

CADEC 2015 - JAVA 8

MAGNUS LARSSON

2015.01.28 | CALLISTAENTERPRISE.SE

■ JAVA 8 NEW FEATURES

- Overview
- Lambda Expressions
- Stream API

■ JAVA 8 NEW FEATURES - OVERVIEW

- New Date/Time API
 - Based on [Joda-Time](#)
- Interface improvements
 - Static methods and default methods
- Optional type
 - Say goodbye to NullPointerException
- Concurrency API additions
 - E.g. CompletableFuture and parallel stream operation
- New JavaScript Engine
 - Nashorn
- PermGen space is gone
 - Yeah!
- **Lambda Expressions**
- **Stream API**

■ JAVA 8 NEW FEATURES - OVERVIEW

- New Date/Time API
 - Based on [Joda-Time](#)
- Interface improvements
 - Static methods and default methods
- Optional type
 - Say goodbye to NullPointerException
- Concurrency API additions
 - E.g. CompletableFuture and parallel stream operation
- New JavaScript Engine
 - Nashorn
- PermGen is gone!
 - Yeah!
- **Lambda Expressions**
- **Stream API**

Enables Functional-style Programming

■ JAVA 8 NEW FEATURES

- Overview
- Lambda Expressions
- Stream API

■ JAVA 8 LAMBDA EXPRESSIONS

- Using an *anonymous inner class* for a callback:

```
asyncHttpClient.execute(url,  
  
    new AsyncCompletionHandler<Response>() {  
        @Override  
        public Response onCompleted  
            (Response response) {  
  
            // TODO: Handle the response...  
  
        }  
    }  
);
```

JAVA 8 LAMBDA EXPRESSIONS

- Using an *anonymous inner class* for a callback:

```
asyncHttpClient.execute(url,  
    new AsyncCompletionHandler<Response>() {  
        @Override  
        public Response onCompleted  
            (Response response) {  
            // TODO: Handle the response...  
        }  
    }  
);
```


This is just
overhead!!

JAVA 8 LAMBDA EXPRESSIONS

- Anonymous inner class

```
asyncHttpClient.execute(url,  
    new AsyncCompletionHandler<Response>() {  
        @Override  
        public Response onCompleted  
            (Response response) {  
            // TODO: Handle the response...  
        }  
    }  
);
```

Gone using
Lambdas!!!



- Lambda expression

```
asyncHttpClient.execute(url,  
    (Response response) -> {  
        // TODO: Handle the response...  
    }  
);
```


JAVA 8 LAMBDA EXPRESSIONS

- Type inference

```
asyncHttpClient.execute(url,  
    response -> {  
        // TODO: Handle the response...  
    }  
);
```

- Local variables

```
final DeferredResult<String> dr =  
    new DeferredResult<>();  
  
asyncHttpClient.execute(url,  
    response -> {  
        // TODO: Handle the response...  
        dr.setResult(response.getResponseBody());  
    }  
);
```

SOME OF THE MAGIC BEHIND LAMBDA EXPRESSIONS

- Functional Interfaces

- A functional interface is an interface that defines exactly one *abstract* method.

