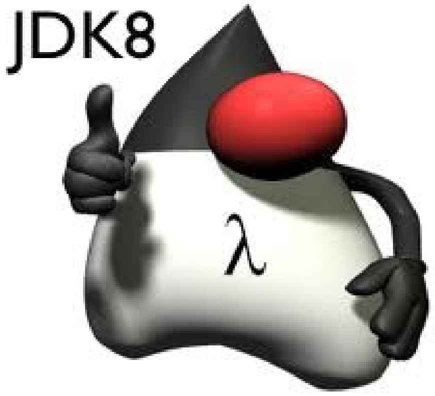
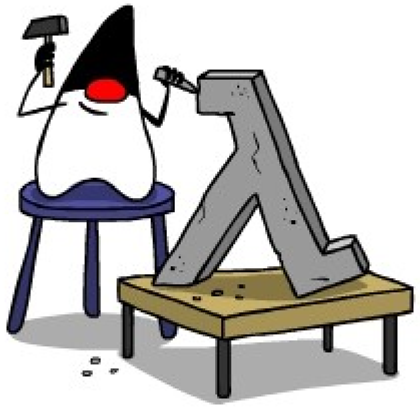
Lambdas And Streams in JDK 1.8





Presented By

Subhash Koganti

Agenda

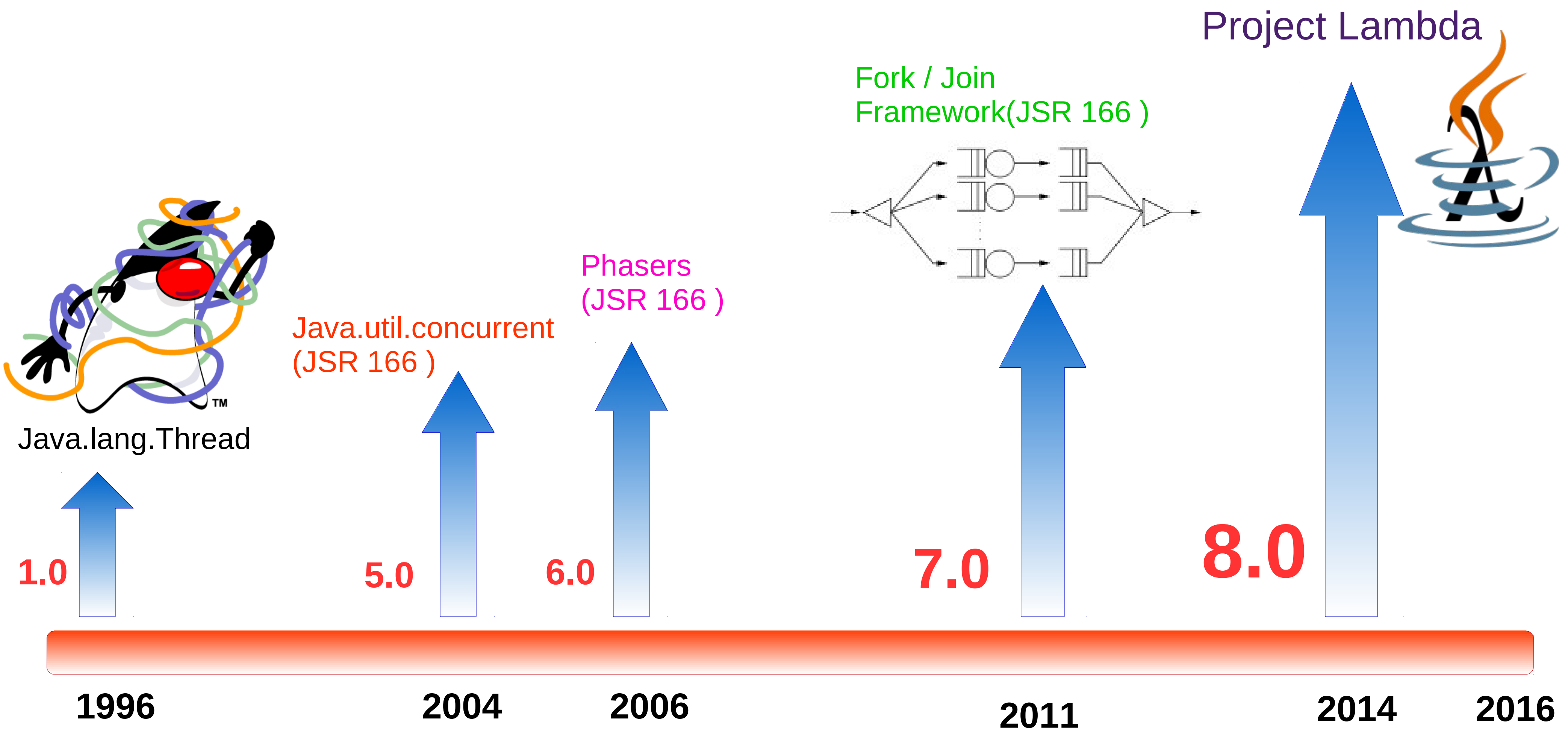
Lambdas

- Why Lambdas are added in Java ? Why now ?
- Lambda Expressions Syntax and Sample Code .
- Functional Interfaces.
- Summary for Lambdas.
- *Hands On DEMO*

Streams API

- Functional Programming Vs Imperative Programming
- What is a stream ?
- What are elements of a Stream ?
- What are some of the sources of Streams ?
- Examples of intermediate and terminal operations.
- Summary for Streams .

Why are Lambdas added in Java ?



Time line for Concurrency In Java.

Sample Code to find Highest RQI

```
List<Loan> loans = createLoansList();
double highestRQI = 0.0;

for( Loan l : loans ){

   if( l.getLoanYear() == 2016 ){

      if(l.getRQI() > highestRQI){
         highestRQI = l.getRQI();
      }
   }
}
```

Hidden Problems with the Code

```
List<Loan> loans =createLoansList();
double highestRQI = 0.0;

for( Loan l : loans ){

    if( l.getLoanYear() == 2016 ){

        if(l.getRQI() > highestRQI){
            highestRQI = l.getRQI();
        }
    }
}
```

- Iteration is in control of programmer
- Nature of the logic is basically serial.
- Not Thread Safe

More Functional looking Code

```
double highestRQI = loans
        .filter( new Predicate<Loan>(){
             public boolean test(Loan l){
