**Лямбда-выражение** или просто лямбда в Java — упрощённая запись анонимного класса, реализующего функциональный интерфейс. (С помощью нее нельзя реализовать абстрактный класс с одним абстрактным методом).

Лямбда-выражение не выполняется само по себе, а образует реализацию метода, определенного в функциональном интерфейсе.

Лямбда-выражение состоит из двух частей: списка аргументов и тела, разделенных при помощи ->.

(a, b) -> a + b;

Типы лямбда вырожений:

* Однострочные (Одиночные)
* Блочные

Обрамляется в {}. Внутри можно использовать внутренние вложенные блоки, циклы, конструкции if, switch, создавать переменные и т. д. Если лямбда-выражение должно возвращать значение, то явно прописывается return.

Внутри лямбда-вырожений можно ссылаться на

* неизменяемые (*effectively final* - не обязательно помеченные как final) локальные переменные;

Случай, когда в lambda используется локальная переменная называется Захват переменной. Лямбда создает копию локальной переменной, так как лямбда может быть вызвана после выхода из объемлющей области (То есть когда сама переменная пропадет из стека). Это также не мешает использовать несколько потоков.

* поля класса;
* статические переменные.

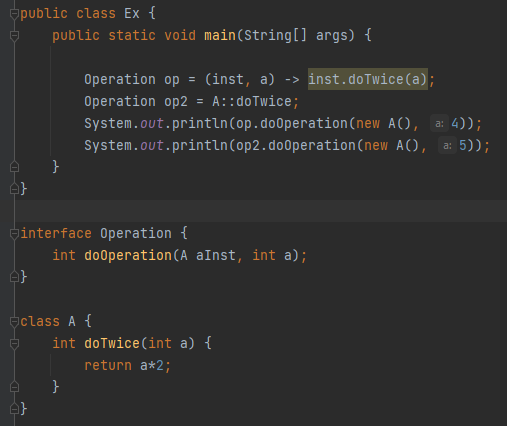
К методам по умолчанию реализуемого функционального интерфейса обращаться внутри лямбда-выражения запрещено. Это связано с тем, что this в лямбда указывает на объект обрамляющего класса. Лямбда-выражения при компиляции превращаются в приватный метод того класса, где они написаны.

Лямбда выражение может генерировать исключение. Если оно генерирует проверяемое исключение, то оно должно быть совместимым с исключениями из throws из объявления абстрактного метода в функциональном интерфейсе.

В качестве реализации функционального интерфейса можно передовать ссылку на уже существующий метод (механизм *instance method reference*). Такая ссылка передается в виде:

* имя\_класса::имя\_статического\_метода **для статического метода**;
* объект\_класса::имя\_метода **для метода экземпляра**;

Если в метод функционального интерфейса требуется передать объект какого-то класса, и внутри выполнить его метод с набором параметров(необязательным) то можно передавать ссылку на нестатический метод через сам тип: *ObjectType::instanceMethod*



* название\_класса::new **для конструктора**. Сигнатура метода в интерфейсе в качестве возвращаемого значения должна содержать имя данного класса. С помощью реализованного таким образом метода можно создавать объекты другого класса.

Можно ссылаться и на обобщенный метод. 

При этом необязательно указывать тип в угловых скобках, он может быть выведен автоматически. Можно также вызовать конструкторы обобщенных классов.



Для создания ссылки на конструктор массива используется следующая конструкиция

Код лямбда выражения не выполняется в момент создания. Он выполняется во время вызова.

**ФУНКЦИОНАЛЬНЫЙ ИНТЕРФЕЙС**

Если интерфейс в Java содержит один и только один абстрактный метод, то он называется **функциональным**. Этот единственный метод определяет назначение интерфейса. При этом нереализованные методы класса Object не учитываются, поскольку автоматически реализуются.

Например, интерфейс Runnable из пакета java.lang, Comparator.

Function — этот функциональный интерфейс принимает аргумент T и приводит его к объекту типа R, который и возвращается как результат.

UnaryOperator — функциональный интерфейс, принимает в качестве параметра объект типа T, выполняет над ним некоторые операции и возвращает результат операций в виде объекта того же типа T.

Методы StreamApi: filter(функция фильтрации), map()

Аннотация *FunctionalInterface*, указывает компилятору, что интерфейс функциональный (не позволяет иметь более одного абстрактного метода). Но это не обязательное условие, так как JVM считает функциональным любой интерфейс с одним абстрактным методом.

В Java 7, функциональные интерфейсы рассматривались как Single Abstract Methods (SAM). SAM обычно реализовывались с помощью анонимных классов.