JavaOne 2010

Douglas Bullard
14 October, 2010

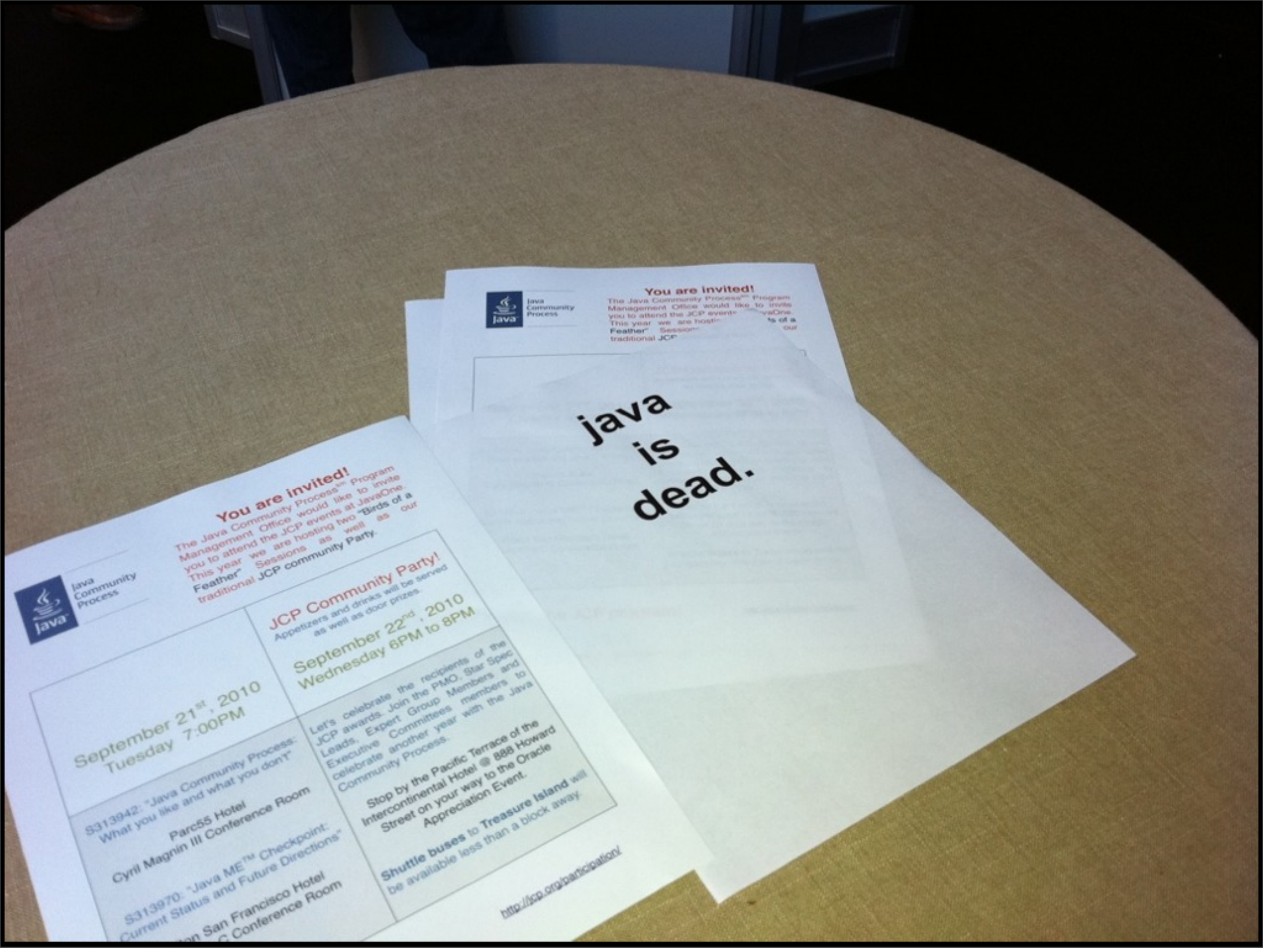
Coolest Presentation

- S314238 - JavaFX: Designer Developer Workflow
  - Designers create UIs in Photoshop and export them to JavaFX