                 return (l.getLoanYear()== 2016);
             }
        })
        .map ( new Mapper<Loan,Double>(){
             public Double extract(Loan l){
                 return l.getRQI();
             }
        })
        .max();
```

Advantages of Functional Code

```
double highestRQI = loans
 .filter(new Predicate<Loan>(){
       public boolean test(Loan l){
       return (l.getLoanYear()== 2016);
       }
 })
 .map ( new Mapper<Loan,Double>(){
       public Double extract(Loan l){
           return l.getRQI();
       }
 })
 .max();
```

- Iteration is in library's control
- Nature of the logic can be parallel
- Thread Safe

BUT

- UGLY Looking 👎

Code using Lambda Expressions

```
List<Loan> loans = createLoansList();
double highestRQI = loans
     . filter (Loan l → l.getLoanYear()== 2016)
     . map( Loan l → l.getRQI() )
     . max();
```

- Iteration is in library's control ✓
- Nature of the logic can be parallel ✓
- Thread Safe ✓
- ~~UGLY Looking~~ →(Much More Readable and Friendly Looking) ✓

Lambda Expressions- Syntax & Examples

(parameters) → { body }

Examples

- `( ) → System.out.println(“ UG Code Cafe “ );`
- `X → X +10`
- `( int X, int Y) → { return X+Y }`
- `(String X, String Y) → x.length()-y.length()`
-
```
(String X, String Y ) → {
        ListA.add(X);
        ListB.add(Y);
        return listB.size();
   }
```

Functional Interfaces

Definition

Its an interface that has one and only one abstract method.

