Java 8! →Presentation By GT

**Topics:**

* Lambdas
* Functional programming
* Benefits of functional programming?
* Functional programming in Java 8
  + - DEMO 1
* SAM/Lambdas
  + - DEMO 2-3
    - SAM eg: Predicate DEMO 4-5
* Stream API
  + - DEMO 6
* Other features: String Join, new Date API, Optional<T> etc
  + - DEMO 7-9

**Lambdas:**

* Skip the math part!!!

**Examples:**

* A Java 8 lambda is basically a method in Java without a declaration
* written as (parameters) -> { body }
* Examples:
  + (int x, int y) -> { return x + y; }
  + (x, y) -> x + y
  + x -> x \* x
  + ( ) -> x
* A lambda can have zero or more parameters separated by commas
  + type can be explicitly declared or inferred from the context.
* Parenthesis are not needed around a single parameter.
* ( ) is used to denote zero parameters.
* The body can contain zero or more statements.
  + Braces are not needed around a single-statement body.

**Benefits of Lambdas**

* Allows us to write easier-to-understand, more declarative, more concise programs than imperative programming
* Allows us to focus on the problem rather than the code
* Facilitating parallel programming
* Developing more generic, flexible and reusable APIs
* Being able to pass behaviors as well as data to functions
* Enabling functional programming

**Functional Programming:**

* A style of programming that treats computation as the evaluation of mathematical functions
* Eliminates side effects
* Treats data as being immutable
* Functions can take functions as arguments and return functions as results

Example:

**1. Print a list of integers with a lambda**

List<Integer> intSeq = Arrays.asList(1,2,3);

intSeq.forEach(x -> System.out.println(x));

x -> System.out.println(x) is a lambda expression that defines an anonymous function with one parameter named x of type Integer

**2. Multiline**

List<Integer> intSeq = Arrays.asList(1,2,3);

intSeq.forEach(x -> {

x += 2;

System.out.println(x);

});

**3. With a local variable : Just as with ordinary functions**

List<Integer> intSeq = Arrays.asList(1,2,3);

intSeq.forEach(x -> {

int y = x \* 2;

System.out.println(y);

});

**Functional Interfaces**

* Design decision: Java 8 lambdas are assigned to functional interfaces.
* A functional interface is a Java interface with exactly one non-default method. E.g.,

public interface Consumer<T> {

void accept(T t);

}

* The package java.util.function defines many new useful functional interfaces.

Example:

public interface Consumer<T> {

void accept(T t);

}

void forEach(Consumer<Integer> action {

for (Integer i:items) {

action.accept(t);

}

}

List<Integer> intSeq = Arrrays.asList(1,2,3);

Consumer<Integer> cnsmr = x -> System.out.println(x);

intSeq.forEach(cnsmr);

**Local Variable Capture:**

List<Integer> intSeq = Arrays.asList(1,2,3);

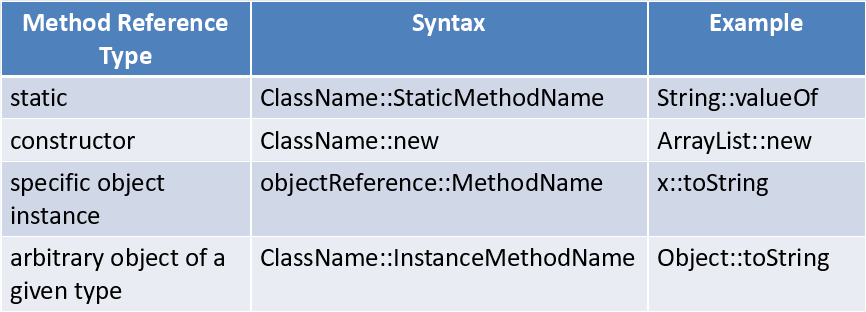
int var = 10; //final or effectively final

intSeq.forEach(x -> System.out.println(x + var));

**Static variable → same way**

**Method reference:**

* Method references can be used to pass an existing function in places where a lambda is expected
* The signature of the referenced method needs to match the signature of the functional interface method



eg:

intSeq.forEach(x -> System.out.println(x));

can be written as ->

intSeq.forEach(System.out::println);

DEMO

**Stream API**

* The new java.util.stream package provides utilities to support functional-style operations on streams of values.
* A common way to obtain a stream is from a collection:
* Stream<T> stream = collection.stream();
* Streams can be sequential or parallel.
* Streams are useful for selecting values and performing actions on the results.

**Pipeline:**

* 3 parts: **Source, Intermediate Operations, Terminal Operation**
* Lazy
* Terminal operation must be final operation

**Some Intermediate Operations:**

* filter excludes all elements that don’t match a Predicate.
* map performs a one-to-one transformation of elements using a Function.

Example:

double developerAverageAge = allEmployees

.stream()

.filter(e -> e.position == Type.DEV)

.mapToInt(e -> e.age)

.sum();

**Creating Streams**

* From individual values
  + Stream.of(val1, val2, …)
* From array
  + Stream.of(someArray)
  + Arrays.stream(someArray)
* From List (and other Collections)
  + someList.stream()
  + someOtherCollection.stream()

**Examples:**

Intermediate Methods

map, filter, distinct, sorted, peek, limit, parallel → they produce another stream (can be of different type)

Terminal Methods

forEach, toArray, reduce, collect, min, max, count, anyMatch, allMatch, noneMatch, findFirst, findAny, iterator

Short-circuit Methods – like break

anyMatch, allMatch, noneMatch, findFirst, findAny, limit, skip

**Other Items**

Default Methods

String Join

New Date API

Optional<T>

**Tired of NPE ? USE → Optional<T> class**

* A container which may or may not contain a non-null value
* Common methods
  + isPresent() – returns true if value is present
  + Get() – returns value if present
  + orElse(T other) – returns value if present, or other
  + ifPresent(Consumer) – runs the lambda if value is present