
Busy Java Developer's Guide to Functional Programming

Ted Neward

Neward & Associates

<http://www.tedneward.com>

- Who is this guy?
 - Principal Consultant: architect, mentor, free agent coach
 - BEA Technical Director, Microsoft MVP Architect
 - JSR 175, 250, 277 EG member
 - Founding Editor-in-Chief: TheServerSide.NET
 - Author
 - *Professional F# (Forthcoming)*
 - *Effective Enterprise Java (Addison-Wesley, 2004)*
 - *Server-Based Java Programming (Manning, 2000)*
 - *C# in a Nutshell (OReilly, 2003)*
 - *SCLI Essentials (w/Stutz, Shilling; OReilly, 2003)*
 - Papers at <http://www.tedneward.com>
 - Weblog at <http://blogs.tedneward.com>

- What is functional programming (concepts)?
 - Why do I care?
- How can one program functionally in Java?
- What “close-to-Java” tools are there?

- Functional languages
 - functional as in mathematics' notion of function
 - for every x , there is a corresponding value y**
 - this implies no side effects**
 - not imperative statements, but expressions
 - " $x = x+1$ " is not increment, ... it's *impossible***
 - this implies expressions can be substituted**
 - ... or executed independently (parallelism)**
 - spectrum of "functional-ness", known as *purity*
 - "pure" functional languages allow for no side effects**
 - "impure" functional languages allow for side effects**

Any
effect

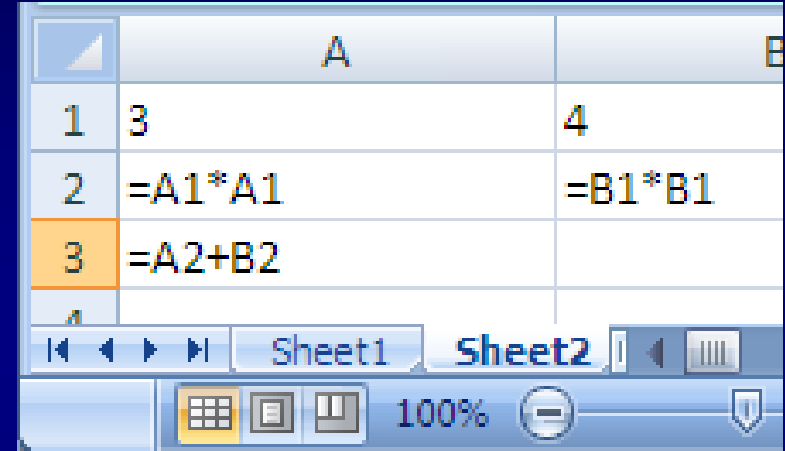
Spectrum

Pure
(no effects)

C, C++, Java, C#, VB

Excel, Haskell

```
X := In1
X := X * X
X := X + In2 * In2
```



The screenshot shows an Excel spreadsheet with the following data:

	A	B
1	3	4
2	=A1*A1	=B1*B1
3	=A2+B2	

The spreadsheet interface includes a status bar at the bottom showing 'Sheet1', 'Sheet2', and a zoom level of 100%.

Commands, control flow

Expressions, data flow

Do this, then do that
"X" is the name of a cell
that has different
values at different
times

No notion of sequence
"A2" is the name of a
(single) value

Imperative

C, C++, Java, C#, VB

→
X := In1
X := X * X
X := X + In2 * In2

In1	3
In2	4
X	

Commands, control flow

Do this, then do that
"X" is the name of a cell
that has different
values at different
times

Imperative

C, C++, Java, C#, VB



```
X := In1  
X := X*X  
X := X + In2*In2
```

In1 3

In2 4


X 3

Commands, control flow

Do this, then do that
"X" is the name of a cell
that has different
values at different
times

Imperative

C, C++, Java, C#, VB



```
X := In1  
X := X * X  
X := X + In2 * In2
```

In1 3

In2 4


X 9

Commands, control flow

Do this, then do that
"X" is the name of a cell
that has different
values at different
times

Imperative

C, C++, Java, C#, VB



```
X := In1  
X := X * X  
X := X + In2 * In2
```

In1 3

In2 4

X 25

Commands, control flow

Do this, then do that
"X" is the name of a cell
that has different
values at different
times

