Introduction to Lambda Expression in Java 8

24Th and 25th Sept'19

Dhanashree Jawle

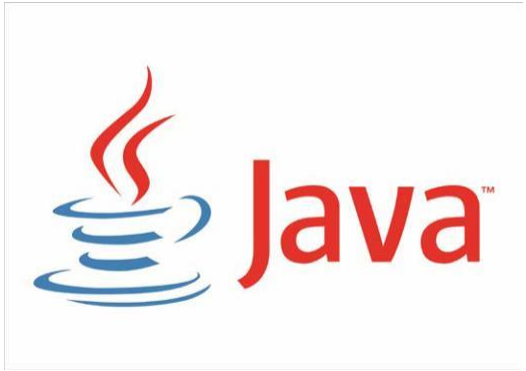
What You Will Learn

- Lambda expressions

Mastek

Targeted Audience

- This is a Java course
- Basic knowledge of the main APIs
- Generics
- Collection API
- Java I/O



Mastek

Module Outline

- Introduction to the « Lambda expressions »

Mastek

Module Outline

- Introduction to the « Lambda expressions »
- The lambda syntax

Mastek

Module Outline

- Introduction to the « Lambda expressions »
- The lambda syntax
- Functional interfaces

Mastek

Module Outline

- Introduction to the « Lambda expressions »
- The lambda syntax
- Functional interfaces
- Method references
- Constructor references

Mastek

Module Outline

- Introduction to the « Lambda expressions »
- The lambda syntax
- Functional interfaces
- Method references
- Constructor references
- How to process data from the Collection API?

Mastek

What Is a Lambda Expression for?

- A simple example

```
public interface FileFilter {

    boolean accept(File file) ;
    }
```

Mastek

What Is a Lambda Expression for?

- Let's implement this interface

```
public class JavaFileFilter implements FileFilter {

   public boolean accept(File file) {
      return file.getName().endsWith(".java") ;
    }
    }
```

Mastek

What Is a Lambda Expression for?

- Let's implement this interface

```
public class JavaFileFilter implements FileFilter {

   public boolean accept(File file) {
      return file.getName().endsWith(".java");
    }
    }
```

- And use it:

```
JavaFileFilter fileFilter = new JavaFileFilter();
File dir = new File("d:/tmp");
File[] javaFiles = dir.listFiles(fileFilter);
```

Mastek

What Is a Lambda Expression for?

- Let's use an anonymous class

```
FileFilter fileFilter = new FileFilter() {

   @Override
   public boolean accept(File file) {
      return file.getName().endsWith(".java");
   }
};

File dir = new File("d:/tmp");
File[] javaFiles = dir.listFiles(fileFilter);
```

Mastek

What Is a Lambda Expression for?

- The first answer is :

  To make instances of anonymous classes easier to write and read!

Mastek

A First Lambda Expression

- Let's use an anonymous class

```
FileFilter fileFilter = new FileFilter() {

   @Override
   public boolean accept(File file) {
      return file.getName().endsWith(".java");
   }
};
```

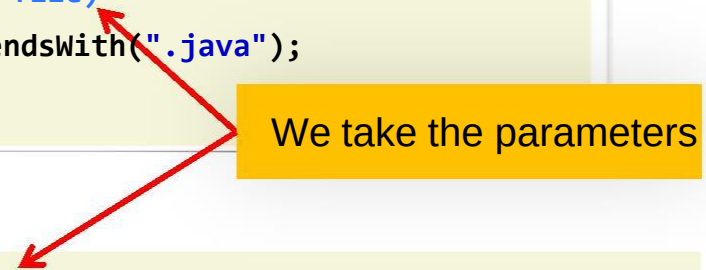
Mastek

A First Lambda Expression

- Let's use an anonymous class

```
FileFilter fileFilter = new FileFilter() {

   @Override
   public boolean accept(File file)
     return file.getName(). .endsWith(".java");
   }
};
```




```
FileFilter filter = (File file)
```

A First Lambda Expression

- Let's use an anonymous class

```
FileFilter fileFilter = new FileFilter() {

   @Override
   public boolean accept(File file)
      return file.getName().endsWith(".java");
   }
};
```

and then…

```
FileFilter filter = (File file) ->
```

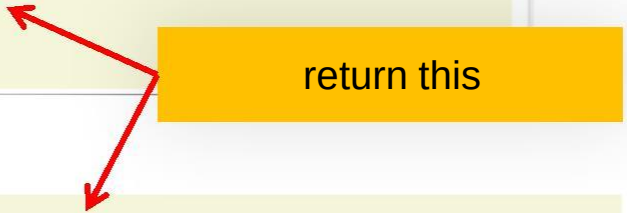
Mastek

A First Lambda Expression

- Let's use an anonymous class

```
FileFilter fileFilter = new FileFilter() {

   @Override
   public boolean accept(File file)
      return file.getName().endsWith(".java");
   }
};
```




```
FileFilter filter = (File file) -> file.getName().endsWith(".java");
```

Mastek

A First Lambda Expression

- Let's use an anonymous class

```
FileFilter fileFilter = new FileFilter() {

   @Override
   public boolean accept(File file) {
      return file.getName().endsWith(".java");
   }
};
```

- This is a Java 8 lambda expression:

```
FileFilter filter = (File file) -> file.getName().endsWith(".java");
```

Mastek

So What Is a Java 8 Lambda Expression?

Answer:

Mastek

So What Is a Java 8 Lambda Expression?

