Embrace Java8

Functional Programming with Eclipse

Sebastian Zarnekow
itemis
Why functional?
Demo
Imperative Programming is about \textit{How to do it}\textsuperscript{1}
Functional Programming focusses on **What** to do
Java Meets Immutability
equals()  
hashCode()  
toString()  
<ctor>()  
final fields  
clone()  
copy ctor
Code written in Haskell is guaranteed to have no side effects.

...because no one will ever run it?
Pureness
No Side Effects
Referential Transparency
Caching & Memoization
Lazy Evaluation
and
Composition
Use Small Atomic Pieces
Assemble Rich Behavior
cf. Pipes & Filters
Behavior Parameterization
Groceries:
10 Eggs
Milk
Bread
Butter
Housework

hoover yr room

go with the dog
Housework

hoover yr room

Pass Behavior / Instructions Around

Code is Data

cf. Strategy Pattern
Separation of Concerns
public List<String> getErrors(String fileName) throws IOException {
    return errors;
}

public List<String> getErrors(String fileName) throws Exception {
    List<String> errors = new ArrayList<>();
    try {
        BufferedReader reader = new BufferedReader(new FileReader(fileName));
        int line = reader.readLine();
        while (line != null) {
            lineCounter++;
            if (line.startsWith("ERROR")) {
                errors.add(line);
            } else if (line.startsWith("ERROR") | line.startsWith("ERROR")) {
                errors.add(line);
            } else if (line.startsWith("ERROR") | line.startsWith("ERROR")) {
                errors.add(line);
            }
        }
    } catch (IOException e) {
        return errors;
    }
    return errors;
}
Lambdas and Streams
History Lesson

(Credits to Mario Fusco, Codemotion)

- 2006 – “We’ll never have lambdas in Java” (James Gosling)
- 2007 – 3 Different Proposals for Lambdas
- 2008 – “We’ll never have lambdas in Java” (Mark Reinhold, Devoxx)
- 2009 – Project Lambda is Alive (JSR 335) (Mark Reinhold, Devoxx)
\[ \lambda \text{ Syntax} \]

\[
\begin{align*}
  s & \rightarrow s.\text{length}() \\
  (\text{int } x, \text{ int } y) & \rightarrow x+y \\
  () & \rightarrow 42
\end{align*}
\]
\( \lambda \) Syntax

\[ (x, y, z) \rightarrow \{
    \text{if} (y == z) \text{ return } x;
    \text{else} \{
        \text{int result } = y;
        \text{for} (\text{int } i = 1; i < z; i++)
            \text{result } *= i;
        \text{return result;}
    \}
\} \]
Method References

String::length
System::getProperty
super::toString
ArrayList::new
double[][::new
"abc"::length
System.out::println
Method References

Poly Expressions

ArrayList::new
  new ArrayList<>(5)
  new ArrayList<>(myCollection)
  new ArrayList<>()
Interface Evolution

- Implementations in Interfaces
- Default Methods
- Static Methods
- Only Behavior is Inherited
- Backwards Compatible
API Evolution

List.replaceAll(UnaryOperator)
List.sort(Comparator)

Collection.removeIf(Predicate)

Iterable.forEach(Consumer)
Streams

Streams [...] are [...] concerned with declaratively describing their source and the computational operations which will be performed in aggregate on that source

```java
myListOfStrings.stream()
    .filter(s -> s != null)
    .mapToInt(String::length)
    .sum();
```
Streams

Stream<T>
IntStream
LongStream
DoubleStream
Streams (2)

Intermediate Operations
Terminal Operations
API Evolution (2)

Collection.stream()
Collection.parallelStream()
Exercises @ GitHub

github.com/szarnekeow/Java8Tutorial