Lambda expressions

Самый короткий вариант написания лямбда выражения:

```
stud ->  stud.avgGrade > 8.5
```

Более полный вариант написания лямбда выражения:

```
(Student stud) ->  {return stud.avgGrade > 8.5;}
```

В лямбда выражении оператор стрелка разделяет параметры метода и тело метода.

В лямбда выражении справа от оператора стрелка находится тело метода, которое было бы у метода соответствующего класса, имплементировавшего наш интерфейс с единственным методом.

Lambda expressions

Вы можете использовать смешанный вариант написания лямбда выражения: слева от оператора стрелка писать короткий вариант, справа – полный. Или наоборот.

Если вы используете полный вариант написания для части лямбда выражения справа от стрелки, то вы должны использовать слово return и знак «;»

Левая часть лямбда выражения может быть написана в краткой форме, если метод интерфейса принимает только 1 параметр. Даже если метод интерфейса принимает 1 параметр, но в лямбда выражении вы хотите писать данный параметр используя его тип данных, тогда уже вы должны писать левую часть лямбда выражения в скобках.

Если в правой части лямбда выражения вы пишите более одного statement-а, то вы должны использовать его полный вариант написания.

Lambda expressions

```
def(  () -> 5  );
```

```
def(  (x) -> x.length()  );
```

```
def(  (String x) -> x.length()  );
```

```
def(  (x, y) -> x.length()  );
```

```
def(  (String x, String y) -> x.length()  );
```

Compile time errors:

```
def(  x -> {x.length();}  );
```

```
def(  x -> {return x.length()}  );
```

```
def(  x, y -> x.length()  );
```

Lambda expressions

method(**(int x, int y) -> {int x=5; return10;}**);	**NOT OK**
method(**(int x, int y) -> {x=5; return10;}**);	**OK**
method(**(int x, int y) -> {int x2=5; return10;}**);	**OK**

Лямбда выражения работают с интерфейсом, в котором есть только 1 абстрактный метод. Такие интерфейсы называются функциональными интерфейсами, т.е. интерфейсами, пригодными для функционального программирования.

Пакет java.util.function

Predicate<T>	boolean test(T t);
Используется методом removeIf	
Supplier<T>	T get ();
Consumer<T>	void accept (T t);
Используется методом forEach	
Function<T, R>	R apply (T t);