ФУНКЦИОНАЛЬНЫЕ ИНТЕРФЕЙСЫ
ЛЯМБДА ВЫРАЖЕНИЯ

Функциональный интерфейс (Functional interface) - интерфейс, который содержит только один абстрактный метод и любое количество методов с реализацией (default или static)

```java
// аннотация FunctionalInterface используется для указания на то,
// что интерфейс является функциональным (указывать не обязательно)
@FunctionalInterface
public interface Operation {
    double calculate(double a, double b);
}
```

Лямбда выражение (Lambda Expression) представляет набор инструкций, которые можно выделить в отдельную переменную и затем многократно вызвать в различных местах программы.

Лямбда-выражение всегда образует реализацию метода, определенного в функциональном интерфейсе.

```java

@FunctionalInterface
public interface Operation {
    double calculate(double a, double b);
}

// Лямбда-выражение - реализация метода
// функционального интерфейса Operation
// сохранена в переменную plus - объект типа Operation
Operation plus = (a, b) -> a + b;
// вызов метода calculate у объекта (plus)
System.out.println(plus.calculate(45, 12));

```

Синтаксис лямбда выражений.

Список аргументов

1. можно не заключать в (), если аргумент одни
2. обязательно заключать в (), если аргументов нет или аргументов больше одного
3. можно не указывать тип аргументов, типы берутся из контекста - метода в интерфейсе

Синтаксис лямбда выражений.

Лямбда оператор - стрелка

Разделяет лямбда-выражение на две части: список параметров выражения и тело лямбда-выражения

Синтаксис лямбда выражений.

Тело лямбда-выражения

1. если реализация метода - одна инструкция и метод возвращает значение, то {} и return не нужны (return по умолчанию)
2. если реализация метода - несколько инструкций, то {} нужны (return при необходимости вернуть значение указывается явно)

Синтаксис лямбда выражений.
Примеры

```
 1
 2 Predicate<Integer> more18 = age -> age > 18;
 3
 4 Operation minus = (a, b) -> a - b;
 5
 6 Comparator<User> byLogin =
 7         (user1, user2) -> user2.getLogin().compareTo(user1.getLogin()
 8
 9 Consumer<User> update = user -> {
10     user.setRole(User.Role.USER);
11     user.setPwd("1234");
12 };
13
```

Встроенные функциональные интерфейсы (пакет java.util.function) определяют наиболее распространенные действия, избавляют нас от необходимости лишний раз создавать свой интерфейс.

java.util.function.Predicate<T> используется для проверки на соответствие условию.
Абстрактный метод boolean test(T t);
Ссылка на документацию

java.util.function.Function<T, R> предоставляет возможность перехода от объекта типа T к объекту типа R.
Абстрактный метод R apply(T t);
Ссылка на документацию

java.util.function.Consumer<T> используется для выполнения некоторого действия (например, обновление свойств) над объектом типа T.
Абстрактный метод void accept(T t);
Ссылка на документацию

java.util.function.Supplier<T> используется для создания объектов
Абстрактный метод T get();
Ссылка на документацию

java.util.function.UnaryOperator<T> используется для выполнения некоторого действия над объектом типа T.
Абстрактный метод T apply(T t);
Ссылка на документацию

java.util.function.BinaryOperator<T> используется для выполнения некоторого действия над двумя объектами типа T
Абстрактный метод T apply(T t1, T t2);
Ссылка на документацию

Ссылки на методы (Method References) - компактные лямбда выражения для методов. Ссылки на методы позволяют упростить код.

Ссылки на методы можно использовать, только если параметры вызываемого метода и параметры в лямбда-выражении совпадают.

```
1 @FunctionalInterface
2 public interface Operation {
3     double calculate(double a, double b);
4 }
5 /* ссылка на метод sum класса Double */
6 Operation plus = Double::sum;
7 plus.calculate(4, 89);
8 /* ссылка на метод getPrice класса Car */
9 Comparator<Car> byPrice = Comparator.comparingInt(Car::getPrice);
```

Comparator - функциональный интерфейс, что позволяет записать реализацию метода compare через лямбда - выражение

```
1
2 Comparator<Student> byAge =
3           (student1, student2)-> student1.getAge() - student2.getAge()
4
```

Метод класса Comparator для более удобного создания компараторов

public static Comparator comparing(Function key)

Метод принимает Function - указание на то, что нужно использовать при сравнении объектов.

Метод принимает на вход объект и возвращает свойство, которое будет использоваться для сравнения объектов друг с другом.

```
1
2 Comparator<Student> byAge =
3           Comparator.comparing(student -> student.getAge());
4 // можно заменить ссылкой на метод,
5 // аргумент student будет получен из контекста Student,
6 // аргумент key метода comparing будет типа Function<Student, Integer>
7 Comparator<Student> byAge = Comparator.comparing(Student::getAge);
```