

The Java “Tiger” Release Java2 Standard Edition 1.5



Dr. Bruce K. Haddon

Sun Professional Services

Bruce.Haddon@sun.com



Based on:

What's New in Java?

A presentation by:

Dr. Mark Reinhold

J2SE Chief Engineer

Sun Microsystems, Inc.



History of Releases

1.4	Merlin	2002/2/13
1.4.1	Hopper	2002/10/16
1.4.2	Mantis	2003/5/29
1.5	Tiger	2004/Q3
1.5.1	Dragonfly	2005/H1
1.6	Mustang	2006/?



Tiger Umbrella JSR 176: Expert Group

- Apache
- Apple
- BEA
- Borland
- Cisco Systems
- Fujitsu Limited
- Hewlett Packard
- IBM
- Macromedia
- Nokia
- Oracle
- SAP AG
- SAS Institute
- Savaje
- Sun Microsystems
- Osvaldo Doederlein
- Juergen Kreieder
- John Zukowsk

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



Tiger Component JSR's

003	<i>JMX Mgmt API</i>	174	<i>Monitoring & Mgmt</i>
013	<i>Decimal Arithmetic</i>	175	<i>Metadata</i>
014	<i>Generic Types</i>	200	<i>Pack Transfer Format</i>
028	<i>SASL</i>	201	<i>Language Updates</i>
114	<i>JDBC Rowsets</i>	202	<i>Class File Spec. Update</i>
133	<i>New Memory Model</i>	204	<i>Unicode Surrogates</i>
163	<i>Profiling API</i>	206	<i>JAXP 1.3</i>
166	<i>Concurrency Utilities</i>		



Overview

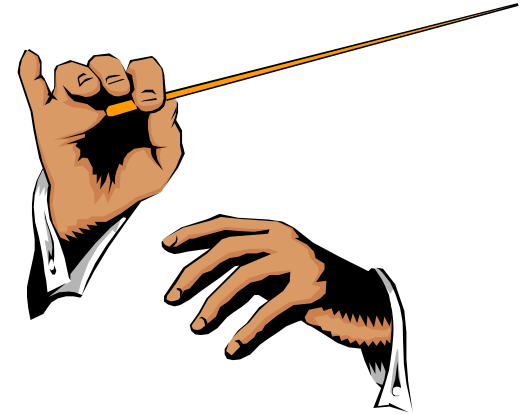
Quality

Ease of development

Performance

Integration

How you can help



What comes next

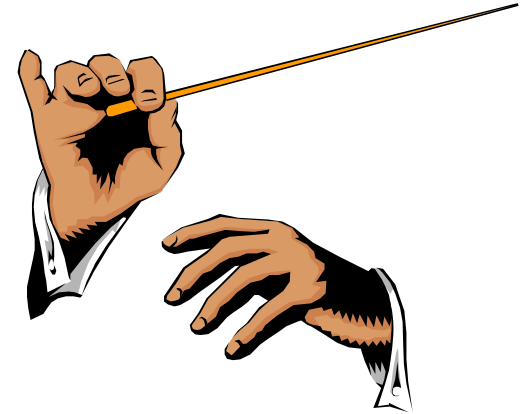
Quality

Ease of development

Performance

Integration

How you can help



Quality

Priorities

1. Compatibility



Quality

Priorities

- 1. Compatibility**
- 2. Compatibility**



Quality

Priorities

- 1. Compatibility**
- 2. Compatibility**
- 3. Compatibility**



Quality

Priorities

- 1. Compatibility**
- 2. Compatibility**
- 3. Compatibility**
- 4. Reliability**
- 5. Availability**
- 6. Performance**



What comes next

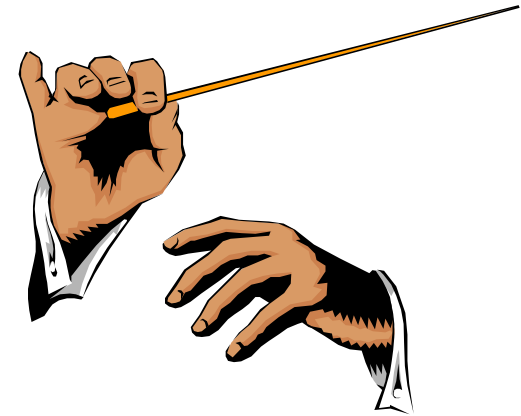
Quality

Ease of development

Performance

Integration

How you can help



Ease of Development

Generic types

```
// This HashMap maps Strings to Mammals  
HashMap m = new HashMap();  
m.put("wombat", new Mammal());  
Mammal w = (Mammal) m.get("wombat");
```



Ease of Development

Generic types

```
// This HashMap maps Strings to Mammals
HashMap m = new HashMap();
m.put("wombat", new Mammal());
Mammal w = (Mammal) m.get("wombat");

m.put("gecko", new Lizard());
Mammal x = (Mammal) m.get("gecko");
```