Overall Themes


ORACLE
CSC
NetApp
CSC
NetApp


Google
Rolling Thunder
myuniportal.com
JAMES
GOSLING
IS GOD
AND I'M HIS
PROPHET


Java Community Process
You are invited!
The Java Community Process Program Management Office would like to invite you to attend the JCP events at JavaOne. This year we are hosting two "Birds of a Feather" Sessions as well as our traditional JCP community Party.
JCP Community Party!
Appetizers and drinks will be served as well as door prizes.
September 22nd, 2010
Wednesday 6PM to 8PM
September 21st, 2010
Tuesday 7:00PM
S313942: "Java Community Process: What you like and what you don't"
Parc55 Hotel
Cyril Magnin III Conference Room
S313970: "Java ME Checkpoint: Current Status and Future Directions"
San Francisco Hotel
Conference Room
Let's celebrate the recipients of the JCP awards. Join the PMO, Star Spec Leads, Expert Group Members and Executive Committees members to celebrate another year with the Java Community Process.
Stop by the Pacific Terrace of the Intercontinental Hotel @ 888 Howard Street on your way to the Oracle Appreciation Event.
Shuttle buses to Treasure Island will be available less than a block away.
http://jcp.org/participation/
java is dead.

Overall Themes

- Java isn't dead
- New versions of Java will be on a more regular cadence
  - JDK 7 will introduce lots of new features (2011)
  - JDK 8 will introduce even more (2012)

GlassFish

- 2 New GlassFish Releases in the next year
  - Committed Feature List for 2011: http://glassfish.dev.java.net/roadmap/

GlassFish

- GlassFish 3.1 - 2010
  - Centralized Administration/Clusters
  - High Availability/State Replication
  - Value added features, like Coherence Support
- GlassFish 3.2 - 2011
  - Improved Cluster/HA administration
  - Better integration w/ Oracle Identity Management
  - Virtualization support
  - Some Java EE 6 updates, some Java EE7 EA
- GlassFish 4
  - Common Server Platform - Shared best of breed with WebLogic Server
  - Java EE 7

JDK 7 & JDK 8

- JDK 7 will be out mid 2011
  - Project Coin - small changes to simplify everyday tasks
- JDK 8 will be out in 2012
  - Project Lambda
- Best of JRockit JIT will be incorporated into HotSpot

JDK 7

- Diamonds
  - Yesterday

```
Map map = new HashMap();
```

  - Today

```
Map<String, String> = new HashMap<String,
String>();
```

  - JDK 7 Diamonds

```
Map<String, String> = new HashMap<>();
```

JDK 7

- Value Classes
  - Today - POJO class

```
public class Person {
  private String firstName;
  private String lastName;

  public String getFirstName(){
    return firstName;
  }

  public String getLastName(){
    return firstName;
  }

  public String setFirstName(String text){
    firstName=text;
  }
  public String setLastName(String text){
    lastName=text;
```

JDK 7

- Value Classes
  - JDK 7 - POJO class

```
value class Person {
  Person property String firstName;
  Person property String lastName;
}
```

JDK 7

- Automatic Resource Management (ARM)
  - Today - copying a file

```
static void copy(String src, String dest) throws IOException {
  InputStream in = new FileInputStream(src);
  try {
    OutputStream out = new FileOutputStream(dest);
    try {
      byte[] buf = new byte[8 * 1024];
      int n;
      while ((n = in.read(buf)) >= 0) out.write(buf, 0, n);
    }
    finally {
      out.close();
    }
  }
  finally {
    in.close();
  }
}
```

JDK 7

- Automatic Resource Management (ARM)
  - JDK 7 - copying a file

```
static void copy(String src, String dest) throws IOException {
  try (InputStream in = new FileInputStream(src);
       OutputStream out = new FileOutputStream(dest)) {
    byte[] buf = new byte[8192];
    int n;
    while ((n = in.read(buf)) >= 0) out.write(buf, 0, n);
  }
}
```

  - If more than one close operation throws an exception, the exception itself has the suppressed exception within it.