 Answer: another way of writing instances of anonymous classes

Mastek

Several Ways of Writing a Lambda Expression

 Live coding : FileFilter, Runnable, Comparator

Mastek

Several Ways of Writing a Lambda Expression

- The simplest way:

```
FileFilter filter = (File file) -> file.getName().endsWith(".java");
```

Mastek

Several Ways of Writing a Lambda Expression

- The simplest way:

```
FileFilter filter = (File file) -> file.getName().endsWith(".java");
```

- If I have more than one line of code:

```
Runnable r = () -> {
   for (int i = 0; i < 5; i++) {
      System.out.println("Hello world!");
   }
};
```

Mastek

Several Ways of Writing a Lambda Expression

- If I have more than one argument:

```
Comparator<String> c =
   (String s1, String s2) ->
      Integer.compare(s1.length(), s2.length());
```

Mastek

Three Questions About Lambdas

- What is the type of a lambda expression?

Mastek

Three Questions About Lambdas

- What is the type of a lambda expression?

- Can a lambda be put in a variable?

Mastek

Three Questions About Lambdas

- What is the type of a lambda expression?
- Can a lambda be put in a variable?
- Is a lambda expression an object?

Mastek

What Is the Type of a Lambda Expression?

- Answer: a functional interface

Mastek

What Is the Type of a Lambda Expression?

- Answer: a functional interface
- What is a functional interface?

Mastek

Functional Interface

- A functional interface is an interface with only one abstract method

Mastek

Functional Interface

- A functional interface is an interface with only one abstract method
- Example:

```
public interface Runnable {

    run();
    };
```

Mastek

Functional Interface

- A functional interface is an interface with only one abstract method
- Example:

```
public interface Runnable {

    run();
    };
```

```
public interface Comparator<T> {

    int compareTo(T t1, T t2);
    };
```

Mastek

Functional Interface

- A functional interface is an interface with only one abstract method
- Example:

```
public interface Runnable {

    run();
    };
```

```
public interface Comparator<T> {

    int compareTo(T t1, T t2);
    };
```

```
public interface FileFilter {

    boolean accept(File pathname);
    };
```

Mastek

Functional Interface

- A functional interface is an interface with only one abstract method
- Methods from the Object class don't count:

```
public interface MyFunctionalInterface {

someMethod();

/**
  * Some more documentation
  */
  equals(Object o);
  };
```

Mastek

Functional Interface

- A functional interface can be annotated

```
@FunctionalInterface
public interface MyFunctionalInterface {

   someMethod();

   /**
      * Some more documentation
      */
      equals(Object o);
      };
```

- It is just here for convenience, the compiler can tell me whether the interface is functional or not

Mastek

Three Questions About Lambdas

- What is the type of a lambda expression?
  Answer: a functional interface

- Can a lambda be put in a variable?

- Is a lambda expression an object?

Mastek

Can I Put a Lambda Expression in a Variable?

- Answer is yes!

```
Comparator<String> c =
   (String s1, String s2) ->
      Integer.compare(s1.length(), s2.length());
```

Mastek

Can I Put a Lambda Expression in a Variable?

- Answer is yes!

```
Comparator<String> c =
   (String s1, String s2) ->
      Integer.compare(s1.length(), s2.length());
```

- Consequences: a lambda can be taken as a method parameter, and can be returned by a method

Mastek

Three Questions About Lambdas

- What is the type of a lambda expression?
  Answer: a functional interface

- Can a lambda be put in a variable?
  Answer: a functional interface

- Is a lambda expression an object?

Mastek

Is a Lambda an Object?

This question is tougher than it seems…

Mastek

Is a Lambda an Object?

Let's compare the following:

```
Comparator<String> c =
   (String s1, String s2) ->
      Integer.compare(s1.length(), s2.length());
```

```
Comparator<String> c =
   new Comparator<String>(String s1, String s2) {

      public boolean compareTo(String s1, String s2) {
         Integer.compare(s1.length(), s2.length());
      }
   };
```

Mastek

Is a Lambda an Object?

- Let's compare the following:

```
Comparator<String> c =
   (String s1, String s2) ->
      Integer.compare(s1.length(), s2.length());
```

```
Comparator<String> c =
   new Comparator<String>(String s1, String s2) {

      public boolean compareTo(String s1, String s2) {
         Integer.compare(s1.length(), s2.length());
      }
   };
```

- A lambda expression is created without using « new »

Mastek

Three Questions About Lambdas

- What is the type of a lambda expression?
  Answer: a functional interface

- Can a lambda be put in a variable?
  Answer: a functional interface

- Is a lambda expression an object?
  Answer:
  The answer is complex, but no
  Exact answer: a lambda is an object without an identity

Mastek

Summary

- The new « lambda expression » syntax
- A lambda expression has a type : a functional interface

Mastek

Functional Interfaces Toolbox

- New package : java.util.function
- With a rich set of functional interfaces

Mastek

Package java.util.function

- 4 categories:
- Supplier

```
@FunctionalInterface
public interface Supplier<T> {

    T get();
    }
```

Mastek

Package java.util.function

- 4 categories:
- Consumer

```
@FunctionalInterface
public interface Consumer<T> {

    void accept(T t);
    }
```

Mastek

Package java.util.function

- 4 categories:
- Consumer / BiConsumer

```
@FunctionalInterface
public interface Consumer<T> {

    void accept(T t);
    }
```

```
@FunctionalInterface
public interface BiConsumer<T, U> {

    void accept(T t, U u);
    }
```

Mastek

Package java.util.function

- 4 categories:
- Predicate

```
@FunctionalInterface
public interface Predicate<T> {

    boolean test(T t);
    }
```

Mastek

Package java.util.function

- 4 categories:
- Predicate / BiPredicate

```
@FunctionalInterface
public interface Predicate<T> {

    boolean test(T t);
    }
```

```
@FunctionalInterface
public interface BiPredicate<T, U> {

    boolean test(T t, U u);
    }
```