X *Runtime error: ClassCastException*



Ease of Development

Generic types

```
public class HashMap<K, V>
    extends AbstractMap<A, B>
    implements Map<K, V>,
               java.lang.Cloneable,
               java.io.Serializable
{
    ...
    public V get(java.lang.Object key)
    ...
    public V put(K key, V value)
    ...
}
```



Ease of Development

Generic types

```
// This HashMap maps Strings to Mammals
HashMap<String, Mammal> m =
    new HashMap<String, Mammal>();
m.put("wombat", new Mammal());
Mammal w = (Mammal) m.get("wombat");

m.put("gecko", new Lizard());
```

X Compile-time error: Type mismatch



Ease of Development

Generic types

- No primitive type parameterization
- No parameterized types in catch's
- Class files will have new signature" information
- Compiling against old class files flagged by "unchecked" compiler message
- No "typedef's"



Ease of Development

Enhanced for loops

With generic types, for-loop iteration idiom simplifies to:

```
Collection<String> c = ... ;  
for( Iterator<String> i = c.iterator();  
      i.hasNext(); )  
{  
    String s = i.next();  
    ...  
}
```



Ease of Development

Enhanced for loops

```
Collection<String> c = ... ;  
for( Iterator<String> i = c.iterator();  
     i.hasNext(); )  
{  
    String s = i.next();  
    ...  
}
```

*What could be become a well known idiom above
for using iterators, becomes:*

```
for( String s : c )  
{  
    ...  
}
```



Ease of Development

Enhanced for loops

So, iterating a collection:

```
for( String s : c )  
{  
    ...  
}
```

For comparison, the enhanced for-loop for arrays looks like:

```
int[] a = ...  
int sum = 0;  
for( int n : a )  
    sum += n;
```



Ease of Development

Enumerated types *(current idiom)*

```
public class Toss
{
private String name;
private Toss(String nm) { name = nm; }
public static final Toss HEADS = new Toss("HEADS");
public static final Toss TAILS = new Toss("TAILS");
public String toString() { return name; }
}
```



Ease of Development

Enumerated types

```
public class Toss
{
    private String name;
    private Toss(String nm) { name = nm; }
    public static final Toss HEADS = new Toss("HEADS");
    public static final Toss TAILS = new Toss("TAILS");
    public String toString() { return name; }
}
```

```
public enum Toss { HEADS, TAILS };
```



Ease of Development

Enumerated types

```
// All the Toss values
Toss list_of_values[] = Toss.values();

// A particular value
Toss head = Toss.valueOf("HEADS");

// The "name" of the value
String name = head.toString();
```

Enums are (almost) fully-fledged classes, and may have programmer-defined methods and other fields.

Their superclass may not be accessed, nor may they be subclassed.



Ease of Development

Autoboxing

`Integer x = new Integer(3);` → `Integer x = 3;`

`int y = x.intValue();` → `int y = x;`

`map.put(new Integer(1),
 new Integer(42));` → `map.put(1, 42);`

*This is not a change to the JVM,
all the “magic” happens in the compiler.*



Ease of Development

Formatting & scanning *(current)*

```
BufferedReader br =  
    new BufferedReader(  
        new InputStreamReader(System.in));  
  
System.out.print("Fahrenheit: ");  
String ln = br.readLine();  
StringTokenizer st =  
    new StringTokenizer(ln);  
double f =  
    Double.parseDouble(st.nextToken());  
out.println("Celsius: " + ((f - 32) / 1.8));
```



Ease of Development

Formatting & scanning

```
BufferedReader br =  
    new BufferedReader(  
        new InputStreamReader(System.in));
```

```
System.out.print("Fahrenheit: ");
```

```
String ln = br.readLine();
```

```
StringTokenizer st =  
    new StringTokenizer(ln);
```

```
double f =
```

```
    Double.parseDouble(ln.split("\\s+")[0]);
```

```
out.println("Celsius: " + ((f - 32) / 1.8));
```



Ease of Development

Formatting & scanning *(default format)*

```
BufferedReader br =  
    new BufferedReader(  
        new InputStreamReader(System.in));  
System.out.print("Fahrenheit: ");  
String ln = br.readLine();  
double f = Double.parseDouble(ln.split("\\s+")[0]);  
out.println("Celsius: " + ((f - 32) / 1.8));
```

```
% java Conv  
Fahrenheit: 32.1  
Celsius: 0.055555555555555556344  
%
```



Ease of Development

Formatting & scanning *(current format)*

```
BufferedReader br = new BufferedReader(...);
System.out.print("Fahrenheit: ");
String ln = br.readLine();
double f = Double.parseDouble(ln.split("\\s+")[0]);
MessageFormat mf =
    new MessageFormat("Celsius: {0,number,.##}");
out.println(mf.format(new Object[] {
    new Double((f - 32) / 1.8) }));
```

```
% java Conv
Fahrenheit: 32.1
Celsius: 0.06
%
```



Ease of Development

Formatting & scanning *(new)*

```
System.out.print("Fahrenheit: ");  
Scanner sc = new Scanner(System.in);  
double f = sc.nextDouble();  
out.printf("Celsius: %.2f%n", (f - 32) / 1.8);
```

```
% java Conv  
Fahrenheit: 32.1  
Celsius: .06  
%
```