- A Lambda Expression is an anonymous function and its not associated with any class.
- What is its type ??
- A Lamda can be used wherever the type is a Functional Interface
  - One and Only one Abstract method.
  - Lambda expression provides the implementation of that one abstract method. Hence its easy to map the type of the method to the type of the lambda expression.

Lambdas - Summary

- Lambdas are the anonymous methods that provide implementation to the functional interfaces
- They are more user friendly and better looking than anonymous classes.
- They allow programmers to pass behavior to the methods rather than values or references.

Lambda Expressions

Functional VS Imperative Programming

Imperative Programming

- Every value is associated with a variable name. That can be changed
- Order Of Execution matters.
- Repetition is controlled by programmer.

Functional Programming

- Every value is associated only once and will never change.
- Order Of Execution is not defined.
- Repetition is controlled by library through recursion.

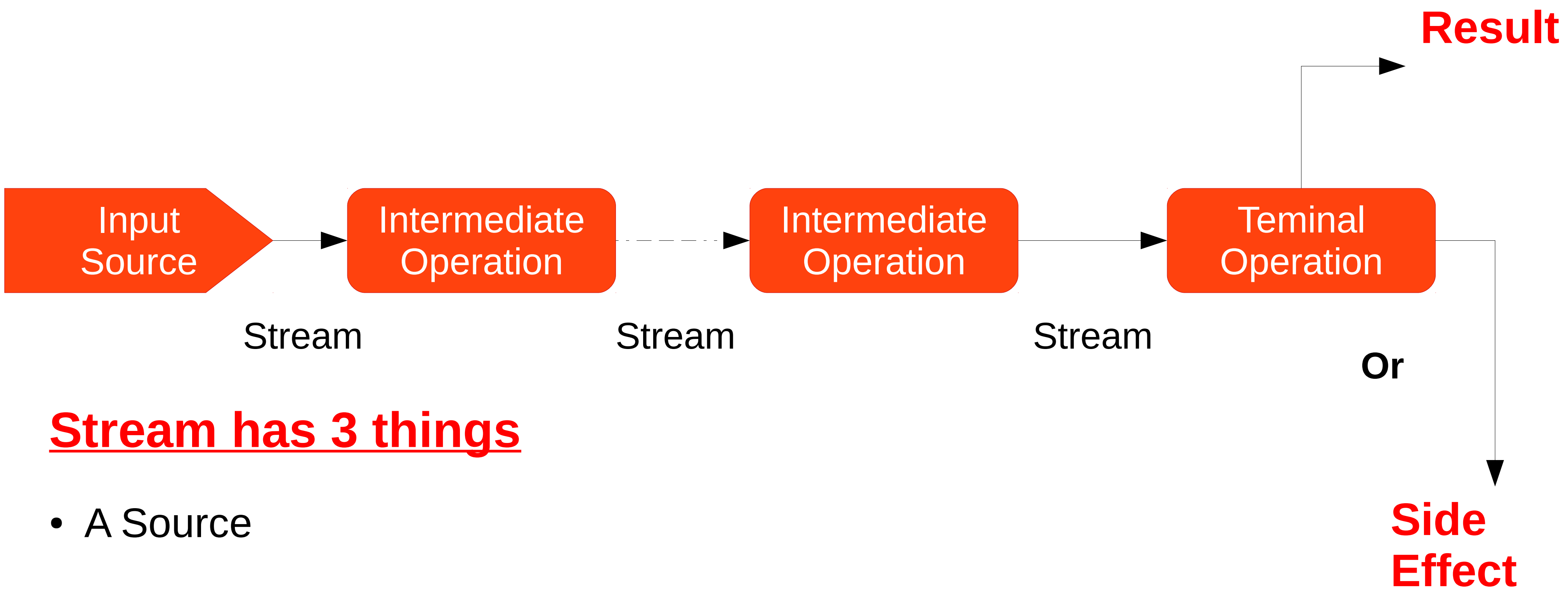
Streams API – What is a stream ??