Mastek

Package java.util.function

- 4 categories:
- Function

```
@FunctionalInterface
public interface Function<T, R> {

    R apply (T t);
    }
```

Mastek

Package java.util.function

- 4 categories:
- Function / BiFunction

```
@FunctionalInterface
public interface Function<T, R> {

    R apply (T t);
    }
```

```
@FunctionalInterface
public interface BiFunction<T, U, R> {

    R apply (T t, U u);
    }
```

Mastek

Package java.util.function

- 4 categories:
- Function / UnaryOperator

```
@FunctionalInterface
public interface Function<T, R> {

    R apply (T t);
    }
```

```
@FunctionalInterface
public interface UnaryOperator<T> extends Function<T, T> {
}
```

Mastek

Package java.util.function

- 4 categories:
- BiFunction / BinaryOperator

```
@FunctionalInterface
public interface Function<T, U, R> {

    R apply (T t, U u);
    }
```

```
@FunctionalInterface
public interface BinaryOperator<T> extends BiFunction<T, T, T> {
}
```

Mastek

More Lambda Expressions Syntax

- Most of the time, parameter types can be omitted

```
Comparator<String> c =
   (String s1, String s2) ->
      Integer.compare(s1.length(), s2.length());
```

- Becomes:

```
Comparator<String> c =
   (s1, s2) ->
      Integer.compare(s1.length(), s2.length());
```

Mastek

Method References

- This lambda expression:

```
Function<String, String> f = s -> s.toLowerCase();
```

- Can be written like that:

```
Function<String , String> f = String::toLowerCase;
```

Mastek

Method References

- This lambda expression:

```
Consumer<String> c = s -> System.out.println(s);
```

- Can be written like that:

```
Consumer<String> c = System.out::println;
```

Mastek

Method References

- This lambda expression:

```
Comparator<Integer> c = (i1, i2) -> Integer.compare(i1, i2);
```

- Can be written like that:

```
Comparator<Integer> c = Integer::compare;
```

Mastek

So What Do We Have so Far?

- A new concept: the « lambda expression », with a new syntax

Mastek

So What Do We Have so Far?

- A new concept: the « lambda expression », with a new syntax
- A new interface concept: the « functional interface »

Mastek

So What Do We Have so Far?

- A new concept: the « lambda expression », with a new syntax
- A new interface concept: the « functional interface »

- Question: how can we use this to process data?

Mastek

How Do We Process Data in Java?

 Where are our objects?

Mastek

How Do We Process Data in Java?

- Where are our objects?
- Most of the time: in a Collection (or maybe a List, a Set or a Map)

Mastek

How Do We Process Data in Java?

- Where are our objects?
- Most of the time: in a Collection (or maybe a List, a Set or a Map)
- Can I process this data with lambdas?

```
List<Customer> list = ...;
list.forEach(customer -> System.out.println(customer));
```

Mastek

How Do We Process Data in Java?

- Where are our objects?
- Most of the time: in a Collection (or maybe a List, a Set or a Map)
- Can I process this data with lambdas?

```
List<Customer> list = ...;
list.forEach(customer -> System.out.println(customer));
```

- Or:

```
List<Customer> list = ...;
list.forEach(System.out::println);
```

Mastek

Can I Process This Data with Lambdas?

The good news is: yes!

Mastek

Can I Process This Data with Lambdas?

- The good news is: yes!
- We can write:

```
List<Customer> list = ...;
list.forEach(System.out::println);
```

Mastek

Can I Process This Data with Lambdas?

- The good news is: yes!
- We can write:

```
List<Customer> list = ...;
list.forEach(System.out::println);
```

- But… where does this forEach method come from?

Mastek

Can I Process This Data with Lambdas?

- The good news is: yes!
- We can write:

```
List<Customer> list = ...;
list.forEach(System.out::println);
```

- But… where does this forEach method come from?
- Adding a forEach method on the Collection interface breaks the compatibility: all the implementations have to be refactored!

Mastek

How to Add Methods to Iterable?

- Without breaking all the existing implementations?

```
public interface Iterable<E> {

// the usual methods

void forEach(Consumer<E> consumer);
}
```

Mastek

How to Add Methods to Iterable?

- Without breaking all the existing implementations?

```
public interface Iterable<E> {

// the usual methods

void forEach(Consumer<E> consumer);
}
```

- Refactoring these implementations is not an option

Mastek

How to Add Methods to Iterable?

- If we cant put the implementation in ArrayList, then…

```
public interface Iterable<E> {

// the usual methods

default void forEach(Consumer<E> consumer) {

  for (E e : this) {
     consumer.accept(e);
  }
}
}
```

Mastek

Default Methods

- This is a new Java 8 concept
- It allows to change the old interfaces without breaking the existing implementations

Mastek

Default Methods

- This is a new Java 8 concept
- It allows to change the old interfaces without breaking the existing implementations
- It also allows new patterns!

Mastek

Default Methods

- This is a new Java 8 concept
- It allows to change the old interfaces without breaking the existing implementations
- It also allows new patterns!

- And by the way…

Mastek

Default Methods

- This is a new Java 8 concept
- It allows to change the old interfaces without breaking the existing implementations
- It also allows new patterns!

- And by the way…
- Static methods are also allowed in Java 8 interfaces!

