**Вопросы по теме Лямбда-выражения и ссылка на метод:**

1. Что такое функциональный интерфейс?
2. Распространенные функциональные интерфейсы?
3. Что такое лямбда-выражения?
4. К каким переменным есть доступ у лямбда выражений?
5. Отсортируйте список с помощью лямбда-выражения.
6. Что такое «ссылка на метод» (method reference)?
7. Виды ссылок на метод?

**Функциональный интерфейс.**

Функциональный интерфейс – интерфейс, который содержит только один абстрактный метод. Может содержать сколько угодно статических методов и методов по умолчанию. Для определения функционального интерфейса используется аннотация @FunctionalInterface, которая на даст определить второй абстрактный метод в интерфейсе.

Базовые функциональные интерфейсы:

**– Function<T, R>** - получает на вход объект типа T и возвращает объект типа R.

*Function<String, Integer> toInt = x -> Integer.parseInt(x);*

*System.out.println(toInt.apply("123").getClass()); // java.lang.Integer*

– **BinaryOperator<T>** - получает на вход два объекта типа T и возвращает один типа Т.

*BinaryOperator<String> operator = (x, y) -> x + y;*

*System.out.println(operator.apply("Min", "sk")); // Minsk*

– **UnaryOperator<T>** - получает на вход объект типа T, выполняет операции над ним и возвращает объект типа Т.

*UnaryOperator<Integer> operator = x -> x + x;*

*System.out.println(operator.apply(15)); // 30*

– **Pedicate<T>** - получает на вход объект типа T и возвращает значение типа boolean.

*Predicate<Integer> predicate = x -> x > 0;*

*System.out.println(predicate.test(-10)); //false*

– **Consumer<T>** - на вход объект типа T, выполняет операции над ним, ничего не возвращает.

*Consumer<String> consumer = x -> System.out.println(x.toLowerCase());*

*consumer.accept("MiNsK"); //minsk*

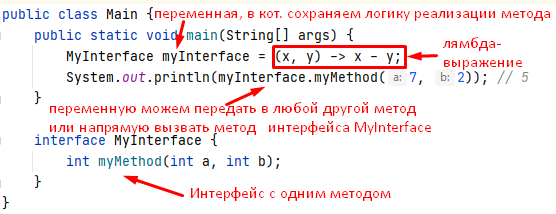
– **Supplier<T>** - ничего не принимает на вход, возвращает объект типа Т.

*Supplier<Double> supplier = () -> Math.PI;*

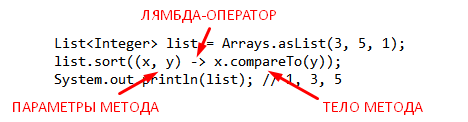
*System.out.println(supplier.get()); // 3.141592653589793*

**Лямбда-выражение.**

Лямбда-выражение – набор инструкций (блок кода), который можно выделить в отдельную переменную и потом многократно вызывать или передавать в какой-нибудь метод в качестве аргумента.



Лямбда-выражение используется для реализации метода в интерфейсах, содержащих всего один единственный метод. Интерфейс не обязательно должен быть помечен аннотацией @FunctionalInterface, но желательно (для контроля наличия всего одного абстрактного метода).



**Параметры метода** – переменные, которые передаются в единственный метод интерфейса. Как и любая другая переменная могут принимать любое произвольное название.

**Лямбда-оператор** – разделяет выражение на параметры и тело метода.

**Тело метода** – реализация логики единственного абстрактного метода интерфейса.

**Блочное лямбда-выражение** – заключено в фигурные скобки {} и может содержать в себе конструкции if, switch, циклы, создавать собственные переменные. Если такое выражение должно вернуть значение, то явно применяется оператор return.

*TestInterface test = (x) -> { return x > 0 ? "Положительное" : "Отрицательное"; };*

У лямбда-выражений есть доступ к:

1. неизменяемым локальным переменным;
2. полям класса;
3. статическим переменным.

К методам по умолчанию реализуемого функционального интерфейса обращаться внутри лямбда-выражения запрещено.

При компиляции лямбда-выражение превращается в новый private static метод.

**Ссылка на метод.**

Ссылка на метод – непосредственная передачи ссылка на существующий метод. Например,

*Function<String, Integer> toInt = x -> Integer.parseInt(x);* можно переписать на

*Function<String, Integer> toInt = Integer::parseInt;*

Ссылку на метод использовать предпочтительнее, т.к. лямбда-выражение в байткоде превращается в новый private static метод, а при использовании ссылки на метод, используемый уже существует метод. Получается меньше затрат памяти и быстрее загрузка класса в JVM.

**Виды ссылок на метод:**

1. на статический метод имя класса :: имя статического метода
2. на конструктор имя класса :: new
3. на метод экземпляра объект класса :: имя метода