Ease of Development

Varargs *(the way things were ...)*



```
public class PrintStream ...
{
...
public PrintStream printf(String fmt, Object[] args)
{ ... }
...
}
```



Ease of Development

Varargs *(or would have been ...)*



```
public class PrintStream ...
{
...
public PrintStream printf(String fmt, Object[] args)
{ ... }
...
}
```

```
out.printf("%4d/%08X %tY-%<tm-%<tD %s%n",
           new Object[] { new Integer(bits),
                           new Integer(fingerprint),
                           date, user });
```



Ease of Development

Varargs

```
public class PrintStream ...
{
...
public PrintStream printf(String fmt, Object ... args)
{...}
...
}
```

```
out.printf("%4d/%08X %tY-%<tm-%<tD %s%n",
           bits, fingerprint, date, user);
```



Ease of Development

Concurrency utilities *(The Doug Lea Toolkit)*

java.util.concurrent.atomic

`AtomicBoolean`
`AtomicInteger`
`AtomicIntegerArray`
`AtomicIntegerFieldUpdater`
`AtomicLong`
`AtomicLongArray`
`AtomicLongFieldUpdater`
`AtomicMarkableReference`
`AtomicReference`
`AtomicReferenceArray`
`AtomicReferenceFieldUpdater`
`AtomicStampedReference`

java.util.concurrent.locks

`Condition`
`Lock`
`ReadWriteLock`
`LockSupport`
`ReentrantLock`
`ConditionObject`
`ReentrantReadWriteLock`



Ease of Development

Concurrency utilities *(The Doug Lea Toolkit)*

java.util.concurrent

ArrayBlockingQueue

CancellableTask

ConcurrentHashMap

ConcurrentLinkedQueue

CopyOnWriteArrayList

CopyOnWriteArraySet

CountDownLatch

CyclicBarrier

DelayQueue

Exchanger

Executors

FairSemaphore

FutureTask

LinkedBlockingQueue

PriorityBlockingQueue

ScheduledExecutor

Semaphore

SynchronousQueue

ThreadPoolExecutor

AbortPolicy

CallerRunsPolicy

DiscardOldestPolicy

DiscardPolicy

TimeUnit



Ease of Development

Concurrency utilities

```
ServerSocketChannel ssc =  
    ServerSocketChannel.open();  
  
...  
  
for (;;)   
{  
    SocketChannel sc = ssc.accept();  
    new Thread(new Handler(sc)).start();  
}
```



Ease of Development

Concurrency utilities

```
import java.util.concurrent.*;

ServerSocketChannel ssc =
    ServerSocketChannel.open();

Executor xec = Executors.newCachedThreadPool();

for (;;)
{
    SocketChannel sc = ssc.accept();
    Cancellable c =
        xec.execute(new Handler(sc));
    ...
}
```



Ease of Development

Annotations (*a.k.a.* metadata)

```
/**
 * Associates a copyright notice with the
 * annotated API element.
 */
public @interface Copyright
{
    String value();
}

...

@Copyright("2002 Yoyodyne Propulsion"
           "Systems, Inc., All rights reserved.")
public class OscillationOverthruster
{ ... }
```



Ease of Development

Annotations (*a.k.a.* metadata)

```
public class StockQuote
    implements java.rmi.Remote
{ ... }
```

```
public class Demo
    implements StockQuote
{
    ...

    public float getPrice(String symbol);
}
```



Ease of Development

Annotations (*a.k.a.* metadata)

```
public class StockQuote
    implements java.rmi.Remote
{ ... }
```

```
@WebService(contextRoot = "Demo",
             style = WRAPPED,
             serviceImpl = StockQuote.class,
             serviceElement = "StockQuoteService")
```

```
public class Demo
    implements StockQuote
{
    ...
    @Operation
    public float getPrice(String symbol);
}
```



Ease of Development

Annotations (*a.k.a.* metadata)

Defining the metadata structure

```
public @interface WebService
{
    String contextRoot();
    WSDLStyle style();
    WSDLUse use() default LITERAL;
    Class serviceImpl();
    String serviceElement();
}

public @interface Operation { }
```



Ease of Development

Annotations (*a.k.a.* metadata)

- *java.lang.annotations.Annotations*
- Additions to class files for annotations
- API provided to read annotations
- Access controlled by retention policy metatags
(*i.e.*, `@Retention(SOURCE)` or `CLASS` or `RUNTIME`)



Ease of Development

And more ...

