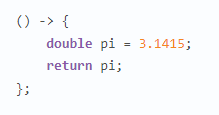
**Функциональный интерфейс и лямбда выражение и Stream API**

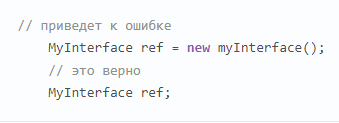
Изначально Java задумывалась как объектно ориентированный язык, позже функциональное программирование стало топовым и ввели возможность функционального программирования. Для проверки компилятором над функциональным интерфейсом может устанавливаться аннотация @FunctionalInterface, что не позволит создать больше 1 абстрактного метода.

Функциональный интерфейс содержит только 1 метод (абстрактный) и может содержать сколько угодно static и default методов, следовательно, нам не нужно указывать название метода при передаче его в качестве аргумента. Именно это и позволяет нам lambda-выражения. Лямбда-выражение это по сути анонимный класс или метод, которое не выполняется само по себе. Вместо этого оно используется для реализации метода, объявленного в функциональном интерфейсе.

Лямбда-выражение делится на левую и правую часть, в правой части после -> может находится однострочный блок кода или многострочный, который выглядит следующим образом и обязан что то возвращать (используем return):



Создать экземпляр интерфейса мы не сможем, но можем реализовать метод интерфейса применяя лямбда-выражение:

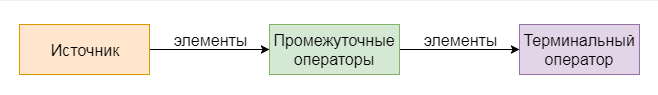

Лямбда-выражения могут принимать параметры, возвращать разные типы (прописывается это всё в методе интерфейса), а так же функциональные интерфейсы можно параметризировать.

Есть ещё особенность в лямбда-выражениях, что у неё отсутствует своя область видимости (нельзя создать переменную А во вне и в нашей лямбда).

**Stream API**

(Функциональное написание кода)

Stream – это объект для универсальной работы с данными. Позволяет удобно извлекать данные, сортировать, записывать в новый формат и др. Данные могут быть получены из разных источников: из метода, из коллекции.

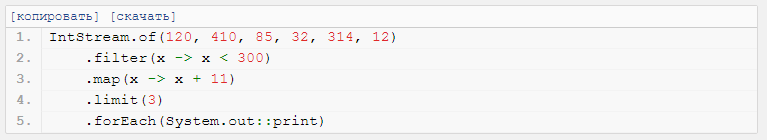


Операторы могут быть:

Промежуточные – обрабатывают поступающие элементы и возвращают stream (может быть много в цепочке).

Терминальные – обрабатывают элементы и завершают работу stream (в цепочке только один).

У Stream есть особенность – обработка не начнётся, пока не будет вызван терминальный оператор. Stream после обработки нельзя переиспользовать. Обработка происходит от терминального оператора к источнику данных.



Пример:

Пока мы не присоединили терминальный оператор, доступа к источнику не проводилось. Как только появился терминальный оператор forEach (к примеру), он стал запрашивать элементы у стоящего перед ним оператора limit. Тот в свою очередь обращается к map, map к filter, а filter уже обращается к источнику. Затем элементы поступают в прямом порядке: источник, filter, map, limit и forEach. Пока какой-либо из операторов не обработает элемент должным образом, новые запрошены не будут. Как только через оператор limit прошло 3 элемента, он переходит в закрытое состояние и больше не будет запрашивать элементы у map. forEach запрашивает очередной элемент, но limit сообщает, что больше не может поставить элементов, поэтому forEach делает вывод, что элементы закончились и прекращает работу.

Классная ссылка с подробным описанием:

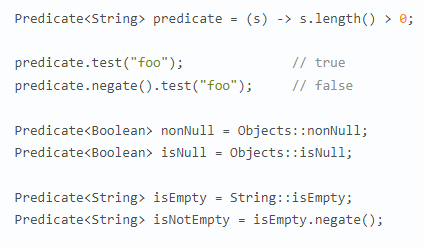
<https://annimon.com/article/2778>

**Базовые функциональные интерфейсы Java 8**

Примеры функциональных интерфейсов, которые активно применяются в связки с Stream API.

**Predicate**

Это функциональный интерфейс принимающий один аргумент и возвращающие значение типа boolean. Интерфейс так же содержит некоторые методы для построения логических связей (or, and, negate).



**Consumer**

Потребитель. Функциональный интерфейс который принимает в качестве входного аргумента объект типа Т, совершает некоторые действия но при этом ничего не возвращает.

**Supplier**

Поставщик. Функциональный интерфейс, который не принимает никаких аргументов, но возвращает обьект типа Т.