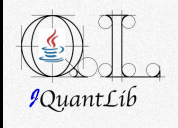


1.

# JQuantLib

A framework for Quantitative Finance written in Java



---

---

---

---

---

---

---

---

2.

## Quantitative Finance

What is Quantitative Finance?

- Valuation of financial instruments
- Involves complex mathematics and statistics

How it is used?

- Model trading strategies
- Determine optimal investment portfolios
- Testing of new models and strategies
- Risk valuation and management

---

---

---

---

---

---

---

---

3.

## Background

JQuantLib is based on QuantLib (implemented in C++)

### QuantLib

- Started in 2000
- ~35 developers
- MSVC++ / GCC
- Production quality v0.9.7 (November 18th, 2008)
- Near 2m lines of code
- Ported several languages (C# (QLNet), Python, Perl, Ruby...)

---

---

---

---

---

---

---

---

4.

## The Need of JQuantLib

QuantLib misses integration with Java

### Evaluation of QuantLib showed

- SWIG wrappers: inconvenient, incomplete, wrong
- JNI: inconvenient, complex, counterproductive
- SWIG/JNI: Difficult customization / extensibility

### Alternatives

- Integration via J2EE and CORBA containers
- Translate QuantLib to Java : ~1300 classes

---

---

---

---

---

---

---

---

5.

## Objectives

### JQuantLib aims...

- Translate QuantLib, which is written in C++
- Offer syntax and semantics Java developers expect but keeping JQuantLib API as close as possible to QuantLib API
- Be exceptionally well coded, accurate and well documented
- Take advantage of features Java can offer

---

---

---

---

---

---

---

---

6.

## Challenges

### QuantLib (C++)

- Very complex object model (example: Monte Carlo)
- Abuse of templates and other idioms

### Other challenges

- Maximum accuracy
- Very strong type checking at compile time
- Relative bad performance of Java
- Latency due to Objects and GC

---

---

---

---

---

---

---

---

7.

## Current Status - Jan/2009

- Started September 2007
- Coding started January 2008
- ~10 active developer
- First release in June 2008
- 40% of classes translated (January 2009)

---

---

---

---

---

---

---

---

8.

## Features

### From QuantLib

- Day counters, calendars, IMM
- Term structures, yield structures
- Instruments: stocks, options, bonds, swaps, etc
- Methods: Black-Scholes, binomial, LMM, MonteCarlo, low-discrepancy numbers, etc

### JQuantLib specific (planned)

- OSGi support
- Support for parallelism (Parallel Colt, <http://piotr.wendykier.googlepages.com/parallelcolt>)
- Grid enabled

---

---

---

---

---

---

---

---

9.

## Architecture

- Build Environment
- Quality Assurance
- Documentation
- Accuracy
- Strong Type Checking
- Performance

---

---

---

---

---

---

---

---

10.

## Build Environment

- Linux
- Java6
- Eclipse
- KDE
- Umbrello
- SVN
- Maven
- Continuum
- Archiva

---

---

---

---

---

---

---

---

11.

## Quality Assurance

- JUnit4
- PMD
- FindBugs
- EclEmma
- Cobertura
- Mantis

---

---

---

---

---

---

---

---

12.

## Correctness

### Strong type checking

- Enumerations
- Generic types
- Annotation on Java types (JSR-308)

### Accuracy

- Floating point rounding errors

---

---

---

---

---

---

---

---

13.

## Annotations on Java types

```
double calc(double rate, double year) {
    return Math.exp(1+rate*year);
}
```

```
double rate = 0.45;
double year = 0.5;
```

```
double result1 = calc(rate, year);
double result2 = calc(year, rate); // Wrong, not recognized by the compiler
```

### Need for semantic checking

- Eradicate errors at compile time
- Test cases are not 100% guaranteed :(

```
C++
typedef double Rate;
typedef double Year;
double calc(Rate rate, Year year);
```

---

---

---

---

---

---

---

---

---

---

14.

## JSR-308

```
private Double calc(@Rate double rate, @Time double time) {
    return new Double(Mathc.exp(1+rate, time));
}
```

```
@Rate double rate = 0.45;
@Time double time = 0.5;
```

```
// This call will pass
Double result1 = calc(rate, time);
```

```
// This call can give us a compiler error
Double result2 = calc(time, rate);
```

- Available in JDK7 (target date for JDK7: Early 2010)
- Annotations wherever a type is accepted
- Annotation processor plugged into compilation phase

### Link

<http://groups.csail.mit.edu/pag/jsr308/>

---

---

---

---

---

---

---

---

---

---

15.