- **Static imports**
- **Auto serial UID generation**
- **Decimal arithmetic ease of use and more specific control of accuracy and rounding**
- **Bit manipulation**
- **Document and make available: `javac.Main`**
- **Document and make available: `javadoc.Main`**



What comes next

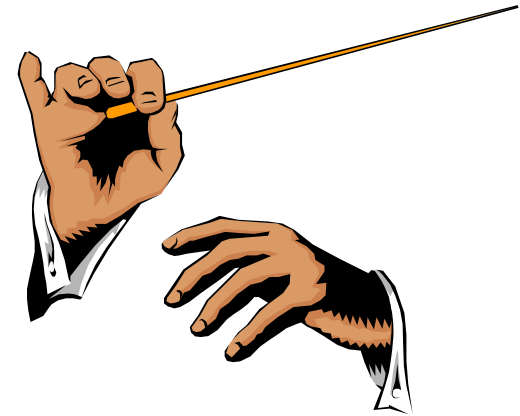
Quality

Ease of development

Performance

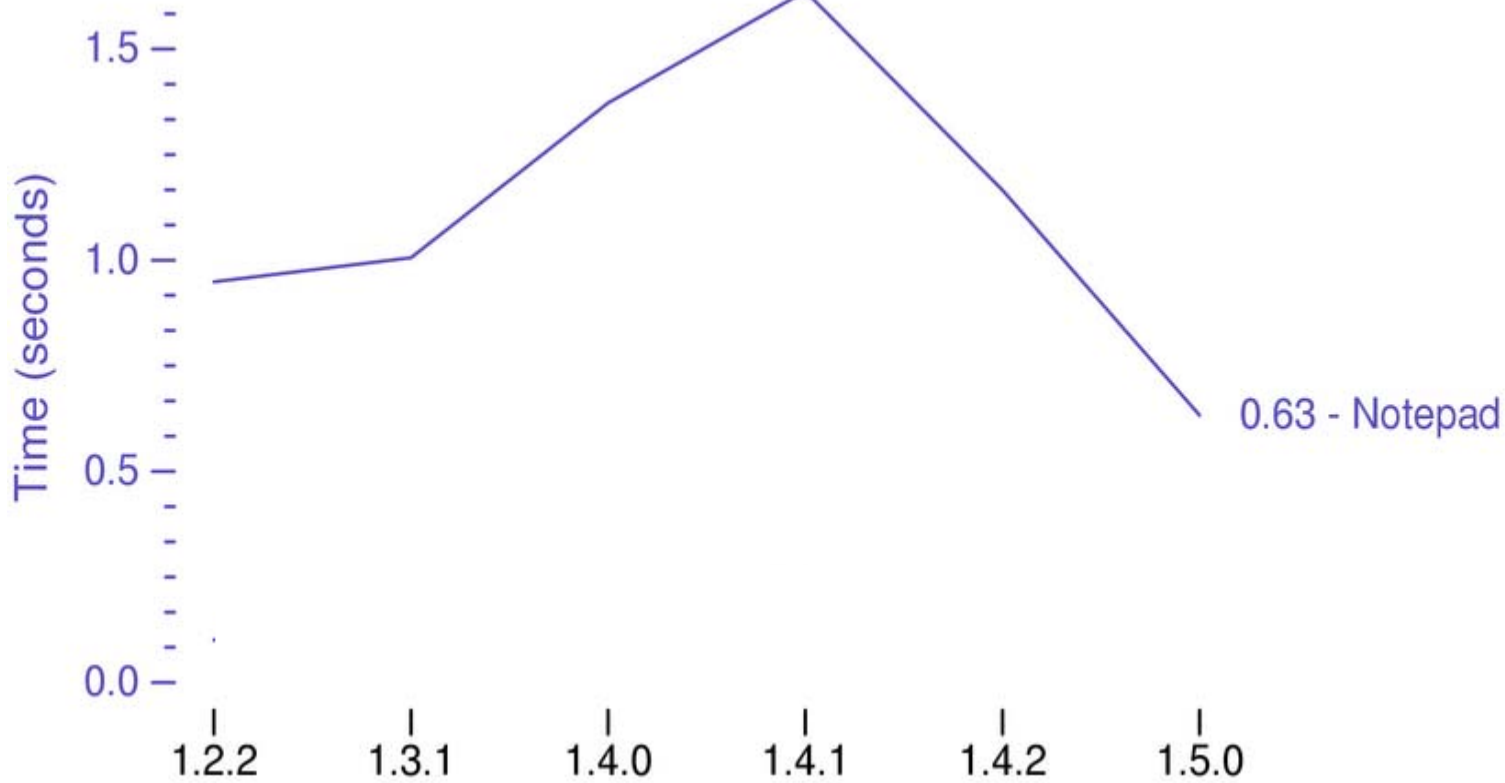
Integration

How you can help



Performance

Startup time

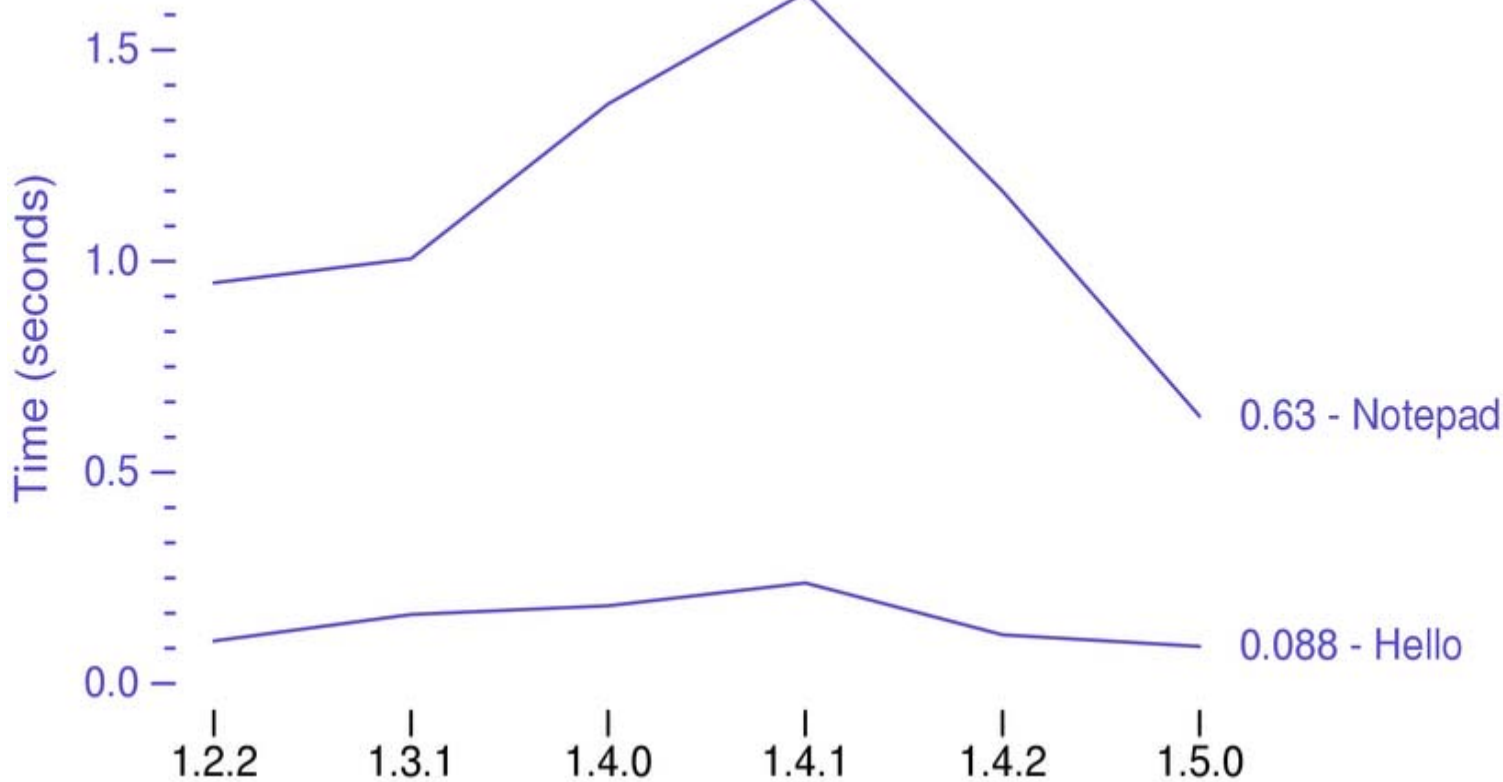


1.5GHz Athlon, 512MB, Linux 2.4.20



Performance

Startup time

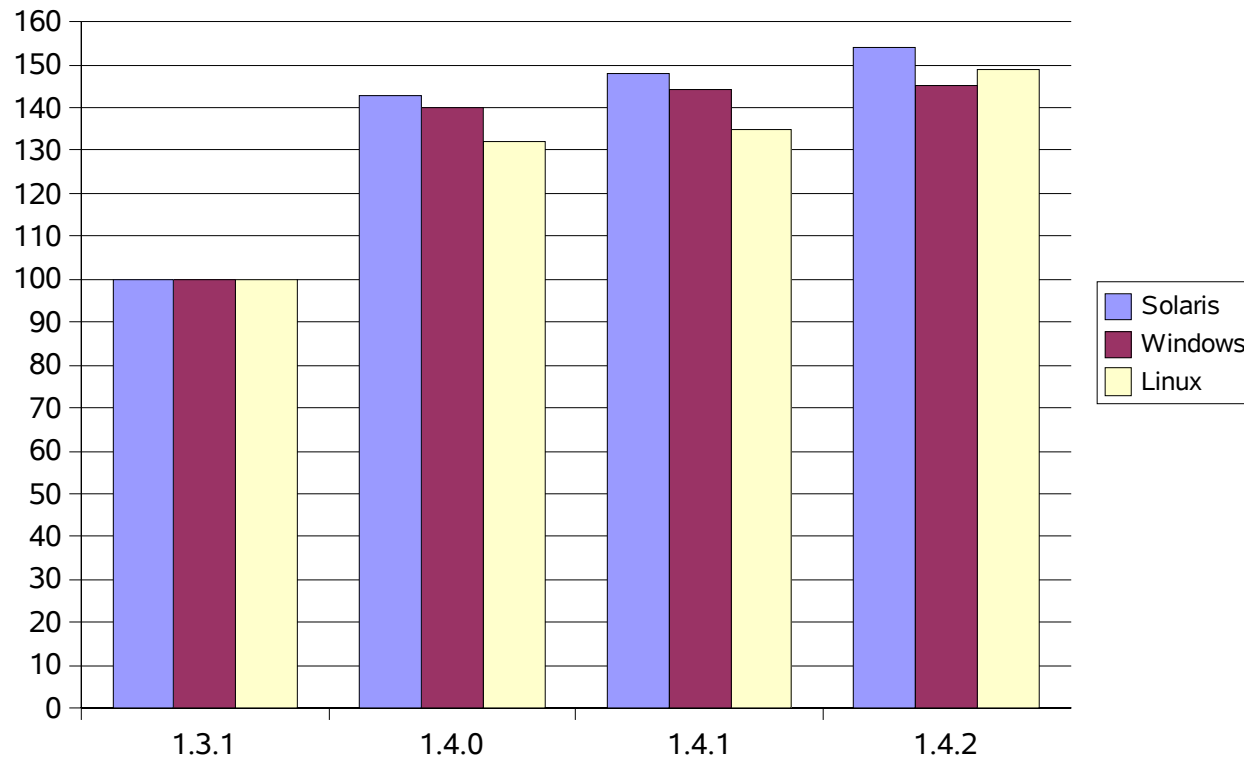


1.5GHz Athlon, 512MB, Linux 2.4.20



Performance

Graphics performance



SwingMark (bigger is better)



Performance

New garbage collectors