Mastek

Examples Of New Patterns

- Predicates

```
Predicate<String> p1 = s -> s.length() < 20;
Predicate<String> p2 = s -> s.length() > 10;
```

Mastek

Examples Of New Patterns

- Predicates

```
Predicate<String> p1 = s -> s.length() < 20;
Predicate<String> p2 = s -> s.length() > 10;

Predicate<String> p3 = p1.and(p2);
```

Mastek

Examples Of New Patterns

- Predicates

```
Predicate<String> p1 = s -> s.length() < 20;
Predicate<String> p2 = s -> s.length() > 10;

Predicate<String> p3 = p1.and(p2);
```

```
@FunctionalInterface
public interface Predicate<T> {

   boolean test(T t);

   default Predicate<T> and(Predicate<? super T> other) {
      Objects.requireNonNull(other);
      return (t) -> test(t) && other.test(t);
    }
    }
```

Mastek

Examples Of New Patterns

- Predicates

```
Predicate<String> id = Predicate.isEqual(target);
```

Mastek

Examples Of New Patterns

- Predicates

```
Predicate<String> id = Predicate.isEqual(target);
```

```
@FunctionalInterface
public interface Predicate<T> {

   boolean test(T t);

   static <T> Predicate<T> isEqual(Object targetRef) {
      return (null == targetRef)
             ? Objects::isNull
             : object -> targetRef.equals(object);
    }
    }
```

Mastek

Summary

- Definition of a functional interface, examples
- Method and constructor references
- Iterable.forEach method
- Default and static methods in interfaces, examples

Mastek

Module Outline

. Introduction: map / filter / reduce

. What is a « Stream »?

. Patterns to build a Stream

. Operations on a Stream

Mastek

Map / Filter / Reduce

. Example:

. Let’s take a list a Person

```
List<Person> list = new ArrayList<>() ;
```

Mastek

Map / Filter / Reduce

. Example:

. Let’s take a list a Person

```
List<Person> list = new ArrayList<>() ;
```

. Suppose we want to compute the

« average of the age of the people older than 20 »

Mastek

Map / Filter / Reduce

. 1st step: mapping

Mastek

Map / Filter / Reduce

. 1st step: mapping

. The mapping step takes a List<Person> and returns a List<Integer>

. The size of both lists is the same

Mastek

Map / Filter / Reduce

. 2nd step: filtering

. The filtering step takes a List<Integer> and returns a List<Integer>

. But there some elements have been filtered out in the process

Mastek

Map / Filter / Reduce

. 3rd step: average

. This is the reduction step, equivalent to the SQL aggregation

Mastek

What Is a Stream?

. Technical answer: a typed interface

```
public interface Stream<T> extends BaseStream<T, Stream<T>> {

            // ...
            }
```

Mastek

What Is a Stream?

. Technical answer: a typed interface

```
public interface Stream<T> extends BaseStream<T, Stream<T>> {

            // ...
            }
```

. And a new concept!

Mastek

What Is a Stream?

. What does it do?

Mastek

What Is a Stream?

. What does it do?

. It gives ways to efficiently process large amounts of data… and also smaller ones

Mastek

What Is a Stream?

. What does efficiently mean?

Mastek

What Is a Stream?

. What does efficiently mean?

. Two things:

Mastek

What Is a Stream?

. What does efficiently mean?

. Two things:

. In parallel, to leverage the computing power of multicore CPUs

. Pipelined, to avoid unnecessary intermediary computations

Mastek

What Is a Stream?

. Why can’t a Collection be a Stream?

Mastek

What Is a Stream?

. Why can’t a Collection be a Stream?

. Because Stream is a new concept, and we dont want to change the way the Collection API works

Mastek

What Is a Stream?

. So what is a Stream?

Mastek

What Is a Stream?

. So what is a Stream?

. An object on which one can define operations

Mastek

What Is a Stream?

. So what is a Stream?

. An object on which one can define operations

. An object that does not hold any data

Mastek

What Is a Stream?

. So what is a Stream?

. An object on which one can define operations

. An object that does not hold any data

. An object that should not change the data it processes

Mastek

What Is a Stream?

. So what is a Stream?

. An object on which one can define operations

. An object that does not hold any data

. An object that should not change the data it processes

. An object able to process data in « one pass »

Mastek

What Is a Stream?

. So what is a Stream?

. An object on which one can define operations

. An object that does not hold any data

. An object that should not change the data it processes

. An object able to process data in « one pass »

. An object optimized from the algorithm point of view, and able to process data in parallel

Mastek

How Can We Build a Stream?

. Many patterns!

Mastek

How Can We Build a Stream?

. Many patterns!

```
List<Person> persons = ... ;

Stream<Person> stream = persons.stream();
```

Mastek

A First Operation

. First operation: forEach()

```
List<Person> persons = ... ;

Stream<Person> stream = persons.stream();
stream.forEach(p -> System.out.println(p));
```

Mastek

A First Operation

. First operation: forEach()

```
List<Person> persons = ... ;

Stream<Person> stream = persons.stream();
stream.forEach(p -> System.out.println(p));
```

. Prints all the elements of the list

Mastek

A First Operation

. First operation: forEach()

```
List<Person> persons = ... ;

Stream<Person> stream = persons.stream();
stream.forEach(p -> System.out.println(p));
```

. Prints all the elements of the list

. It takes an instance of Consumer as an argument

Mastek

A First Operation

. Interface Consumer<T>

```
@FunctionalInterface
public interface Consumer<T> {

      void accept(T t);
      }
```

Mastek

A First Operation

. Interface Consumer<T>

```
@FunctionalInterface
public interface Consumer<T> {

      void accept(T t);
      }
```

. Consumer<T> is a functional interface

. Can be implemented by a lambda expression

```
Consumer<T> c = p -> System.out.println(p);
```

Mastek

A First Operation

. Interface Consumer<T>

```
@FunctionalInterface
public interface Consumer<T> {

      void accept(T t);
      }
```

. Consumer<T> is a functional interface

. Can be implemented by a lambda expression

```
Consumer<T> c = p -> System.out.println(p);
```

```
Consumer<T> c = System.out::println; // Method reference
```

Mastek

A First Operation

. In fact Consumer<T> is a bit more complex

```
@FunctionalInterface
public interface Consumer<T> {

    void accept(T t);

    default Consumer<T> andThen(Consumer<? super T> after) {
        Objects.requireNonNull(after);
        return (T t) -> { accept(t); after.accept(t); };
    }
}
```

Mastek

A First Operation

. In fact Consumer<T> is a bit more complex

```
@FunctionalInterface
public interface Consumer<T> {

    void accept(T t);

    default Consumer<T> andThen(Consumer<? super T> after) {
        Objects.requireNonNull(after);
        return (T t) -> { accept(t); after.accept(t); };
    }
}
```

. One can chain consumers!