JDK 7

- Strings in switch statements

```
String division = getProductDivision();
switch(division)
{
  case "FOOTWEAR":
    processFootwear(style);
    break;
  case "APPAREL":
  case "EQUIPMENT":
    processFootwearEquipment(style);
    break;
  default:
    processDefault(s);
    break;
}
```

JDK 7

- Multi-catch with Precise Rethrow
  - Today:

```
try{
  // some nasty code
} catch (SomeException e){
  // do some code here
} catch (SomeException2 e1){
  // do some code here
} catch (SomeException3 e2){
  // do some code here
} catch (SomeException4 e3){
  // do some code here
} catch (SomeException5 e4){
  // do some code here
}
```

JDK 7

- Multi-catch with Precise Rethrow
  - JDK 7:

```
try{
  // some nasty code
} catch (SomeException | SomeException2 |
 SomeException3 | SomeException4 |
 SomeException5 e){
  // do some code here
}
```

JDK 7

- Collections manipulations and declarations

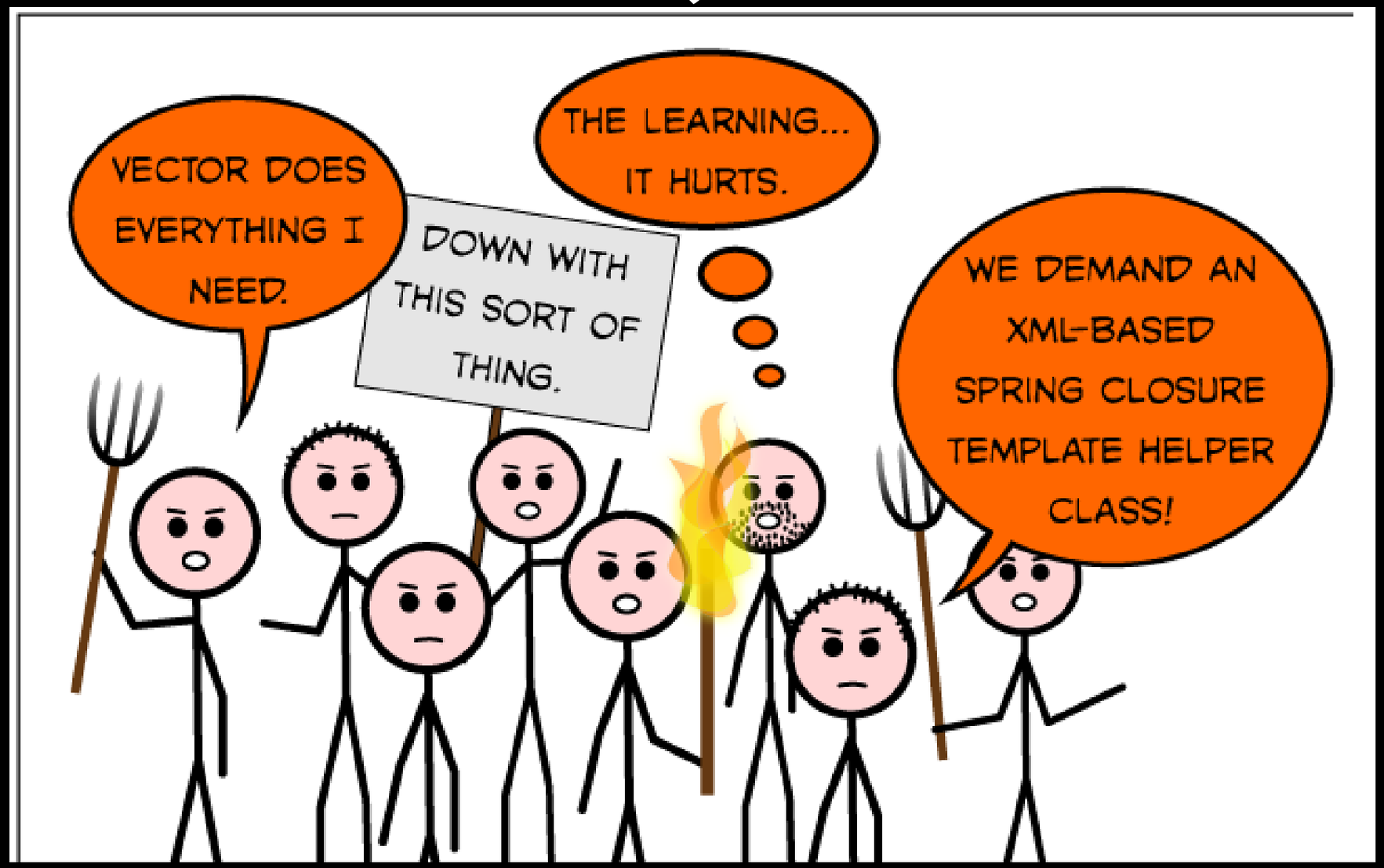
```
List<String> list = ...;
Map<String, String> map = ...;

String firstValue = list[0];
map["Test"] = firstValue;
String valueFromMap = map["Test"];
List<Integer> numbers = [ 1, 2, 4, 8, 16, 32, 64, 128 ];
Set<Integer> numbers = { 256, 512, 1024, 2048, 4096 };
Map<String, String> translations = {
  "Hi" : "Bonjour",
  "Goodbye" : "Au revoir",
  "Thanks" : "Merci" };
```