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

    boolean test(T t);
}
```

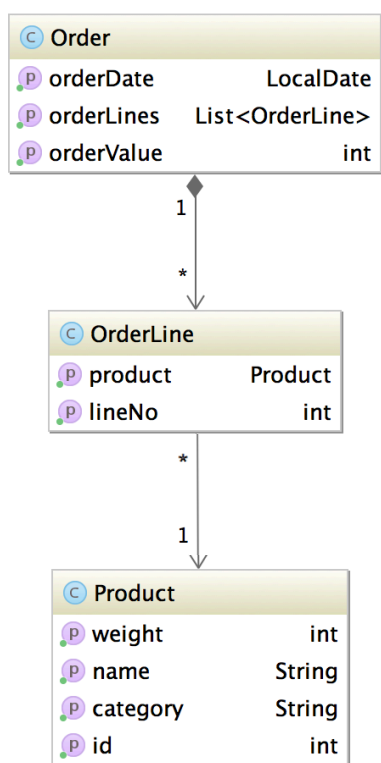
- Functional interfaces can be instantiated using lambda expressions...

■ JAVA 8 NEW FEATURES

- Overview
- Lambda Expressions
- Stream API

INTRODUCTION TO FUNCTIONAL PROGRAMMING

- Consider the following model



- How to implement:

1. What products from category X have been ordered in the date interval M to N?

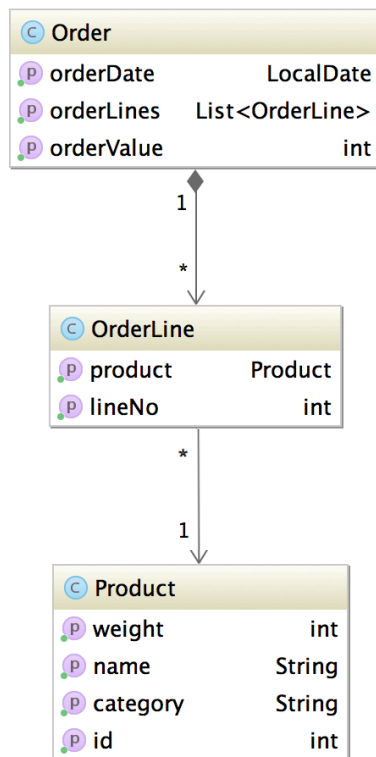
I want them sorted by their weight!

2. What products with weight from X to Y has been sold in orders with an order value M to N?

I want them sorted by their product id!

INTRODUCTION TO FUNCTIONAL PROGRAMMING

- Consider the following model



- Assume the following API:

```
public interface QueryApi {

    public List<Product> getProductsByDateAndCategoryOrderByWeight(
        LocalDate minDate,
        LocalDate maxDate,
        String category);

    public List<Product> getProductsByOrderValueAndWeightOrderByProductId(
        int minOrderValue,
        int maxOrderValue,
        int minProductWeight,
        int maxProductWeight);
}
```

START WITH A IMPERATIVE SOLUTION, #1

```
public List<Product> getProductsByDateAndCategoryOrderByWeight
(LocalDate minDate, LocalDate maxDate, String category) {

    List<Order> orders = getOrders();

    List<Product> products = new ArrayList<>();
    for (Order order : orders) {

        // Filter on order date
        LocalDate date = order.getOrderDate();
        if (date.isAfter(minDate) && date.isBefore(maxDate)) {

            List<OrderLine> orderLines = order.getOrderLines();
            for (OrderLine orderLine : orderLines) {

                // Filter on product category
                Product product = orderLine.getProduct();
                if (product.getCategory().equals(category)) {
                    products.add(product);
                }
            }
        }
    }
    ...
}
```

START WITH A IMPERATIVE SOLUTION, #1

```
        }  
        ...  
    }  
}  
  
// Remove any duplicates from the list of selected products  
products = new ArrayList<>(new HashSet<>(products));  
  
// Sort on product weight  
Collections.sort(products, new Comparator<Product>() {  
    @Override  
    public int compare(Product p1, Product p2) {  
        return (p1.getWeight() < p2.getWeight()) ? -1 : 1;  
    }  
});  
  
return products;  
}
```

START WITH A IMPERATIVE SOLUTION, #2

```
public List<Product> getProductsByOrderValueAndWeightOrderByProductId(
    int minOrderValue, int maxOrderValue, int minProductWeight, int maxProductWeight) {

    List<Order> orders = getOrders();

    List<Product> products = new ArrayList<>();
    for (Order order : orders) {

        // Filter on order value
        int orderValue = order.getOrderValue();
        if (minOrderValue <= orderValue && orderValue <= maxOrderValue) {