Mastek

A First Operation

. Let’s chain consumers

```
List<String> list = new ArrayList<>();

Consumer<String> c1 = s -> list.add(s);
Consumer<String> c2 = s -> System.out.println(s);
```

Mastek

A First Operation

. Let’s chain consumers

```
List<String> list = new ArrayList<>();

Consumer<String> c1 = list::add;
Consumer<String> c2 = System.out::println;
```

Mastek

A First Operation

. Let’s chain consumers

```
List<String> list = new ArrayList<>();

Consumer<String> c1 = list::add;
Consumer<String> c2 = System.out::println;

Consumer<String> c3 = c1.andThen(c2);
```

Mastek

A First Operation

. Only way to have several consumers on a single stream

```
List<String> result = new ArrayList<>();
List<Person> persons = ...;

Consumer<String> c1 = result::add;
Consumer<String> c2 = System.out::println;

persons.stream()
        .forEach(c1.andThen(c2));
```

. Because forEach() does not return anything

Mastek

A Second Operation: Filter

. Example:

```
List<Person> list = ...;
Stream<Person> stream = list.stream();
Stream<Person> filtered =
    stream.filter(person -> person.getAge() > 20);
```

Mastek

A Second Operation: Filter

. Example:

```
List<Person> list = ...;
Stream<Person> stream = list.stream();
Stream<Person> filtered =
    stream.filter(person -> person.getAge() > 20);
```

. Takes a predicate as a parameter:

```
Predicate<Person> p = person -> person.getAge() > 20;
```

Mastek

A Second Operation: Filter

. Predicate interface:

```
@FunctionalInterface
public interface Predicate<T> {

      boolean test(T t);
      }
```

Mastek

A Second Operation: Filter

. Predicate interface, with default methods:

```
@FunctionalInterface
public interface Predicate<T> {

    boolean test(T t);

    default Predicate<T> and(Predicate<? super T> other) { ... }

    default Predicate<T> or(Predicate<? super T> other) { ... }

     default Predicate<T> negate() { ... }
     }
```

Mastek

A Second Operation: Filter

. Predicates combinations examples:

```
Predicate<Integer> p1 = i -> i > 20;
Predicate<Integer> p2 = i -> i < 30;
Predicate<Integer> p3 = i -> i == 0;

Predicate<Integer> p = p1.and(p2).or(p3); // (p1 AND p2) OR p3
Predicate<Integer> p = p3.or(p1).and(p2); // (p3 OR p1) AND p2
```

Mastek

A Second Operation: Filter

. Predicates combinations examples:

```
Predicate<Integer> p1 = i -> i > 20;
Predicate<Integer> p2 = i -> i < 30;
Predicate<Integer> p3 = i -> i == 0;

Predicate<Integer> p = p1.and(p2).or(p3); // (p1 AND p2) OR p3
Predicate<Integer> p = p3.or(p1).and(p2); // (p3 OR p1) AND p2
```

. Warning: method calls do not handle priorities

Mastek

A Second Operation: Filter

. Predicate interface, with static method:

```
@FunctionalInterface
public interface Predicate<T> {

    boolean test(T t);

    // default methods

     static <T> Predicate<T> isEqual(Object o) { ... }
     }
```

Mastek

A Second Operation: Filter

. Predicate interface, with static method:

```
@FunctionalInterface
public interface Predicate<T> {

    boolean test(T t);

    // default methods

     static <T> Predicate<T> isEqual(Object o) { ... }
     }
```

. Example:

```
Predicate<String> p = Predicate.isEqual("two") ;
```

Mastek

A Second Operation: Filter

. Use case:

```
Predicate<String> p = Predicate.isEqual("two");

Stream<String> stream1 = Stream.of("one", "two", "three");

Stream<String> stream2 = stream1.filter(p);
```

. The filter method returns a Stream

Mastek

A Second Operation: Filter

. Use case:

```
Predicate<String> p = Predicate.isEqual("two");

Stream<String> stream1 = Stream.of("one", "two", "three");

Stream<String> stream2 = stream1.filter(p);
```

. The filter method returns a Stream

. This Stream is a new instance

Mastek

A Second Operation: Filter

. Question: what do I have in this new Stream?

Mastek

A Second Operation: Filter

. Question: what do I have in this new Stream?

. Simple answer: the filtered data

Mastek

A Second Operation: Filter

. Question: what do I have in this new Stream?

. Simple answer: the filtered data

. Really?

Mastek

A Second Operation: Filter

. Question: what do I have in this new Stream?

. Simple answer: the filtered data

. Really?

. We just said: « a stream does not hold any data »

Mastek

A Second Operation: Filter

. Question: what do I have in this new Stream?

. Simple answer: the filtered data WRONG!

Mastek

A Second Operation: Filter

. Question: what do I have in this new Stream?

. Simple answer: the filtered data WRONG!

. The right answer is: nothing, since a Stream does not hold any data

Mastek

A Second Operation: Filter

. Question: what do I have in this new Stream?