JDK 8 & Project Lambda

- Lambda expressions == closures

JDK 8 & Project Lambda



JDK 8 & Project Lambda

- Chips aren't getting any faster (think about it)
  - Speed increases will come through parallelism
  - Software must be written to parallelize gracefully
  - Without more language support for parallel idioms, people will instinctively reach for serial idioms

JDK 8 & Project Lambda

- The biggest serial idiom of all: the for loop

```
double highestScore = 0.0;

for (Student s : students) {
   if (s.gradYear == 2010) {
      if (s.score > highestScore) {
         highestScore = s.score;
      }
   }
}
```

- This code is *inherently serial*
  - Traversal logic is fixed (iterate serially from beginning to end)
  - Business logic is stateful (use of > and accumulator variable)

JDK 8 & Project Lambda

- The biggest serial idiom of all: the for loop

```
double highestScore = 0.0;

for (Student s : students) {
   if (s.gradYear == 2010) {
      if (s.score > highestScore) {
         highestScore = s.score;
      }
   }
}
```

- Existing collections impose external iteration
  - Client of collection determines mechanism of iteration
  - Implementation of accumulation is over-specified
  - Computation is achieved via side-effects

JDK 8 & Project Lambda

“The pain of anonymous inner classes makes us roll our eyes in the back of our heads every day”

JDK 8 & Project Lambda

- Lambda expressions
  - Lambda expression is introduced with #
  - Zero or more formal parameters
    - Like a method
  - Body may be an expression or statements
    - Unlike a method
    - If body is an expression, no need for ‘return’ or ‘;’

JDK 8 & Project Lambda

- Example: Find the highest score for all students in the collection who graduated in 2010

```
Collection<Student> students = …

double highestScore =
  students.filter(#{ s -> s.gradYear == 2010})
          .map(   #{ s -> s.score})
          .max();
```

- Since we're filtering on a collection of Students, s must be a Student
- You can give parameter types in case of ambiguity

JDK 8 & Project Lambda

- Ensure lambda expressions work easily with existing libraries
  - Java SE will include a “starter kit” of types such as
    - Predicate
    - Filter
    - Extractor
    - Mapper
    - Reducer

JDK 8 & Project Lambda

- Example: Sort a list of Persons by last name

```
class Person { String getLastName() {...} }

List<Person> people = …
Collections.sort(people, new Comparator<Person>() {
  public int compare(Person a, Person b) {
    return a.getLastName().compareTo(b.getLastName());
  }
}
```

- Nasty inner class…
- Worse if the key is a primitive!

JDK 8 & Project Lambda

- Example: Sort a list of Persons by last name

```
class Person { String getLastName() {...} }

List<Person> people = …
Collections.sortBy(people, #Person.getLastName);
```

JDK 8 & Project Lambda

- Extension methods: a measured step towards more flexible inheritance

```
public interface Set<T> extends Collection<T> {
  public int size();
  ...
  public extension T reduce(Reducer<T> r)
    default Collections.<T>setReducer;
}
```

- Allows library maintainers to effectively add methods after the fact by specifying a default implementation
- Less problematic than traits, mix-ins, full multiple inheritance

JDK 8 & Project Lambda

- Extension methods
  - An extension method is just an ordinary interface method
  - For a client:
    - Nothing new to learn – calling the extension method works as usual, and the default method is linked dynamically if needed
  - For an API implementer: – An implementation of an augmented interface may provide the method, or not
  - For an API designer:
    - Default method can only use public API of augmented interface
  - For a JVM implementer – Lots of work

JDK 8 & Project Lambda

- Example: Sort a list of Persons by last name

```
class Person { String getLastName() {...} }

List<Person> people = …
Collections.sort(people, Person.getLastName);
```

JEE Testing

- Doing any AJAX? GWT?
  - Use Google SpeedTracer to test your UI performance
- Embedded container for unit testing
  - Jetty
  - Arquillian
    - JBoss as default
    - Glassfish
    - Open EJB, Open Web
    - WebLogic (future)