            List<OrderLine> orderLines = order.getOrderLines();
            for (OrderLine orderLine : orderLines) {

                // Filter on product weight
                Product product = orderLine.getProduct();
                int productWeight = product.getWeight();
                if (minProductWeight <= productWeight && productWeight <= maxProductWeight) {
                    products.add(product);
                }
            }
        }
    }
}
```

- A lot of code
- Most parts are the same
- Not much differs!!!

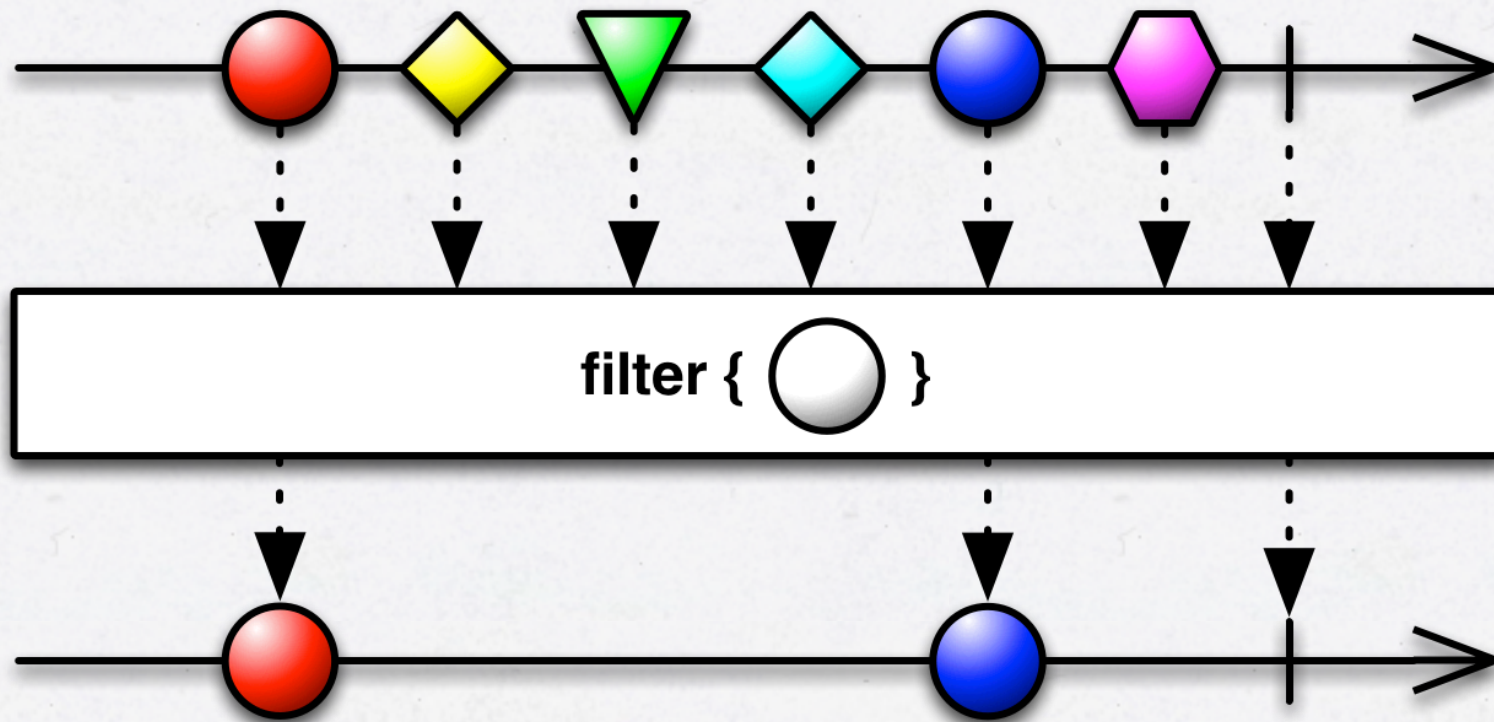
START WITH A IMPERATIVE SOLUTION, #2