. Simple answer: the filtered data WRONG!

. The right answer is: nothing, since a Stream does not hold any data

. So, what does this code do?

```
List<Person> list = ... ;
Stream<Person> stream = list.stream();
Stream<Person> filtered =
        stream.filter(person -> person.getAge() > 20);
```

Mastek

A Second Operation: Filter

. Question: what do I have in this new Stream?

. Simple answer: the filtered data WRONG!

. The right answer is: nothing, since a Stream does not hold any data

. So, what does this code do?

```
List<Person> list = ... ;
Stream<Person> stream = list.stream();
Stream<Person> filtered =
         stream.filter(person -> person.getAge() > 20);
```

. Answer is: nothing

This call is only a declaration, no data is processed

Mastek

A Second Operation: Filter

. The call to the filter method is lazy

Mastek

A Second Operation: Filter

. The call to the filter method is lazy

. And all the methods of Stream that return another Stream are lazy

Mastek

A Second Operation: Filter

. The call to the filter method is lazy

. And all the methods of Stream that return another Stream are lazy

. Another way of saying it:

an operation on a Stream that returns a Stream
is called an intermediary operation

Mastek

Back to the Consumer

. What does this code do?

```
List<String> result = new ArrayList<>();
List<Person> persons = ... ;

persons.stream()
        .peek(System.out::println)
        .filter(person -> person.getAge() > 20)
        .peek(result::add);
```

Mastek

Back to the Consumer

. What does this code do?

```
List<String> result = new ArrayList<>();
List<Person> persons = ... ;

persons.stream()
        .peek(System.out::println)
        .filter(person -> person.getAge() > 20)
        .peek(result::add);
```

. Hint: the peek() method returns a Stream

Mastek

Back to the Consumer

. What does this code do?

```
List<String> result = new ArrayList<>();
List<Person> persons = ... ;

persons.stream()
        .peek(System.out::println)
        .filter(person -> person.getAge() > 20)
        .peek(result::add);
```

. Answer: nothing!

. This code does not print anything

. The list « result » is empty

Mastek

Summary

. The Stream API defines intermediary operations

. We saw 3 operations:

. forEach(Consumer)

. peek(Consumer)

. filter(Predicate)

Mastek

Summary

- The Stream API defines intermediary operations
- We saw 3 operations.
  1. forEach(Consumer) (not lazy)
  2. peek(Consumer) (lazy)
  3. filter(Predicate) (lazy)

Mastek

Mapping Operation

. Example:

```
List<Person> list = ... ;
Stream<Person> stream = list.stream();
Stream<String> names =
        stream.map(person -> person.getName());
```

Mastek

Mapping Operation

. Example:

```
List<Person> list = ... ;
Stream<Person> stream = list.stream();
Stream<String> names =
         stream.map(person -> person.getName());
```

. map() returns a Stream, so it is an intermediary operation

Mastek

Mapping Operation

. A mapper is modeled by the Function interface

```
@FunctionalInterface
public interface Function<T, R> {

      R apply(T t);
      }
```

Mastek

Mapping Operation

. … with default methods to chain and compose mappings

```
@FunctionalInterface
public interface Function<T, R> {

    R apply(T t);

    default <V> Function<V, R> compose(Function<V, T> before);

     default <V> Function<T, V> andThen(Function<R, V> after);
     }
```

Mastek

Mapping Operation

. … with default methods to chain and compose mappings

```
@FunctionalInterface
public interface Function<T, R> {

    R apply(T t);

    default <V> Function<V, R> compose(Function<V, T> before);

     default <V> Function<T, V> andThen(Function<R, V> after);
     }
```

. In fact this is the simplified version, beware the generics!

Mastek

Mapping Operation

. compose() and andThen() methods with their exact signatures

```
@FunctionalInterface
public interface Function<T, R> {

    R apply(T t);

    default <V> Function<V, R> compose(
        Function<? super V, ? extends T> before);

     default <V> Function<T, V> andThen(
     Function<? super R, ? extends V> after);
     }
```

Mastek

Mapping Operation

. One static method: identity

```
@FunctionalInterface
public interface Function<T, R> {

    R apply(T t);

    // default methods

    static <T> Function<T, T> identity() {
        return t -> t;
    }
}
```

Mastek

Flatmapping Operation

. Method flatMap()

. Signature:

```
<R> Stream<R> flatMap(Function<T, Stream<R>> flatMapper);
```

```
<R> Stream<R> map(Function<T, R> mapper);
```

Mastek

Flatmapping Operation

. Method flatMap()

. Signature:

```
<R> Stream<R> flatMap(Function<T, Stream<R>> flatMapper);
```

```
<R> Stream<R> map(Function<T, R> mapper);
```

. The flatMapper takes an element of type T, and returns an element of type Stream<R>

Mastek

Flatmapping Operation

. Method flatMap()

. Signature:

```
<R> Stream<R> flatMap(Function<T, Stream<R>> flatMapper);
```

```
<R> Stream<R> map(Function<T, R> mapper);
```

. If the flatMap was a regular map, it would return a Stream<Stream<R>>

Mastek

Flatmapping Operation

. Method flatMap()

. Signature:

```
<R> Stream<R> flatMap(Function<T, Stream<R>> flatMapper);
```

```
<R> Stream<R> map(Function<T, R> mapper);
```

. If the flatMap was a regular map, it would return a Stream<Stream<R>>

. Thus a « stream of streams »

Mastek

Flatmapping Operation

. Method flatMap()

. Signature:

```
<R> Stream<R> flatMap(Function<T, Stream<R>> flatMapper);
```

```
<R> Stream<R> map(Function<T, R> mapper);
```

. If the flatMap was a regular map, it would return a Stream<Stream<R>>

. But it is a flatMap!

