Функциональное программирование

**Функциональное программирование** – это стиль программирование в котором функции являются объектами, и их можно присваивать переменным, передавать в качестве аргументов другим функциям, возвращать в качестве результата от функций и т. п.

функциональные интерфейсы

**функциональный интерфейс** – это интерфейс который содержит только один абстрактный метод. Они помечены аннотацией @FunctionalInterface.

Лямбда выражения

**Лямбда-выражение** — это анонимный класс или метод. . Лямбда-выражение не выполняется само по себе. Вместо этого, оно используется для реализации метода, определенного в функциональном интерфейсе.

Одним из ключевых моментов в использовании лямбд является отложенное выполнение (deferred execution). То есть мы определяем в одном месте программы лямбда-выражение и затем можем его вызывать при необходимости неопределенное количество раз в различных частях программы.

**Синтаксис лямбда-выражений:**

(parameter list) -> lambda body

Основу лямбда-выражения составляет лямбда-оператор, который представляет стрелку ->. Этот оператор разделяет лямбда-выражение на две части: левая часть содержит список параметров выражения, а правая собственно представляет тело лямбда-выражения, где выполняются все действия.

Предположим, у нас есть такой метод:

double getPiValue() {

return 3.1415;

}

Мы можем записать его, используя лямбда, как:

() -> 3.1415

**Типы лямбда-выражений:**

1. **Однострочные**

() -> System.out.println("Lambdas are great");

**2. Блочные (многострочные)**

() -> {

double pi = 3.1415;

return pi;

};

В блочных лямбда-выражениях можно использовать внутренние вложенные блоки, циклы, конструкции if, switch, создавать переменные и т.д. Если блочное лямбда-выражение должно возвращать значение, то явным образом применяется оператор return:

**Пример лямбда-выражение:**

import java.lang.FunctionalInterface;

// функциональный интерфейс

@FunctionalInterface

interface MyInterface{

double getPiValue();

}

public class Main {

public static void main( String[] args ) {

// объявление ссылки на MyInterface

MyInterface ref;

// лямбда-выражение

ref = () -> 3.1415;

System.out.println("Value of Pi = " + ref.getPiValue());

}

}

**Лямбда-выражения с параметрами**

(n) -> (n % 2) == 0

Переменная n внутри скобок является параметром, переданном в лямбда-выражение. Тело лямбды принимает параметр и проверяет его.

Параметры лямбда-выражения должны соответствовать по типу параметрам метода из функционального интерфейса. При написании самого лямбда-выражения тип параметров писать необязательно

**С ними используется Параметризированный функциональный интерфейс**

@FunctionalInterface

interface MyInterface {

String reverseString(String n);

}

Java.util.function

**Java.util.function –** этобиблиотека с готовыми функциональными интерфейсами.

**BiConsumer<T,U> -** представляет операцию, которая принимает два входных аргумента и не возвращает результата.

**BiFunction<T,U,R> -** представляет функцию, которая принимает два аргумента и выдает результат.

**BinaryOperator<T> -** представляет функцию, которая принимает два аргумента и выдает результат.

**BiPredicate<T,U> -** представляет собой предикат (функцию с логическим значением) из двух аргументов.

**BooleanSupplier -** представляет поставщика boolean-значимые результаты.

**Consumer<T> -** представляет операцию, которая принимает единственный входной аргумент и не возвращает результата.

**DoubleBinaryOperator -** представляет собой операцию над двумя double-многозначные операнды и выдающие double- значимый результат.

**DoubleConsumer -** представляет операцию, которая принимает один double- многозначный аргумент и не возвращает результата.

**DoubleFunction<R> -** представляет функцию, которая принимает двузначный аргумент и выдает результат.

**DoublePredicate -** представляет собой предикат (булевозначную функцию) одного double- значимый аргумент.

**DoubleSupplier** - представляет поставщика double-значимые результаты.

**DoubleToIntFunction -** представляет функцию, которая принимает двузначный аргумент и выдает многозначный результат.

**DoubleToLongFunction -** представляет функцию, которая принимает двузначный аргумент и выдает многозначный результат.

**DoubleUnaryOperator -** представляет операцию над одним double- многозначный операнд, который выдает double- значимый результат.