```
        }  
        }  
    }  
  
    // Remove any duplicates from the list of selected products  
    products = new ArrayList<>(new HashSet<>(products));  
  
    // Sort on product Id  
    Collections.sort(products, new Comparator<Product>() {  
        @Override  
        public int compare(Product p1, Product p2) {  
            return (p1.getId() < p2.getId()) ? -1 : 1;  
        }  
    });  
  
    return products;  
}
```

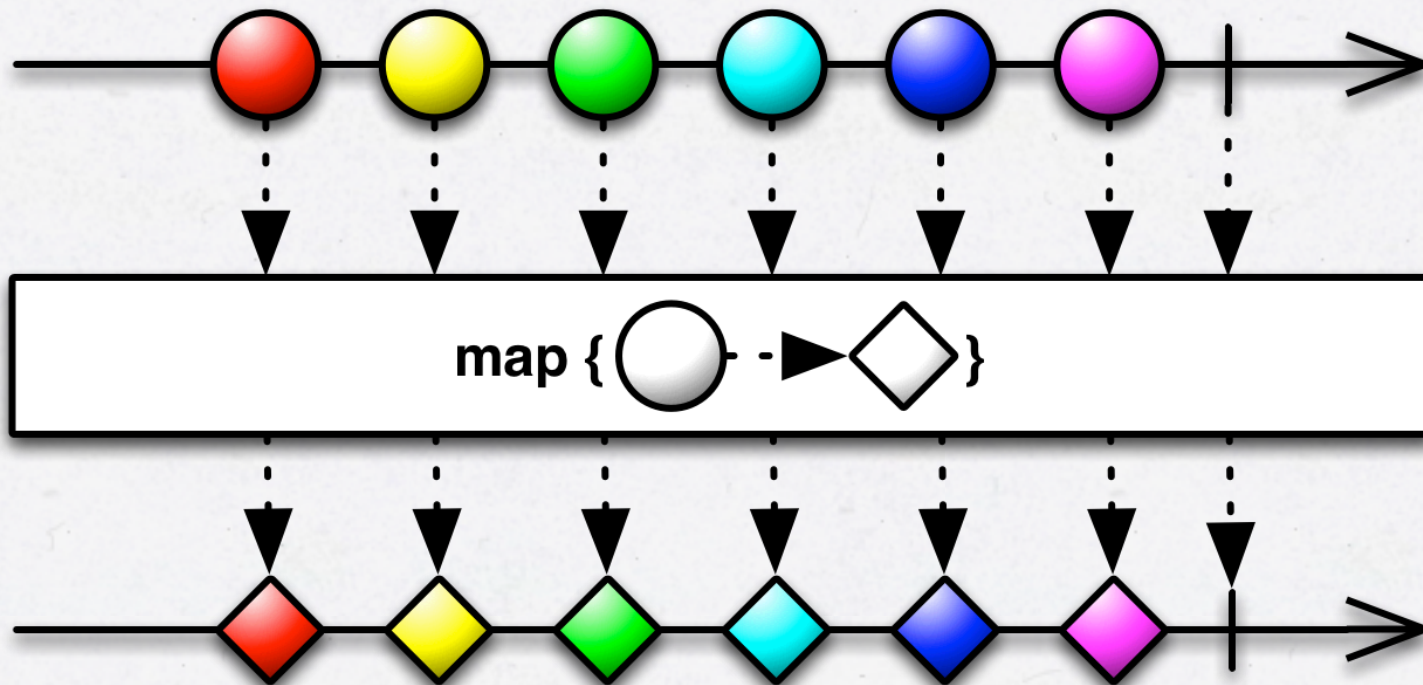