```
% # Default serial collector  
% java Foo
```



Performance

New garbage collectors

```
% # Default serial collector
```

```
% java Foo
```

```
% # Throughput collector
```

```
% java -XX:+UseParallelGC Foo
```



Performance

New garbage collectors

```
% # Default serial collector
```

```
% java Foo
```

```
% # Throughput collector
```

```
% java -XX:+UseParallelGC Foo
```

```
% # Concurrent low-pause collector
```

```
% java -XX:+UseConcMarkSweepGC \  
      -XX:+UseParNewGC Foo
```

<http://java.sun.com/docs/hotspot/gc1.4.2/>



Performance

Ergonomics

```
% # Uses client compiler, starts w/ small heap  
% java BigFatApp
```



Performance

Ergonomics

```
% # Uses client compiler, starts w/ small heap
```

```
% java BigFatApp
```

```
% # More realistically...
```

```
% java -server -XX:UseParallelGC -Xmx1G BigFatApp
```



Performance

Ergonomics

```
% # Uses client compiler, starts w/ small heap
```

```
% java BigFatApp
```

```
% # More realistically...
```

```
% java -server -XX:UseParallelGC -Xmx1G BigFatApp
```

```
% # In Tiger, on a big machine, you just say
```

```
% java BigFatApp
```



Performance

Ergonomics

```
% java -XX:UseParallelGC \  
-XX:MaxGCPauseMillis=ms \  
-XX:GCTimeRatio=n \  
    Foo
```



Performance

New I/O: Simple file copy

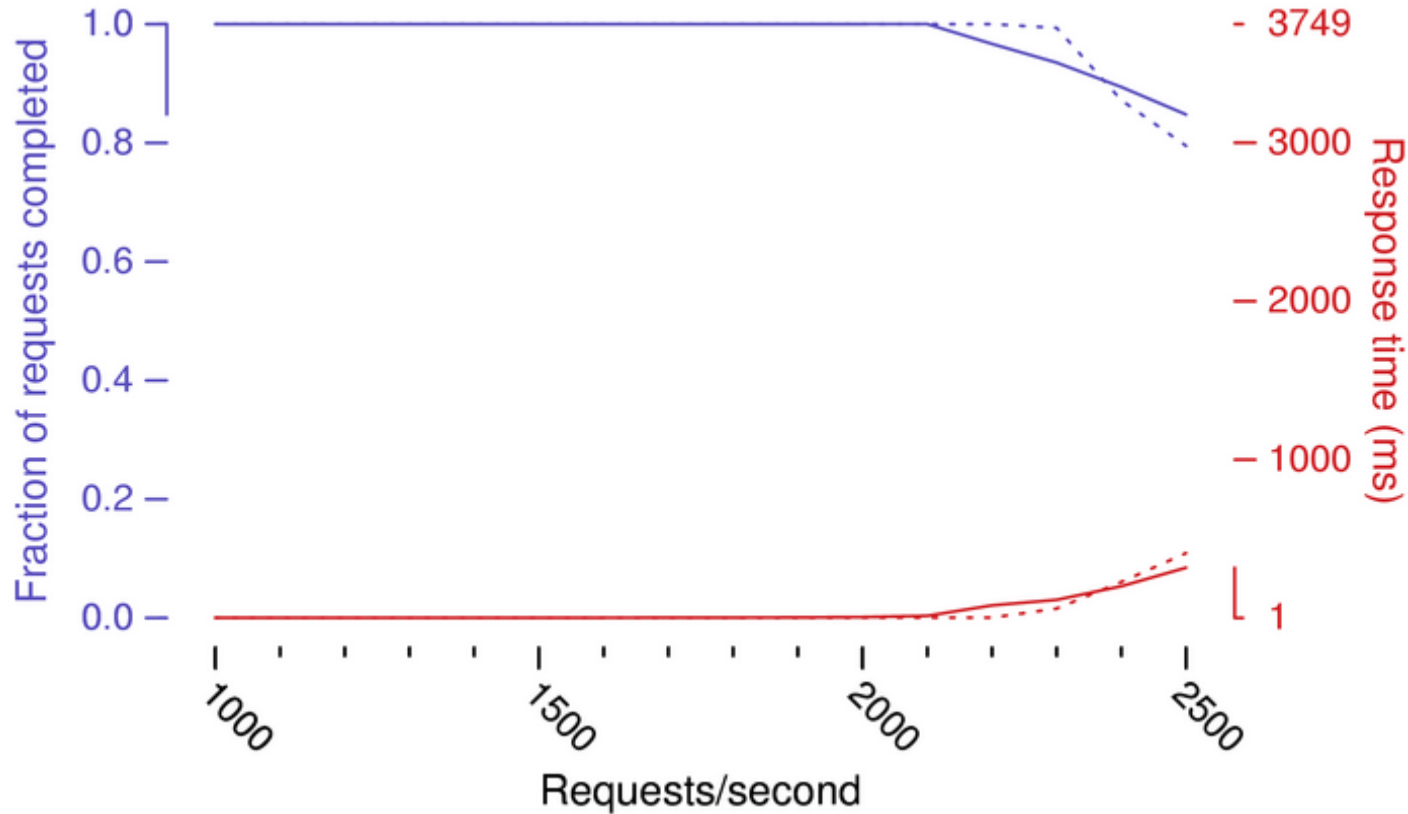
	MB/s speedup	
Old read/write	69.86	1.00
New read/write	102.98	1.47
Mapped copy	132.45	1.90
Transfer	148.12	2.12
Native (C) transfer	150.86	2.16