**Function<T,R> -** представляет функцию, которая принимает один аргумент и выдает результат.

**IntBinaryOperator -** представляет операцию над двумя int-значными операндами и выдает int-значный результат.

**IntConsumer -** представляет операцию, которая принимает аргумент с одним int значением и не возвращает результата.

**IntFunction<R> -** представляет функцию, которая принимает многозначный аргумент и выдает результат.

**IntPredicate -** представляет собой предикат (булевозначную функцию) одного int-значного аргумента.

**IntSupplier -** представляет поставщика int-значимых результатов.

**IntToDoubleFunction -** представляет функцию, которая принимает многозначный аргумент и выдает двузначный результат.

**IntToLongFunction -** представляет функцию, которая принимает многозначный аргумент и выдает многозначный результат.

**IntUnaryOperator -** представляет операцию над однозначным int операндом, которая выдает int многозначный результат.

**LongBinaryOperator -** представляет операцию над двумя long-значными операндами и выдает long-значный результат.

**LongConsumer -** представляет операцию, которая принимает аргумент с одним long значением и не возвращает результата.

**LongFunction<R> -** представляет функцию, которая принимает длиннозначный аргумент и выдает результат.

**LongPredicate -** представляет собой предикат (булевозначную функцию) одного long-значного аргумента.

**LongSupplier -** представляет поставщика long-значимых результатов.

**LongToDoubleFunction -** представляет функцию, которая принимает длиннозначный аргумент и выдает двузначный результат.

**LongToIntFunction -** представляет функцию, которая принимает длиннозначный аргумент и выдает многозначный результат.

**LongUnaryOperator -** представляет операцию над однозначным long операндом, которая выдает long многозначный результат.

**ObjDoubleConsumer<T> -** представляет операцию, которая принимает объектно-значный и double-значный аргументы и не возвращает результата.

**ObjIntConsumer<T> -** представляет операцию, которая принимает объектно-значный и int-значный аргументы и не возвращает результата.

**ObjLongConsumer<T> -** представляет операцию, которая принимает объектно-значный и long-значный аргументы и не возвращает результата.

**Predicate<T> -** представляет собой предикат (функцию с логическим значением) одного аргумента.

**Supplier<T> -** представляет поставщика результатов.

**ToDoubleBiFunction<T,U> -** представляет функцию, которая принимает два аргумента и выдает двузначный результат.

**ToDoubleFunction<T> -** представляет функцию, которая выдает двузначный результат.

**ToIntBiFunction<T,U> -** представляет функцию, которая принимает два аргумента и выдает результат со значением int.

**ToIntFunction<T> -** представляет функцию, которая выдает многозначный результат.

**ToLongBiFunction<T,U> -** представляет функцию, которая принимает два аргумента и выдает многозначный результат.

**ToLongFunction<T> -** представляет функцию, которая выдает многозначный результат.

**UnaryOperator<T> -** представляет операцию над одним операндом, которая выдает результат того же типа, что и его операнд.

Stream API

**Stream API** – позволяет упростить работу с наборами данных. Например, операции фильтрации, сортировки и т.д. с коллекциями и массивами.

BaseStream

**BaseStream** –это базовый интерфейс для потоков который представляет собой последовательности элементов, поддерживающих последовательные и параллельные операции на ними.

**Методы:**

**close()** - закрывает этот поток, вызывая все обработчики close для этого конвейера stream.

**isParallel()** - возвращает, будет ли этот поток, если должна была выполняться операция терминала, выполняться параллельно.

**iterator()** - возвращает итератор для элементов этого потока.

**onClose(Runnable closeHandler)** - возвращает эквивалентный поток с дополнительным обработчиком закрытия.

**parallel()** - возвращает эквивалентный поток, который является параллельным.

**sequential()** - возвращает эквивалентный поток, который является последовательным.

**spliterator()** - возвращает разделитель для элементов этого потока.

**unordered()** -возвращает эквивалентный поток, который является неупорядоченным.

Stream

**Stream** – это интерфейс, расширяющий BaseStream который, представляет последовательность элементов, поддерживающих последовательные и параллельные операции над ними.

**Методы:**

**allMatch(Predicate<? super T> predicate)** - возвращает, соответствуют ли все элементы этого потока предоставленному предикату.

**anyMatch(Predicate<? super T> predicate)** - возвращает, соответствуют ли какие-либо элементы этого потока предоставленному предикату.

**builder()** - возвращает конструктор для Stream.

**collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)** - выполняет изменяемое сокращение работа с элементами этого потока.

**collect(Collector<? super T,A,R> collector)** - выполняет изменяемое сокращение операции над элементами этого потока с использованием Collector.

**concat(Stream<? extends T> a, Stream<? extends T> b)** - создает лениво объединенный поток, элементами которого являются все элементы первого потока, за которыми следуют все элементы второго потока.