INTRODUCTION TO FUNCTIONAL PROGRAMMING

- Can we simplify this using functional programming?
- Let's try with plain Java 8
 - Express Functions using Lambda Expressions
 - Declare the data processing using Java 8 Stream API
- Lambda's we know about already,
but what about Java 8 Stream API???

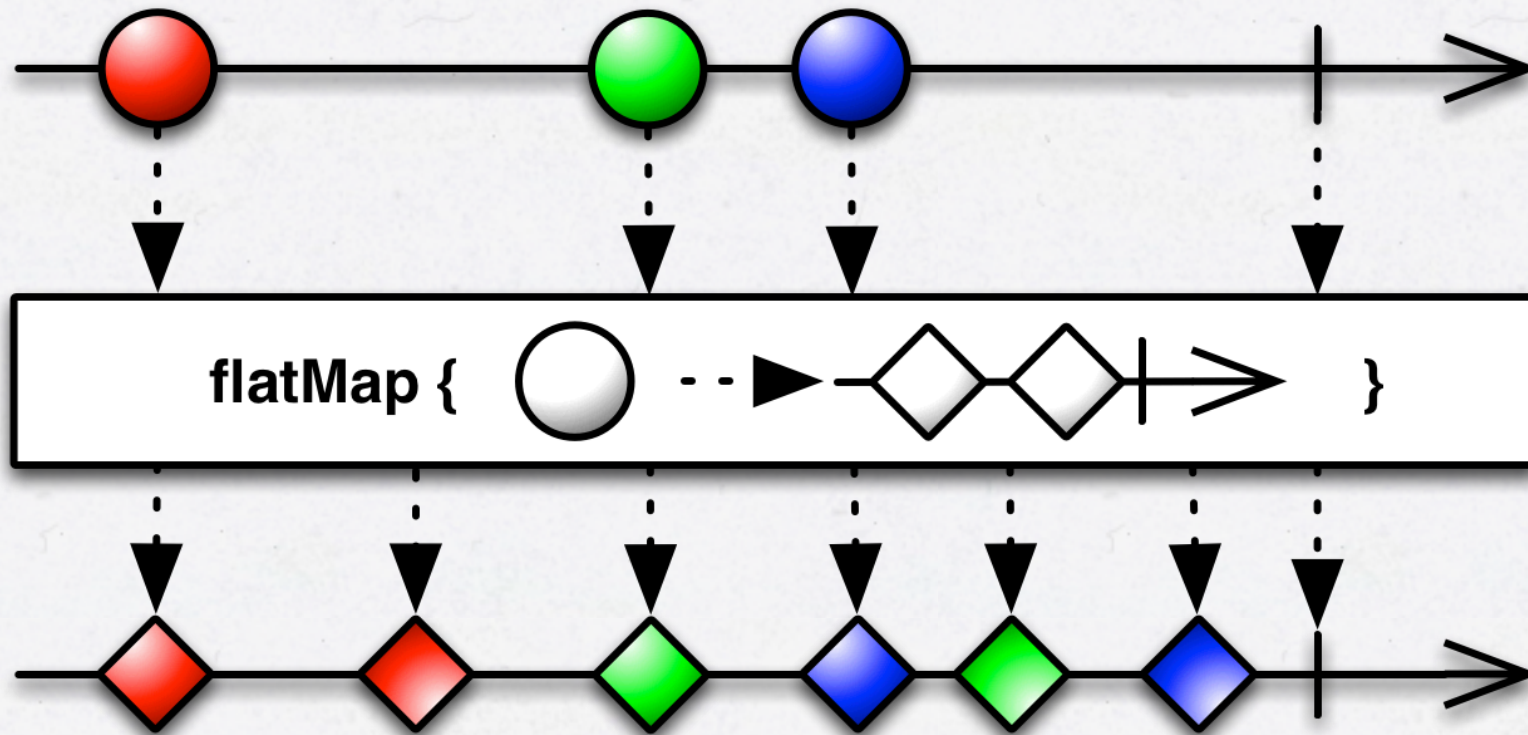
STREAM FILTER



STREAM MAP



STREAM FLATMAP



ENTERING THE FUNCTIONAL PROGRAMMING WORLD...

- Example #1 using Java 8 Streams