500 MHz Pentium 3, 256MB, Linux 2.2.19



Performance

New I/O: Web server, small files

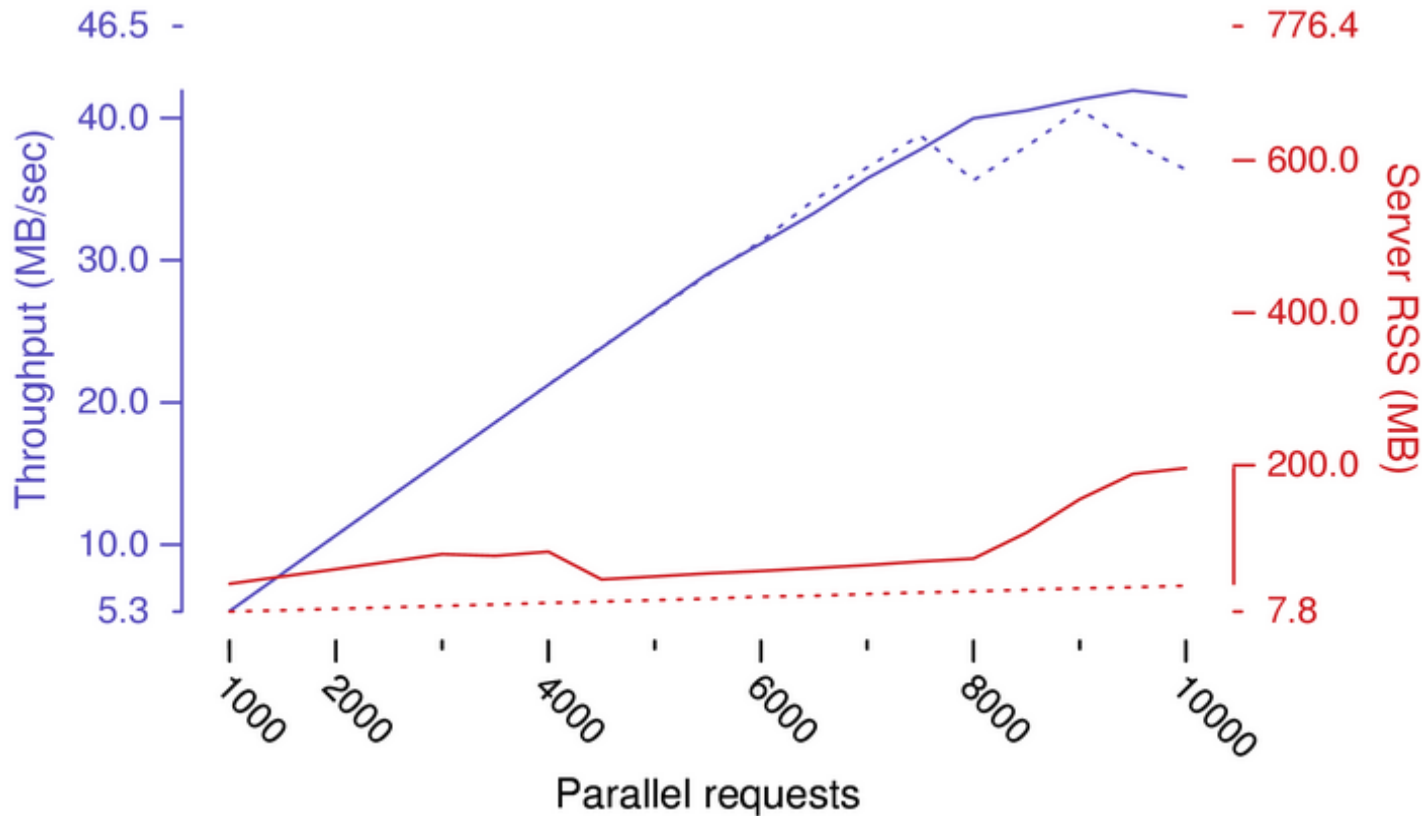


4x480MHz UltraSparc II, 1GB, Solaris 9, GB ethernet