**count()** - возвращает количество элементов в этом потоке.

**distinct()** - возвращает поток, состоящий из отдельных элементов (в соответствии с Object.equals(Object)) этого потока.

**dropWhile(Predicate<? super T> predicate)** - возвращает, если этот поток упорядочен, поток, состоящий из оставшихся элементов этого потока после удаления самого длинного префикса элементов, соответствующих данному предикату.

**empty()** - возвращает пустой последовательный Stream.

**filter(Predicate<? super T> predicate)** - возвращает поток, состоящий из элементов этого потока, которые соответствуют заданному предикату.

**findAny()** - возвращает Optional описывающий некоторый элемент потока или пустой Optional если поток пуст.

**findFirst()** - возвращает Optional описание первого элемента этого потока или пустой Optional если поток пуст.

**flatMap(Function<? super T,? extends Stream<? extends R>> mapper)** - возвращает поток, состоящий из результатов замены каждого элемента этого потока содержимым отображенного потока, созданного путем применения предоставленной функции отображения к каждому элементу.

**flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)** - возвращает DoubleStream состоящий из результатов замены каждого элемента этого потока содержимым отображенного потока, созданного путем применения предоставленной функции отображения к каждому элементу.

**flatMapToInt(Function<? super T,? extends IntStream> mapper)** - возвращает IntStream состоящий из результатов замены каждого элемента этого потока содержимым отображенного потока, созданного путем применения предоставленной функции отображения к каждому элементу.

**flatMapToLong(Function<? super T,? extends LongStream> mapper)** - возвращает LongStream, состоящий из результатов замены каждого элемента этого потока содержимым отображенного потока, созданного путем применения предоставленной функции отображения к каждому элементу.

**forEach(Consumer<? super T> action)** - выполняет действие для каждого элемента этого потока.

**forEachOrdered(Consumer<? super T> action)** - выполняет действие для каждого элемента этого потока в порядке встреч потока, если поток имеет определенный порядок встреч.

**generate(Supplier<? extends T> s)** - возвращает бесконечный последовательный неупорядоченный поток, где каждый элемент генерируется предоставленным Supplier.

**iterate(T seed, Predicate<? super T> hasNext, UnaryOperator<T> next) -** возвращает последовательный упорядоченный Stream создается путем итеративного применения данного next функция для начального элемента, обусловленная удовлетворением заданного hasNext предикат.

**iterate(T seed, UnaryOperator<T> f)** - возвращает бесконечный последовательный упорядоченный Stream создается путем итеративного применения функции f к исходному элементу seed, создавая Stream состоящий из seed, f(seed), f(f(seed)) и т.д.

**limit(long maxSize)** - возвращает поток, состоящий из элементов этого потока, усеченный так, чтобы он был не длиннее maxSize в длину.

**map(Function<? super T,? extends R> mapper)** - возвращает поток, состоящий из результатов применения данной функции к элементам этого потока.

**mapMulti(BiConsumer<? super T,? super Consumer<R>> mapper)** - возвращает поток, состоящий из результатов замены каждого элемента этого потока несколькими элементами, в частности нулем или более элементов.

**mapMultiToDouble(BiConsumer<? super T,? super DoubleConsumer> mapper)** - возвращает DoubleStream состоящий из результатов замены каждого элемента этого потока несколькими элементами, в частности нулем или более элементов.

**mapMultiToInt(BiConsumer<? super T,? super IntConsumer> mapper)** - возвращает IntStream состоящий из результатов замены каждого элемента этого потока несколькими элементами, в частности нулем или более элементов.

**mapMultiToLong(BiConsumer<? super T,? super LongConsumer> mapper)** - возвращает LongStream, состоящий из результатов замены каждого элемента этого потока несколькими элементами, в частности нулем или более элементов.

**mapToDouble(ToDoubleFunction<? super T> mapper)** - возвращает DoubleStream состоящий из результатов применения данной функции к элементам этого потока.

**mapToInt(ToIntFunction<? super T> mapper)** - возвращает IntStream состоящий из результатов применения данной функции к элементам этого потока.

**mapToLong(ToLongFunction<? super T> mapper)** - возвращает LongStream состоящий из результатов применения данной функции к элементам этого потока.

**max(Comparator<? super T> comparator)** - возвращает максимальный элемент этого потока в соответствии с предоставленным Comparator.

**min(Comparator<? super T> comparator)** - возвращает минимальный элемент этого потока в соответствии с предоставленным Comparator.

**noneMatch(Predicate<? super T> predicate)** - возвращает, соответствует ли ни один элемент этого потока предоставленному предикату.