JEE Testing

- Selenium for UI testing
- Splunk for log sifting
  - BIG HINT:
    - Use key-value paris in logs
    - Splunk finds these automatically

GWT

- GWT 2.0
  - Hosted mode plugin
    - Test in the browser you're running in
  - Code splitting
    - Allows JavaScript to be split up and loaded as needed
  - Layout panels
    - Pre-2.0, all JavaScript
    - Now CSS based
    - Big difference in performance

GWT

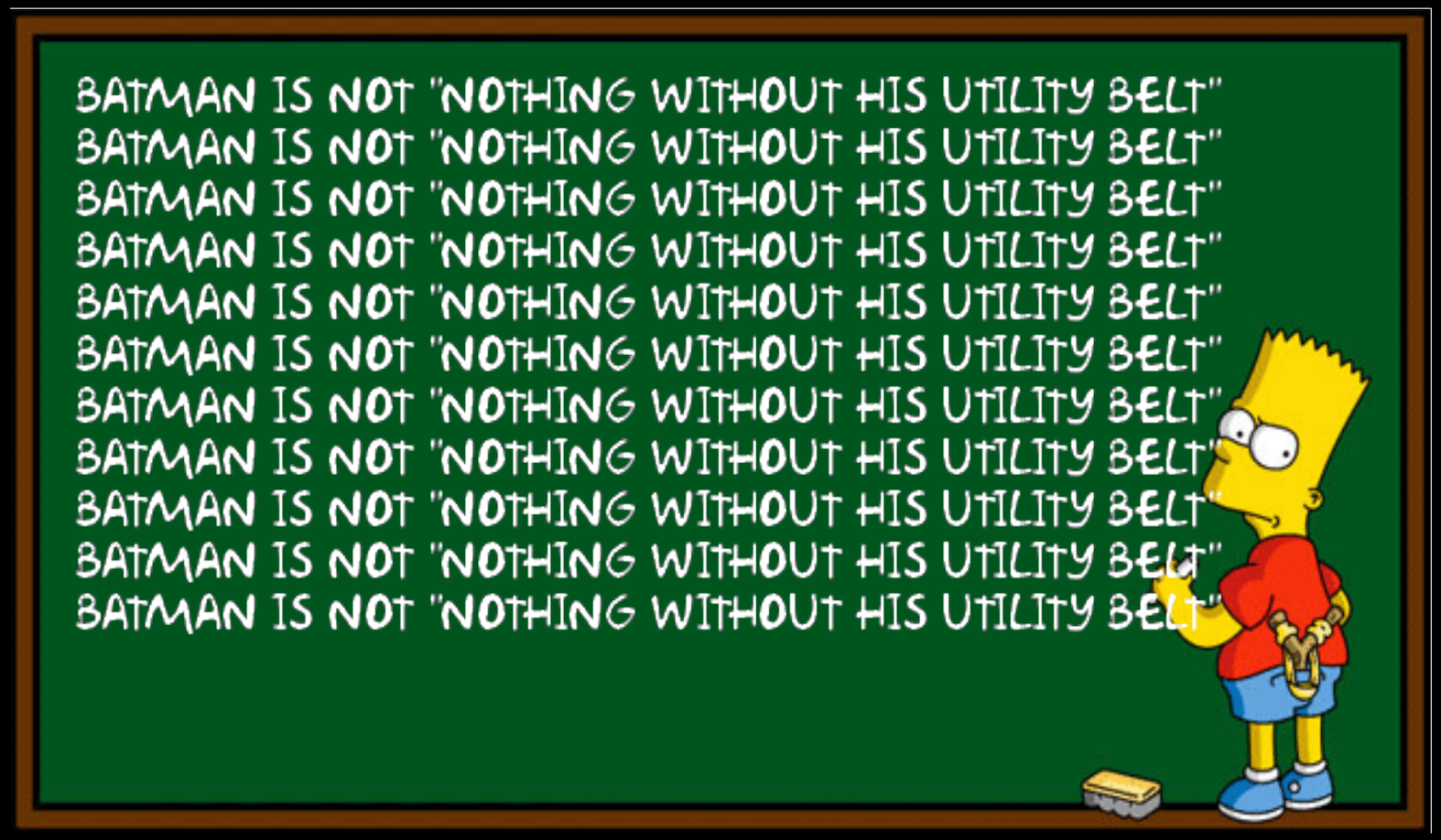
- GWT 2.0
  - Client Bundles
    - Used to be just images
    - Now has CSS
  - UI Binder
    - Splits UI layout from Java into XML
  - SpeedTracer
    - Works with any web page to evaluate page performance