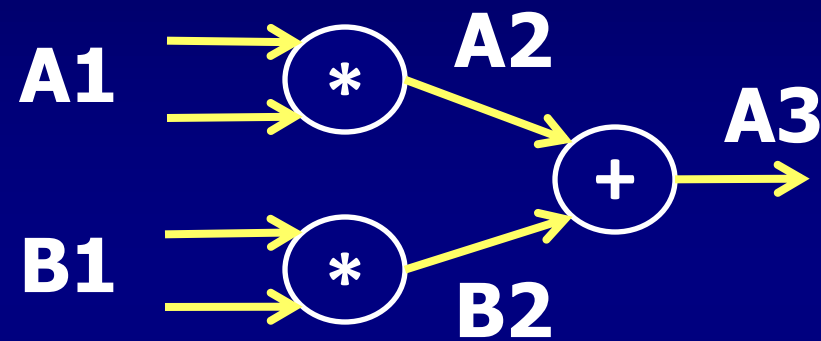
Functional

Excel, Haskell, F#

	A	B
1	3	4
2	=A1*A1	=B1*B1
3	=A2+B2	

Expressions, data flow

No notion of sequence
"A2" is the name of a
(single) value



$A2 = A1 * A1$
 $B2 = B1 * B1$
 $A3 = A2 + B2$

- Some basic functional concepts
 - strongly-typed, type-inferenced
 - immutable values
 - functions as first-class values
 - expressions-not-statements
 - tuples, lists
 - recursion
 - pattern-matching
 - currying, partial-application of functions

- Strongly-typed
 - Java already has this... to a point
 - Generics could/should be all the way through the JVM
- Type-inferenced
 - Java fails on this point—requires explicit declarations
 - Not a deal-breaking concern, just more verbosity
- Expressions-not-statements
 - Java fails on this point—statements core to the language
 - We could (maybe) support this by doing everything declaratively (e.g. generics-based expressions), but... bleah

Concepts

- Recursion
 - Java supports recursion, obviously
- Immutable values
 - Java supports immutability, but only with help
 - in other words, you have to train your fingers to type “final”**

- Functions as first-class values
 - Java fails here, *but* libraries can help some
 - Essentially, we will make heavy use of anonymous inner-class implementations to mimic/fake standalone functions; this is commonly known as *functors***
 - Not perfect, but manageable... maybe...
 - (Note: other languages make this point much easier, by hiding the ugly details behind their syntax)
 - BGGA-javac, Groovy, Scala, Clojure**

- Tuples, lists, option
 - Lists can be List<E>, but List<E> is missing some key ideas
 - Namely, all the “functional” operations**
 - Tuples are strongly-typed "bundles" of public data
 - Tuple2/Pair, Tuple3/Triplet, Tuple4/Quad, ...**
 - Option<E> is a single-membered collection, with two derived classes: Some<E>, and None
 - Some<E> represents a value**
 - None<E> represents no value**
 - Being an Iterable<E> means we can operate safely**
 - Most “functional” operations can be Collections-style algorithms on a support class

- Generators

- Iterable<E> knows how to produce Iterator<E>s
 - any Iterable<E> can participate in the enhanced "for" loop**
- But Iterator<E>s can *produce* values, not just report them
 - these are sometimes called *generators***
- Iterator<E>s could even never terminate!
 - so long as we have appropriate methods to handle them**
- this makes Iterator<E>s akin to *sequences*
 - this opens up a new way of thinking/processing**

- Sequences

- lots of things can be seen as sequences

- characters in a string**

- fields in a record**

- records in a database**

- files in a directory**

- algorithmic calculations (factorial, fibonacci, ...)**

- lines in a file**

- sequences and Collections have a deep relationship

- more on this later...**

- Pattern-matching
 - pattern-matching feels like a next-generation switch/case
 - patterns can be a variety of types
 - in Java, pattern-matching is not supported

- Partial application of functions
 - returning a function that is defined by taking another function and filling in some (not all) of its parameters
 - in Java, this means having to define a new method (as a subclass of a differently-defined functor interface) that manually passes in the filled-in parameters
 - in other words, not precisely doing us any favors here**

Implementation

- Most “functional” implementations in Java have to be done as a library of functors
 - functor: object behaving as a function
 - as a consequence, most “functional” implementations miss out on much of the functional language’s goodness
 - no type inference, no pattern-matching**
 - not impossible to program functionally... just hard
 - imagine programming to the JVM in C**

Implementation

- Some “functional Java” implementations
 - Apache Commons Functor library
<http://commons.apache.org/sandbox/functor>
 - FunctionalJava
<http://www.functionaljava.org>
 - BGGA compiler
<http://javac.info>
 - Mango
<http://www.jezuk.co.uk/cgi-bin/view/mango>
 - Generic Algorithms for Java
<http://jga.sourceforge.net/>

Summary

- Functional programming is a powerful approach
 - particularly when married against Java Collections
 - particularly when married against immutable values
- Consider a functional JVM language
 - Scala or Clojure are the front-runners
 - Jaskell is Haskell-on-JVM

Questions

?