```
public List<Product> getProductsByDateAndCategoryOrderByWeight(
    LocalDate minDate, LocalDate maxDate, String category) {

    return getOrders()
        .stream()
        .filter(o -> o.getOrderDate().isAfter(minDate) && o.getOrderDate().isBefore(maxDate))
        .flatMap(o -> o.getOrderLines().stream())
        .map(ol -> ol.getProduct())
        .filter(p -> p.getCategory().equals(category))
        .distinct()
        .sorted((p1, p2) -> (p1.getWeight() < p2.getWeight()) ? -1 : 1)
        .collect(Collectors.toList());
}
```

ENTERING THE FUNCTIONAL PROGRAMMING WORLD...

- ...and example #2...

```
public List<Product> getProductsByOrderValueAndWeightOrderByProductId(  
    int minOrderValue, int maxOrderValue, int minProductWeight, int maxProductWeight) {  
  
    return getOrders().stream()  
        .filter (o -> minOrderValue <= o.getOrderValue() && o.getOrderValue() <= maxOrderValue)  
        .flatMap (o -> o.getOrderLines().stream())  
        .map      (ol -> ol.getProduct())  
        .filter  (p -> minProductWeight <= p.getWeight() && p.getWeight() <= maxProductWeight)  
        .distinct()  
        .sorted ((p1, p2) -> (p1.getId() < p2.getId()) ? -1 : 1))  
        .collect (Collectors.toList());  
}
```

- Much better!
- But still repetitions
- Violates the DRY principle

LET'S TAKE ONE MORE STEP INTO THE FUNCTIONAL WORLD...

- Separate the knowledge of the object model to a separate function
 - That takes three functions as arguments...

```
private List<Product> getProducts(  
    Predicate <Order>    orderFilter,  
    Predicate <Product> productFilter,  
    Comparator<Product> productComparator) {  
  
    return getOrders().stream()  
        .filter (orderFilter)  
        .flatMap (o -> o.getOrderLines().stream())  
        .map     (ol -> ol.getProduct())  
        .filter (productFilter)  
        .distinct()  
        .sorted (productComparator)  
        .collect (Collectors.toList());  
}
```


FUNCTIONAL PROGRAMMING TAKE 2

- Example #1

```
public List<Product> getProductsByDateAndCategoryOrderByWeight(
    LocalDate minDate, LocalDate maxDate, String category) {

    return getProducts(
        o      -> o.getOrderDate().isAfter(minDate) && o.getOrderDate().isBefore(maxDate),
        p      -> p.getCategory().equals(category),
        (p1, p2) -> (p1.getWeight() < p2.getWeight()) ? -1 : 1);
}
```

- Example #2

```
public List<Product> getProductsByOrderValueAndWeightOrderByProductId(
    int minOrderValue, int maxOrderValue, int minProductWeight, int maxProductWeight) {

    return getProducts(
        o      -> minOrderValue <= o.getOrderValue() && o.getOrderValue() <= maxOrderValue,
        p      -> minProductWeight <= p.getWeight() && p.getWeight() <= maxProductWeight,
        (p1, p2) -> ((p1.getId() < p2.getId()) ? -1 : 1));
}
```

INTRODUCTION TO FUNCTIONAL PROGRAMMING

- Also see

<http://callistaenterprise.se/blogg/teknik/2014/12/29/trying-out-functional-programming-in-java8/>

SUMMARY

- Start use Java 8 now!
- Look for areas where you can benefit from the new features
 - Start at small scale and widen the usage once proven within your organization...
- Verify that your 3PP's support Java 8