GWT

- GWT 2.0
  - History change handlers
    - Hooks into “back”, “next” buttons on browser
    - Store arbitrary application states
    - Not “back/next page” - roll application forward or backward to desired state

Top 10 Production Issues

- Tools for profiling



Top 10 Production Issues

- Tools for profiling
  - What the JVM is doing - lightweight
    - dtrace, hprof, introscope, jconsole, visualvm, yourkit, azul zvision

Top 10 Production Issues

- Tools for profiling
  - What the JVM is doing - heavyweight
    - bci, jvmti, jvmdi/pi agents
    - logging (Splunk, anyone?)

Top 10 Production Issues

- Tools for profiling
  - What the OS is doing
    - dtrace, oprofile, vtune
  - What the network/disk is doing
    - ganglia, iostat, lsof, nagios, netstat

Top 10 Production Issues

- 10 - Instrumentation is not cheap
  - Production monitoring can be very expensive
  - Stage environment doesn't reproduce issues
  - Instrumented code changes cache profile
  - MBeans aren't cheap!
- Solutions
  - Pick the right tool for the problem
  - Asynchronous logging, jconsole

Top 10 Production Issues

- 9 - Leaks
  - Symptoms
    - App consumes all the memory
    - Heap trend is a ramping sawtooth
    - App slows, then throws OutOfMemory
  - Tools
    - yourkit, hprof, eclipse mat, jconsole, jhat, jps, visualvm, azul zvision
  - Causes
    - Allocated vs Live Objects, vm memory, Perm Gen
    - Finalizers, ClassLoaders, ThreadLocal

Top 10 Production Issues

- 8 - I/O: Serialization
  - Symptom
    - Multi-node scale-out does not scale linearly
    - Time in both CPU and I/O (serialization costs)
  - Tools
    - Cpu profiling, I/O profiling
  - Solution
    - All serialization libraries are not equal!
    - Pick a high performance serialization library or roll-your-own
    - Avro, kryo, protocol-buffers, thrift

Top 10 Production Issues

- 8 - I/O: Limits, Tuning
  - Symptom
    - Application hangs or remote call fails after awhile
    - "Too many open File Descriptors", "Cursors"
    - Inconsistent response times
  - Tools
    - nagios, pkg, rpm info, ulimit, yum
  - Solutions
    - Check for "new" OS patches, user & process limits, network & semaphore configurations
    - Close all I/O streams
    - Maybe you are I/O bound or locked

Top 10 Production Issues

- 8 - I/O: Sockets, Files, DB
  - Symptoms
    - Socket.create/close takes too long
    - JRMP timeouts, long JDBC calls
    - Running out of file descriptors, cursors, disk
  - Tools
    - dbms tools, du, iostat, gmon, lsof, netstat
  - Workaround
    - Check all O/S patches, sysctl flags, run ping/telnet test
    - Check & set SO_LINGER, TCP_LINGER2

Top 10 Production Issues

- 7 – Locks & `synchronized`
  - Symptoms
    - Adding users / threads / CPUs causes app slow down (less throughput, worse response)
    - High lock acquire times & contention
    - Race conditions, deadlock, I/O under lock
  - Tools
    - d-trace, lockstat, azul zvision
  - Solution
    - Use non-blocking collections
    - Striping locks, reducing hold times, no I/O

Top 10 Production Issues

- 6 – Endless Compilation
  - Symptom
    - “JIT gone wild”
  - Tools
    - -XX:+PrintCompilation, cpu profiler
    - Find endlessly-recompiling method
  - Workaround
    - Exclude using .hotspot_compiler file
  - Root cause: It's a JVM Bug! File a bug report!