**of(T t)** - возвращает последовательный Stream содержащий единственный элемент.

**of(T... values)** - возвращает последовательный упорядоченный поток, элементами которого являются указанные значения.

**ofNullable(T t)** - возвращает последовательный Stream содержащий один элемент, если он не равен null, в противном случае возвращает пустой Stream.

**peek(Consumer<? super T> action)** - возвращает поток, состоящий из элементов этого потока, дополнительно выполняя предусмотренное действие над каждым элементом, поскольку элементы потребляются из результирующего потока.

**reduce(BinaryOperator<T> accumulator)** - выполняет сокращение для элементов этого потока, используя ассоциативный функция накопления и возвращает Optional описание уменьшенного значения, если таковое имеется.

**reduce(T identity, BinaryOperator<T> accumulator)** - выполняет сокращение на элементах этого потока, используя предоставленное идентификационное значение и ассоциативный функция накопления и возвращает уменьшенное значение.

**reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)** - выполняет сокращение над элементами этого потока, используя предоставленные функции идентификации, накопления и объединения.

**skip(long n)** - возвращает поток, состоящий из оставшихся элементов этого потока, после отбрасывания первого n элементы потока.

**sorted()** - возвращает поток, состоящий из элементов этого потока, отсортированных в соответствии с естественным порядком.

**sorted(Comparator<? super T> comparator)** - возвращает поток, состоящий из элементов этого потока, отсортированных в соответствии с предоставленными Comparator.

**takeWhile(Predicate<? super T> predicate)** - возвращает, если этот поток упорядочен, поток, состоящий из самого длинного префикса элементов, взятых из этого потока, которые соответствуют данному предикату.

**toArray()** - возвращает массив, содержащий элементы этого потока.

**toArray(IntFunction<A[]> generator)** - возвращает массив, содержащий элементы этого потока, используя предоставленный generator функция для выделения возвращаемого массива, а также любых дополнительных массивов, которые могут потребоваться для секционированного выполнения или для изменения размера.

**toList()** - аккумулирует элементы этого потока в List.

**Вложенные классы:**

**Stream.Builder** – это изменяемый конструктор для Stream. Это позволяет создавать a Stream путем генерации элементов по отдельности и добавления их в Builder.

**Методы:**

**accept(T t)** - добавляет элемент в создаваемый поток.

**add(T t)** - добавляет элемент в создаваемый поток.

**build()** - создает поток, переводя этот конструктор в состояние built.

Интерфейсы Stream API

**BaseStream<T,S extends BaseStream<T,S**>> - базовый интерфейс для потоков, которые представляют собой последовательности элементов, поддерживающих последовательные и параллельные операции агрегирования.

**Collector<T,A,R>** - изменяемая операция сокращения, которая накапливает входные элементы в изменяемый контейнер результатов, необязательно преобразуя накопленный результат в окончательное представление после обработки всех входных элементов.

**Collector.Characteristics** - характеристики, указывающие на свойства Collector, которые могут быть использованы для оптимизации реализаций сокращения.

**Collectors** - реализации Collector, которые реализуют различные полезные операции сокращения, такие как накопление элементов в коллекции, суммирование элементов в соответствии с различными критериями и т.д.

**DoubleStream** - последовательность примитивных двузначных элементов, поддерживающих последовательные и параллельные операции агрегирования.

**DoubleStream.Builder** - изменяемый конструктор для DoubleStream.

**DoubleStream.DoubleMapMultiConsumer** - представляет операцию, которая принимает double-значный аргумент и значение DoubleConsumer и не возвращает результата.

**IntStream** - последовательность примитивных многозначных элементов, поддерживающих последовательные и параллельные операции агрегирования.

**IntStream.Builder** - изменяемый конструктор для IntStream.

**IntStream.IntMapMultiConsumer** - представляет операцию, которая принимает int-значный аргумент и значение IntConsumer и не возвращает результата.

**LongStream** - последовательность примитивных длиннозначных элементов, поддерживающих последовательные и параллельные операции агрегирования.

**LongStream.Builder** - Изменяемый конструктор для LongStream.

**LongStream.LongMapMultiConsumer** - представляет операцию, которая принимает long-значный аргумент и значение LongConsumer и не возвращает результата.

**Stream<T>** - последовательность элементов, поддерживающих последовательные и параллельные операции агрегирования.

**Stream.Builder<T>** - изменяемый конструктор для Stream.

**StreamSupport** - низкоуровневые служебные методы для создания потоков и управления ими.