Mastek

Flatmapping Operation

. Method flatMap()

. Signature:

```
<R> Stream<R> flatMap(Function<T, Stream<R>> flatMapper);
```

```
<R> Stream<R> map(Function<T, R> mapper);
```

. If the flatMap was a regular map, it would return a Stream<Stream<R>>

. But it is a flatMap!

. Thus the « stream of streams » is flattened, and becomes a stream

Mastek

Summary

- 3 Categories of operations.
  1. forEach(Consumer) and peek()
  2. filter(Predicate) (lazy)
  3. map() and flatMap()

Mastek

Reduction

. And what about the reduction step?

Mastek

Reduction

. And what about the reduction step?

. Two kinds of reduction in the Stream API

Mastek

Reduction

. And what about the reduction step?

. Two kinds of reduction in the Stream API

. 1st: aggregation = min, max, sum, etc…

Mastek

Reduction

. How does it work?

```
List<Integer> ages = ... ;
Stream<Integer> stream = ages.stream();
Integer sum =
        stream.reduce(0, (age1, age2) -> age1 + age2);
```

Mastek

Reduction

. How does it work?

```
List<Integer> ages = ... ;
Stream<Integer> stream = ages.stream();
Integer sum =
        stream.reduce(0, (age1, age2) -> age1 + age2);
```

. 1st argument: identity element of the reduction operation

Mastek

Reduction

. How does it work?

```
List<Integer> ages = ... ;
Stream<Integer> stream = ages.stream();
Integer sum =
         stream.reduce(0, (age1, age2) -> age1 + age2);
```

. 1st argument: identity element of the reduction operation

. 2nd argument: reduction operation, of type BinaryOperator<T>

Mastek

BinaryOperator

. A BinaryOperator is a special case of BiFunction

```
@FunctionalInterface
public interface BiFunction<T, U, R> {

    R apply(T t, U u);

     // plus default methods
     }
```

Mastek

BinaryOperator

. A BinaryOperator is a special case of BiFunction

```
@FunctionalInterface
public interface BiFunction<T, U, R> {

    R apply(T t, U u);

     // plus default methods
     }
```

```
@FunctionalInterface
public interface BinaryOperator<T>
extends BiFunction<T, T, T> {

    // T apply(T t1, T t2);

     // plus static methods
     }
```

Mastek

Identity Element

. The bifunction takes two arguments, so…

Mastek

Identity Element

. The bifunction takes two arguments, so…

. What happens if the Stream is empty?

Mastek

Identity Element

. The bifunction takes two arguments, so…

. What happens if the Stream is empty?

. What happens if the Stream has only one element?

Mastek

Identity Element

. The bifunction takes two arguments, so…

. What happens if the Stream is empty?

. What happens if the Stream has only one element?

. The reduction of an empty Stream is the identity element

Mastek

Identity Element

. The bifunction takes two arguments, so…

. What happens if the Stream is empty?

. What happens if the Stream has only one element?

. The reduction of an empty Stream is the identity element

. If the Stream has only one element, then the reduction is that element

Mastek

Aggregations

. Examples:

```
Stream<Integer> stream = ...;
BinaryOperation<Integer> sum = (i1, i2) -> i1 + i2;
Integer id = 0; // identity element for the sum

int red = stream.reduce(id, sum);
```

```
Stream<Integer> stream = Stream.empty();

int red = stream.reduce(id, sum);
System.out.println(red);
```

. Will print:

```
> 0
```

Mastek

Aggregations

. Examples:

```
Stream<Integer> stream = ...;
BinaryOperation<Integer> sum = (i1, i2) -> i1 + i2;
Integer id = 0; // identity element for the sum

int red = stream.reduce(id, sum);
```

```
Stream<Integer> stream = Stream.of(1);

int red = stream.reduce(id, sum);
System.out.println(red);
```

. Will print:

```
> 1
```

Mastek

Aggregations

. Examples:

```
Stream<Integer> stream = ...;
BinaryOperation<Integer> sum = (i1, i2) -> i1 + i2;
Integer id = 0; // identity element for the sum

int red = stream.reduce(id, sum);
```

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4);

int red = stream.reduce(id, sum);
System.out.println(red);
```

. Will print:

```
> 10
```

Mastek

Aggregations: Corner Case

. Suppose the reduction is the max

```
BinaryOperation<Integer> max =
(i1, i2) ->
i1 > i2 ? i1 : i2;
```

Mastek

Aggregations: Corner Case

. Suppose the reduction is the max

```
BinaryOperation<Integer> max =
(i1, i2) ->
i1 > i2 ? i1 : i2;
```

. The problem is, there is no identity element for the max reduction

Mastek

Aggregations: Corner Case

. Suppose the reduction is the max

```
BinaryOperation<Integer> max =
(i1, i2) ->
i1 > i2 ? i1 : i2;
```

. The problem is, there is no identity element for the max reduction

. So the max of an empty Stream is undefined…

Mastek

Aggregations: Corner Case

. Then what is the return type of this call?

```
List<Integer> ages = ... ;
Stream<Integer> stream = ages.stream();
... max =
        stream.max(Comparator.naturalOrder());
```

Mastek

Aggregations: Corner Case

. Then what is the return type of the this call?

```
List<Integer> ages = ... ;
Stream<Integer> stream = ages.stream();
... max =
        stream.max(Comparator.naturalOrder());
```

. If it is an int, then the default value is 0…

Mastek

Aggregations: Corner Case

. Then what is the return type of the this call?

```
List<Integer> ages = ... ;
Stream<Integer> stream = ages.stream();
... max =
        stream.max(Comparator.naturalOrder());
```

. If it is an Integer, then the default value is null…

Mastek

Optionals

. Then what is the return type of the this call?