Top 10 Production Issues

- 5 – Endless Exceptions
  - Symptom
    - Application spends time in j.l.T.fillInStackTrace()
  - Tools
    - Cpu profiler, azul zvision
    - Thread dumps (repeated kill -3, zvision)
    - Track caller/callee to find thrower
      - Not all exceptions appear in log files
  - Solution
    - Don't Throw, alternate return value (e.g. null)

Top 10 Production Issues

- 4 - Fragmentation
  - Symptom
    - Performance degrades over time
    - Inducing a full GC makes problem go away
    - Lots of free memory but in tiny fragments
  - Tools
    - GC logging flags, e.g. for CMS
      -XX:PrintFLSStatistics=2
      -XX:+PrintCMSInitiationStatistics

Top 10 Production Issues

- 3 – GC Tuning
  - Symptom
    - Constant 40-60% CPU utilization by GC
    - Scheduled reboots
    - Full time engineer working GC flags

Top 10 Production Issues

- 3 – GC Tuning
  - Oracle Weblogic GC Flags

```
-server
-Xloggc:gc.log
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-XX:MaxPermSize=128m
-XX:+UseParNewGC
-XX:+UseConcMarkSweepGC
-XX:MaxNewSize=64m
-XX:NewSize=64m
-Xms1536m
-Xmx1536m
-XX:SurvivorRatio=128
-XX:MaxTenuringThreshold=0
-XX:CMSInitiatingOccupancyFraction=60
-Dsun.rmi.dgc.server.gcInterval=0x7FFFFFFF FFFFFFE
-Dsun.rmi.dgc.client.gcInterval=0x7FFFFFFF FFFFFFE
```

Top 10 Production Issues

- 2 - Spikes
  - Symptoms
    - Rush hour traffic, tax day, Black Friday
    - Outages under spikes, power law of networks
  - Solution
    - Measure
      - Test with realistic load & realistic multi-node setup
      - Build redundancy & elasticity into infrastructure
      - Don’t ignore exceptions & retries under load

Top 10 Production Issues

- 1 – Versionitis - *When ears wage class wars with jars*
  - Symptom
    - Different nodes have different configurations, different stack components, versions
    - classpath has lib/*, -verbose:class
    - subtle hard to reproduce issues
  - Solution
    - Method. Version Control
    - Rigor

Too Big to Fail

Tips for Massive Enterprise Applications

- Tip #1 - How can I reduce heap requirements and improve performance?
- Turn on Compressed OOPS
  - Compressed OOPS replace 64-bit pointers with 32-bit indexes from the heap base.
  - Reduces heap size for long-term resident data (in-memory caches) - by ~30%.
  - Increases overall performance/throughput.
  - Simple to enable: -XX:+UseCompressedOOPS

Too Big to Fail

Tips for Massive Enterprise Applications

- Tip #2 : How can I get better scalability on multi-threaded applications that create many objects per thread?
- Turn on NUMA (Non-Uniform Memory Access) support
  - Changes object allocation algorithm.
  - Objects are allocated in memory local to the core on which the thread is executing.
  - Enabled by adding the java command line flag -XX:+UseNUMA.

Too Big to Fail

Tips for Massive Enterprise Applications

- Tip #5: How can I get insight into the runtime behavior of my application with minimal effort?
- Use hprof.
  - Will you be surprised by the results?
  - The simplicity of use and text file output format masks its power.
  - So familiar that people often forget about it.
  - Very simple to use with low overhead:
    - -Xrunhprof:cpu=samples

Too Big to Fail

Tips for Massive Enterprise Applications

- Tip #5: Use hprof
  - Case study : The JAR executable:
    - For years the jar executable had a serious performance problem but no one noticed it!
    - Jar spent a very large percentage of its runtime calling Hashtable.contains().
    - This was proven to be unnecessary.
    - After fix, JAR runs many times faster!

Too Big to Fail

Tips for Massive Enterprise Applications

- Tip #6: How can I get more accurate output from hprof?
- Try disabling Hotspot method inlining.
  - Method inlining is an important compiler optimization used by Hotspot – in general it should always be enabled.
  - But when a method is inlined the stack trace may be unable to show the exact line of code that is being executed.
  - Inlining can be disabled with the java command line flag -XX:-Inline.
  - *This will adversely impact performance.*

Static Analysis in Search for Performance Antipatterns

- Yonita - code analysis tool
  - Finds repeated calls to methods which could be stored and reused
  - Unused objects
- Static analysis doesn't replace code reviews

Where are the slides?

- Presentation slides and audio are at: http://wiki.nike.com/wiki/display/GtmsDev/JavaOne+2010+Presentations
- Pictures are on Flickr at: http://www.flickr.com/photos/douglasbullard/sets/72157624909234141/