## Accuracy

```
System.out.println(0.1 + 0.1 + 0.1);
```

```
0.30000000000000004
```

### Requirements

- As accurate as QuantLib (C++)
- As lightweight as possible
- As fast as possible

### Solution

- Primitive types
- Calculate epsilon when needed

---

---

---

---

---

---

---

---

---

---

16.

## Documentation

### Requirements

1. Replace doxygen

### Solution

1. UMLGraph
2. LaTeXtaglet

### Links

<http://www.umlgraph.org/>

Tool for testing Latex formulas: Laeqed

---

---

---

---

---

---

---

---

---

---

17.

## Performance

- Critical requirement
- Needed for low latency
- Needed for scalability

### Topics

- Comparison with C++
- Numbers and Objects
- Collections
- Math Packages
- Parallelism
- JVM and gc
- Profiling

---

---

---

---

---

---

---

---

18.

## Comparison with C++

### Benchmarks in DDJ (Java 5)

- 32bit integer arithmetic: as fast as
- 64bit double arithmetic: as fast as
- sort algorithms: 50% slower
- list operations: 2x slower
- matrix operations: 2x to 3x slower
- nested loops: 2x slower
- trigonometric functions: deadly slow!

<http://www.jquantlib.org/index.php/DesignPerformance>

---

---

---

---

---

---

---

---

19.

## Collections

### Issues

- Not optimised for high performance systems
- Expansive object management and reference

### Alternatives

- Arrays of primitive types
- Optimised JCF implementation

**fastutil** `List list = DoubleArrayList(); // backed by an array of doubles  
list.add(1.0); // autoboxing :: list.add(new Double(1.0));  
(DoubleArrayList)list.add(2.0); // no autoboxing :)  
double d = (DoubleArrayList)list.getDouble(0); // no autoboxing`

<http://fastutil.dsi.unimi.it/>

---

---

---

---

---

---

---

---

20.

## Math Packages

### Colt was developed at CERN

- Stable and Reliable
- Optimised / High performance
- Production grade
- Vector and Matrix operations
- Linear Algebra
- Statistical methods
- Last release: v1.2, Sept/04



<http://acs.lbl.gov/~hoschek/colt/>

---

---

---

---

---

---

---

---

21.

## Parallelism

### Levels of parallelism

- JQuantLib is thread-safe
- Parallel Colt takes advantage of multiple CPUs
- Customized JVMs

### Parallel Colt

- The natural evolution for Colt
- Important factor for selecting Colt
- Still in development but no major issues
- Last release: v0.6.1, Dec. 2008

<http://piotr.wendykier.googlepages.com/parallelcolt>

---

---

---

---

---

---

---

---

22.

## JVM and gc



JVMs need to be optimised for specific hardware

### Some info about Azul appliances

- Getting rid of the JVM scaling issues
- Customised JVM
- Up to 54 cores per CPU
- Up to 16 processors, 860 cores, 768Gb
- Grid enabled: can scale even more
- Hardware assisted GC
- Hardware assisted Java locking
- Performance increase: order of hundred times

<http://www.azulsystems.com>

---

---

---

---

---

---

---

---

23.

## Profiling

- No memory profiling yet
- Performance tests are still incipient

---

---

---

---

---

---

---

---

24.

## Next releases

### 3rd release

- Date: 12th February 2009 / Eclipse Banking Day, London
- American Options with Finite Differences
- American Options with Integral Engine
- Asian Options
- Translation and tests of all 35 calendar classes, from 2004 to 2012

### 4th release - tbd

- Monte Carlo method
- Sobol (Quasi Monte Carlo, low-discrepancy numbers)
- Bonds

---

---

---

---

---

---

---

---

25.

## Future

### Pluggable OSGi bundles

- Purpose specific implementations
- Hot swap
- 24x7x365

### Marketplace

- Products and services show case
- Cooperation
- Forum, wiki
- Room for innovation

---

---

---

---

---

---

---

---

26.

Thanks :)

<http://www.jquantlib.org/>

Wer möchte mitarbeiten?

---

---

---

---

---

---

---

---