Stream API и лямбда выражения

Рассматриваемые вопросы

1. Лямбда выражения
2. Stream API

Лямбда выражения

Лямбда-выражение, по существу, является анонимным (т.е. безымянным) методом.

Лямбда выражение приводит к некоторой форме анонимного класса.

Лямбда выражения

```
() -> 123.45
```

аналогично

```
double myMeth() { return 123.45 }
```

Лямбда выражения

Функциональный интерфейс (functional interface) – это интерфейс у которого только один абстрактный метод.

Функциональный интерфейс может содержать любое количество методов по умолчанию (default) или статических методов.

Лямбда выражения

```
interface MyNumber {

        double getValue();

}
```

Лямбда выражения

```
MyNumber myNum;

myNum = () -> 123.45;

System.out.println(myNum.getValue());
```

Лямбда выражения

```
interface NumericTest {
    boolean test(int n);
}

class LambdaDemo2 {
    public static void main(String args[]) {
        NumericTest isEven = (n) -> (n % 2)==0;

        if(isEven.test(10)) System.out.println("Число 10 четное");
        if(!isEven.test(9)) System.out.println("Число 9 нечетное");

         //является ли число неотрицательным?
        NumericTest isNonNeg = (n) -> n >= 0;

        if(isNonNeg.test(1)) System.out.println("Число 1
неотрицательное");
        if(!isNonNeg.test(-1)) System.out.println("Число -1
отрицательное");
    }
}
```

Лямбда выражения

```
interface NumericTest2 {
    boolean test(int n, int d);
}

class LambdaDemo3 {
    public static void main(String args[]) {

        //Является ли одно число множителем другого
        NumericTest2 isFactor = (n, d) -> (n % d) == 0;


        if(isFactor.test(10, 2))
            System.out.println("Число 2 является множителем числа 10");

        if(!isFactor.test(10, 3))
            System.out.println("Число 3 не является множителем числа 10");
    }
}
```

Лямбда выражения

```
interface NumericFunc {
    int func(int n);
}

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

        // вычисляет факториал целочисленного значения
        NumericFunc factorial = (n) -> {
            int result = 1;

            for(int i=1; i <= n; i++)
                result = i * result;

            return result;
        };

        System.out.println("Факториал числа 3 равен " + factorial.func(3));
        System.out.println("Факториал числа 5 равен " + factorial.func(5));
    }
}
```

Обобщенные функциональные интерфейсы

```
interface SomeFunc<T> {
    T func(T t);
}
```

Обобщенные функциональные интерфейсы

```
SomeFunc<String> reverse = (str) -> {
            String result = "";
            int i;

            for(i = str.length()-1; i >= 0; i--)
                result += str.charAt(i);
            return result;
        };

System.out.println( reverse.func("Лямбда") );
```

Обобщенные функциональные интерфейсы

```
SomeFunc<Integer> factorial = (n) -> {
            int result = 1;

            for(int i=1; i <= n; n++)
                result = i * result;

            return result;
        };

System.out.println( factorial.func(3) );
```

Ссылки на методы

Для создания ссылки на статический метод служит следующая общая форма:

```
имя_класса::имя_метода
```

Ссылки на методы

```
interface StringFunc {
    String func(String n);
}

class MyStringOps {
    static String strReverse(String str) {
        int i;
        String result = "";

        for(i = str.length()-1; i >= 0; i--)
            result += str.charAt(i);

        return result;
    }
}
```

Ссылки на методы

```
class MethodRefDemo {

    static String stringOp(StringFunc sf, String s) {
        return sf.func(s);
    }

    public static void main(String args[]) {
        String inStr = "some text";

        //ссылка на метод strReverse() передается методу stringOp()
        String outStr; = stringOp(MyStringOps::strReverse, inStr);
    }
}
```

Ссылки на методы

Для передачи ссылки на метод экземпляра для конкретного объекта служит следующая форма:

```
ссылка_на_объект::имя_метода
```

Предопределенные функциональные интерфейсы

- UnaryOperator<T> - принимает в качестве параметра объект типа T, выполняет над ними операции и возвращает результат операций в виде объекта типа T
- BinaryOperator<T> принимает в качестве параметра два объекта типа T, выполняет над ними бинарную операцию и возвращает ее результат также в виде объекта типа T
- Predicate<T> проверяет соблюдение некоторого условия. Если оно соблюдается, то возвращается значение true. В качестве параметра лямбда-выражение принимает объект типа T
- Function<T,R> представляет функцию перехода от объекта типа T к объекту типа R
- Consumer<T> выполняет некоторое действие над объектом типа T, при этом ничего не возвращая
- Supplier<T> не принимает никаких аргументов, но должен возвращать объект типа T

Stream API

Обобщенный базовый интерфейс BaseStream:

```
interface BaseStream<T, S extends BaseStream<T, S>>
```

Производный интерфейс от BaseStream:

```
interface Stream<T>
```

Stream API

Интерфейсы для обработки потоков примитивных типов данных:

- DoubleStream
- IntStream
- LongStream

Stream API

Создание или получение объект java.util.stream.Stream:

- Пустой стрим: Stream.empty() // Stream<String>
- Стрим из List: list.stream() // Stream<String>
- Стрим из Map:

  map.entrySet().stream() // Stream<Map.Entry<String, String>>
- Стрим из массива: Arrays.stream(array) // Stream<String>
- Стрим из указанных элементов: Stream.of("a", "b", "c")

Операции Stream API

- filter(Predicate predicate)
- map(Function mapper)
- sorted() и sorted(Comparator comparator)
- forEach
- toArray

...

Вопросы для самоконтроля

1. Опишите основные методы применяемые в Stream API.
2. Какие основные функциональные интерфейсы появились в Java 8?