Definition

Its an abstraction for specifying aggregate computations on a collection of elements.

- Is it a Data Structure ???? – NO
- Can it be infinite ??? - YES
- Gives java library opportunities for optimization through parallelism .

Elements of Stream- Pipeline Overview



Stream has 3 things

- A Source
- Zero or more Intermediate Operations
- One Terminal Operation
  - Produces a result or a Side Effect.

Streams – Example Code

Stream Source

```
int avgGPA_Class2016 = students.stream()
                    . filter(s → s.getGradYear() == 2016)
                    . mapToInt(s → s.getGPA())
                    . average();
```

Intermediate Operations

Terminal Operation

Stream Sources

There are many stream sources available in java 8 . Some Examples are as below.

Collection Interface

- **stream()**
- **parallelStream()**

Arrays class

- **stream()**

Files class

- **find(Path , BiPredicate, FileVisitOption)**
- **list(Path)**
- **lines (Path)**
- **Walk (Path, FileVisitOption)**

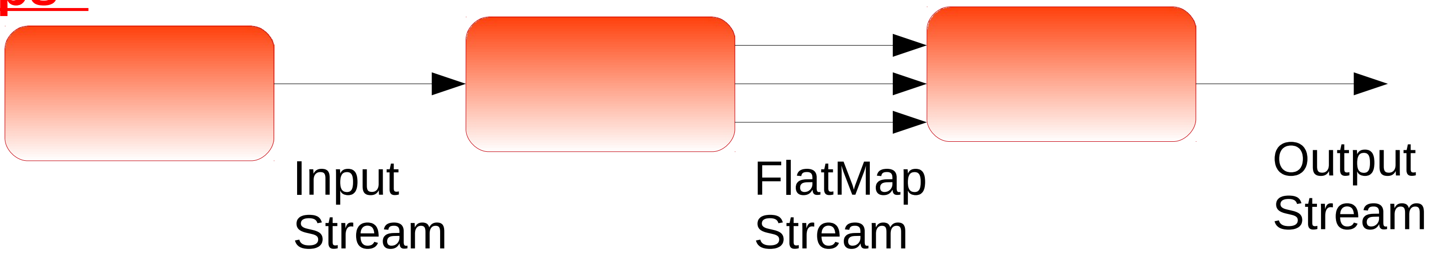
Random class

- **Three flavours of Random Class**
  - **Random**
  - **ThreadLocalRandom**
  - **SplittableRandom**
- **Ints() , doubles() , longs()**
- **Four versions of each**
  - **Finite or Infinite.**
  - **With or with out seed.**

Stream Intermediate Operations- Examples

Filtering & Mapping – distinct() , filter() , map , mapToInt, mapToDouble, mapToLong

Flat Maps



```
List<String> output = bufferedReader. lines()
    .flatMap( line → Stream.of(line.split( regEx) )
    .filter( word → word.length() >0 )
    .collect ( Collectors.toList());
```

Size Restrictions on a Stream – skip() , limit()

```
List<String> output = bufferedReader. lines()
    .skip(2).limit(2).collect( Collectors.toList());
```

Stream Terminal Operations- Examples

MatchingElements

findFirst(p), findAny(p), allMatch(p), anyMatch(p), noneMatch(p)

Collect Results -

Collect (Collector c), toArray()

Numerical Results

Count() , max (Comparator) , min (Comparator c) , average() , sum ()

Iteration

forEach(Consumer c) , forEachOrdered (Consumer c)

Streams - Summary

- Stream should be looked at as a pipeline of aggregate operations on a collection of elements.

- There are no explicit loops, which makes it easy for the library code to make it parallel.

Streams API

Lambdas & Streams in JDK 8