer>() { // анонимный
            @Override
            public boolean test(Integer n) {
                return n % 2 == 0;
            }
        };

        System.out.println(isEven.test(4)); // true
        System.out.println(isEven.test(3)); // false
    }
}
```

Пример с реализацией интерфейса:

```
import java.util.function.Predicate;


public class IsevenPredicate implements Predicate<Integer> { // реализацией
    @Override
    public boolean test(Integer n) {
        return n % 2 == 0;
    }
```

```
}


public class Main {

    public static void main(String[] args) {
        Predicate<Integer> isEven = new IsevenPredicate();
        System.out.println(isEven.test(4)); // true
        System.out.println(isEven.test(3)); // false
    }
}
```

класс Arrays

В Java класс `Arrays` из пакета `java.util` предоставляет набор статических методов для работы с массивами. Вот некоторые из них:

1. **Сортировка**: `Arrays.sort(array)` сортирует массив в порядке возрастания.
    - Arrays.sort(int[] a): Сортирует целочисленный массив в порядке возрастания.
    - Arrays.sort(int[] a, int fromIndex, int toIndex): Сортирует часть массива от fromIndex до toIndex-1.
    - Arrays.sort(Object[] a): Сортирует объекты, реализующие интерфейс **Comparable**
        - public static void sort(Object[] a, int fromIndex, int toIndex): Эта перегрузка сортирует часть массива объектов, реализующих интерфейс Comparable, от индекса fromIndex до toIndex-1. Объекты сравниваются на основе их естественного порядка.
        - public static void sort(T[] a, int fromIndex, int toIndex, Comparator c): Эта версия позволяет сортировать часть массива с использованием специального компаратора. Comparator c определяет, как будут сравниваться объекты.
        - public static void sort(T[] a, Comparator c): Эта версия сортирует весь массив объектов с использованием заданного компаратора. Это удобно, когда естественный порядок сортировки объектов вам не подходит.
2. **Поиск**: `Arrays.binarySearch(array, value)` выполняет бинарный поиск значения в отсортированном массиве.
3. **Копирование**: `Arrays.copyOf(array, newLength)` создаёт копию массива с новой длиной.
4. **Заполнение**: `Arrays.fill(array, value)` заполняет все элементы массива заданным значением.
5. **Сравнение**: `Arrays.equals(array1, array2)` проверяет, равны ли два массива.
6. **Преобразование в строку**: `Arrays.toString(array)` возвращает строковое представление массива.

Методы `Arrays.copyOf` и `System.arraycopy` оба предназначены для копирования массивов, но есть несколько ключевых различий:

Arrays.copyOf:

1. **Создание нового массива**: `Arrays.copyOf` возвращает новый массив, который может иметь другую длину.
2. **Тип возвращаемого массива**: Может быть изменён, если используется перегрузка с параметром типа.
3. **Простота использования**: Очень прост в использовании, так как вам нужно указать только исходный массив и новую длину.

```
int[]original={1,2,3};
int[]copied=Arrays.copyOf(original,5); // [1, 2, 3, 0, 0]
```

System.arraycopy:

1. **Использует существующий массив**: Этот метод не создаёт новый массив, а копирует данные в уже существующий массив.
2. **Больше параметров**: Требует указания исходного и целевого массивов, позиций в этих массивах и количества копируемых элементов.
3. **Быстродействие**: Обычно быстрее, так как работает напрямую с памятью.

```
int[]original={1,2,3};
int[]destination=new int[5];
System.arraycopy(original,0,destination,0,original.length); // dest
```

В общем, `Arrays.copyOf` удобнее и проще в использовании для создания новых массивов, тогда как `System.arraycopy` обычно используется для копирования данных в уже существующие массивы и может быть быстрее в некоторых сценариях.

Перегрузки методов Arrays.copyOf и System.arraycopy, детально

Методы `Arrays.copyOf` и `System.arraycopy` оба предназначены для копирования массивов, но есть несколько ключевых различий:

Arrays.copyOf:

1. **Создание нового массива**: `Arrays.copyOf` возвращает новый массив, который может иметь другую длину.
2. **Тип возвращаемого массива**: Может быть изменён, если используется перегрузка с параметром типа.

3. **Простота использования**: Очень прост в использовании, так как вам нужно указать только исходный массив и новую длину.

```
int[]original={1,2,3};
int[]copied=Arrays.copyOf(original,5); // [1, 2, 3, 0, 0]
```

System.arraycopy:

1. **Использует существующий массив**: Этот метод не создаёт новый массив, а копирует данные в уже существующий массив.
2. **Больше параметров**: Требует указания исходного и целевого массивов, позиций в этих массивах и количества копируемых элементов.
3. **Быстродействие**: Обычно быстрее, так как работает напрямую с памятью.

```
int[]original={1,2,3};
int[]destination=new int[5];
System.arraycopy(original,0,destination,0,original.length); // dest
```

В общем, `Arrays.copyOf` удобнее и проще в использовании для создания новых массивов, тогда как `System.arraycopy` обычно используется для копирования данных в уже существующие массивы и может быть быстрее в некоторых сценариях.

Практика:

1 спринт

Написать метод сортировки рабочих используя Comparator и Comparable.

- написать Comparator для полей
  - id
    - Для поля id используйте интерфейс Comparable
  - hireYear
    - Используйте анонимный класс для поля hireYear
  - name
    - Используйте лямбду для поле name
- написать метод фильтрации массива рабочих по условиям:
  - найти всех рабочих чья зарплата в диапазоне от и до.
  - найти всех рабочих кто был устроен в определенный период времени (fromHireYear, toHireYear).
  - и так далее для всех полей.

2 спринт

Применить полученные знания по Predicate

- внесите изменения в код, где это необходимо
- удалите не нужный код

Homework

<details>
<summary>English</summary>
</details>

<details open>
<summary>На русском</summary>

1. Создайте анонимный класс, реализующий `Comparator<String>`. Сравнивайте строки по количеству гласных букв.
    - **Подсказка**: Используйте метод `charAt()` для прохода по каждому символу строки и подсчета гласных.
2. Используя лямбда-выражение, реализуйте `Comparator<String>`, который сравнивает строки по количеству согласных букв.
    - **Подсказка**: Также можно использовать метод `charAt()` для подсчета согласных. ~~3. **Предикаты**: Создайте предикат, который проверяет, является ли целое число степенью двойки.
    - **Подсказка**: Число является степенью двойки, если `n & (n - 1) == 0` и `n > 0`.~~
3. У вас есть класс `Student` с полями `name` (имя), `age` (возраст), и `gpa` (средний балл). Ваша задача — написать программу, которая сможет фильтровать и сортировать список студентов по различным критериям.

Подзадачи:

~~1. **Предикат**: Создайте предикат, который проверяет, является ли студент совершеннолетним (возраст 18 и выше).~~

1. Создайте метод, который проверяет, является ли студент совершеннолетним (возраст 18 и выше).
2. **Лямбда-выражение и Comparator**: Используя лямбда-выражение, создайте компаратор, который сортирует студентов по среднему баллу (GPA) в убывающем порядке.
3. **Итоговая задача**: Создайте метод, который принимает массив студентов и использует предикат и компаратор для:
    - Фильтрации списка, оставляя только совершеннолетних студентов.
    - Сортировки отфильтрованного списка по среднему баллу в убывающем порядке.

</details>

Code