```
List<Integer> ages = ... ;
Stream<Integer> stream = ages.stream();
Optional<Integer> max =
        stream.max(Comparator.naturalOrder());
```

. Optional means « there might be no result »

Mastek

Optionals

. How to use an Optional?

```
Optional<String> opt = ... ;
if (opt.isPresent()) {
        String s = opt.get() ;
} else {
        ...
}
```

. The method isPresent() returns true if there is something in the optional

Mastek

Optionals

. How to use an Optional?

```
Optional<String> opt = ... ;
if (opt.isPresent()) {
        String s = opt.get() ;
} else {
        ...
}
```

. The method isPresent() returns true if there is something in the optional

. The method get() returns the value held by this optional

Mastek

Optionals

. How to use an Optional?

```
Optional<String> opt = ... ;
if (opt.isPresent()) {
        String s = opt.get() ;
} else {
        ...
}
```

. The method orElse() encapsulates both calls

```
String s = opt.orElse("") ; // defines a default value
```

Mastek

Optionals

. How to use an Optional?

```
Optional<String> opt = ... ;
if (opt.isPresent()) {
        String s = opt.get() ;
} else {
        ...
}
```

. The method orElseThrow() defines a thrown exception

```
String s = opt.orElseThrow(MyException::new) ; // lazy construct.
```

Mastek

Reductions

. Available reductions:

- max(), min()
- count()

Mastek

Reductions

. Available reductions:

- max(), min()
- count()

. Boolean reductions

- allMatch(), noneMatch(), anyMatch()

Mastek

Reductions

. Available reductions:
  - max(), min()
  - count()

. Boolean reductions
  - allMatch(), noneMatch(), anyMatch()

. Reductions that return an optional
  - findFirst(), findAny()

Mastek

Reductions

. Reductions are terminal operations

. They trigger the processing of the data

Mastek

Terminal Operation

. Example:

```
List<Person> persons = ...;

Optional<Integer> minAge =
     persons.map(person -> person.getAge())  // Stream<Integer>
     .filter(age -> age > 20)                // Stream<Integer>
     .min(Comparator.naturalOrder()); // terminal operation
```

Mastek

Terminal Operation

. Example, optimization:

```
List<Person> persons = ... ;

persons.map(person -> person.getLastName())
       .allMatch(length < 20);                    // terminal op.
```

Mastek

Terminal Operation

. Example, optimization:

```
List<Person> persons = ... ;

persons.map(person -> person.getLastName())
       .allMatch(length < 20);                    // terminal op.
```

. The map / filter / reduce operations are evaluated in one pass over the data

Mastek

Summary

. Reduction seen as an aggregation

. Intermediary / terminal operation

. Optional: needed because default values cant be always defined

Mastek

Collectors

. There is another type of reduction

Mastek

Collectors

. There is another type of reduction

. Called « mutable » reduction

Mastek

Collectors

. There is another type of reduction

. Called « mutable » reduction

. Instead of aggregating elements, this reduction put them in a « container »

Mastek

Collecting in a String

. Example:

```
List<Person> persons = ... ;

String result =
persons.stream()
        .filter(person -> person.getAge() > 20)
        .map(Person::getLastName)
        .collect(
           Collectors.joining(", ")
        );
```

. Result is a String with all the names of the people in persons, older than 20, separated by a comma

Mastek

Collecting in a List

. Example:

```
List<Person> persons = ... ;

List<String> result =
persons.stream()
            .filter(person -> person.getAge() > 20)
            .map(Person::getLastName)
            .collect(
            Collectors.toList()
            );
```

. Result is a List of String with all the names of the people in persons, older than 20

Mastek

Collecting in a Map

. Example:

```
List<Person> persons = ... ;

Map<Integer, List<Person>> result =
persons.stream()
            .filter(person -> person.getAge() > 20)
            .collect(
            Collectors.groupingBy(Person::getAge)
            );
```

. Result is a Map containing the people of persons, older than 20

- The keys are the ages of the people
- The values are the lists of the people of that age

Mastek

Collecting in a Map

. Example:

```
List<Person> persons = ... ;

Map<Integer, List<Person>> result =
persons.stream()
            .filter(person -> person.getAge() > 20)
            .collect(
            Collectors.groupingBy(Person::getAge)
            );
```

. It is possible to « post-process » the values, with a downstream collector

Mastek

Collecting in a Map

. Example:

```
List<Person> persons = ... ;

Map<Integer, Long> result =
persons.stream()
            .filter(person -> person.getAge() > 20)
            .collect(
            Collectors.groupingBy(Person::getAge),
            Collectors.counting() // the downstream collector
        );
```

. Collectors.counting() just counts the number of people of each age

Mastek

So What Is a Stream?

. An object that allows one to define processings on data
  - There is no limit on the amount of data that can be processed
. Those processings are typically map / filter / reduce operations

Mastek

So What Is a Stream?

. An object that allows one to define processings on data

- There is no limit on the amount of data that can be processed

. Those processings are typically map / filter / reduce operations

. Those processings are optimized :

. First, we define all the operations

. Then, the operations are triggered

Mastek

So What Is a Stream?

. Last remark:

. A Stream cannot be « reused »

. Once it has been used ot process a set of data, it cannot be used again to process another set

Mastek

Summary

. Quick explanation of the map / filter / reduce

. What is a Stream

. The difference between intermediary and final operations

. The « consuming » operations: forEach() and peek()

. The « mapping » operations: map() and flatMap()

. The « filter » operation: filter()

. The « reduction » operations:

- Aggregations: reduce(), max(), min(),
- …
  Mutable reductions: collect, Collectors

Mastek

Thank you!

Mastek™