Performance

New I/O: Web server, large files

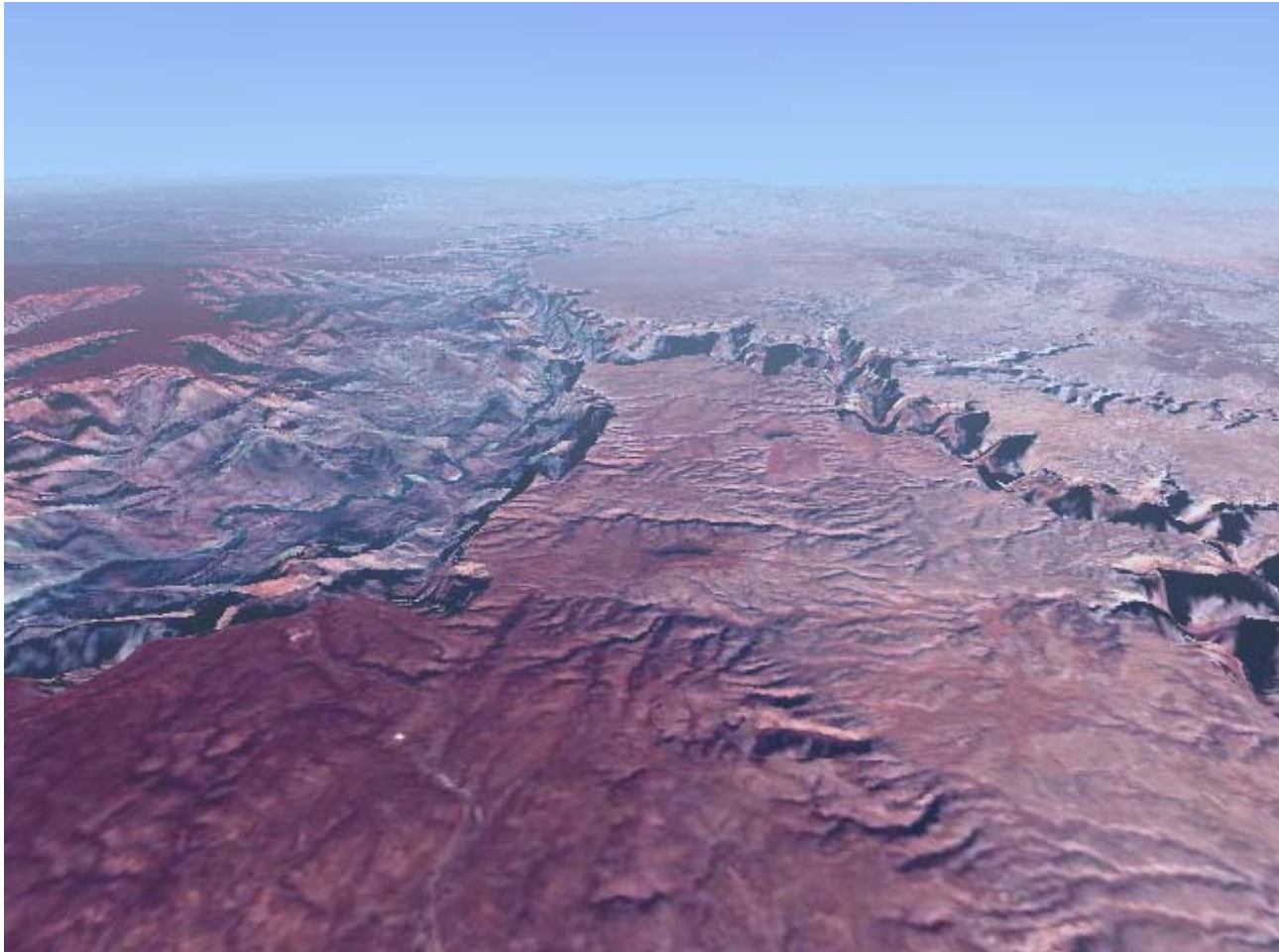


4x480MHz UltraSparc II, 1GB, Solaris 9, GB ethernet



Performance

NIO: Direct memory access



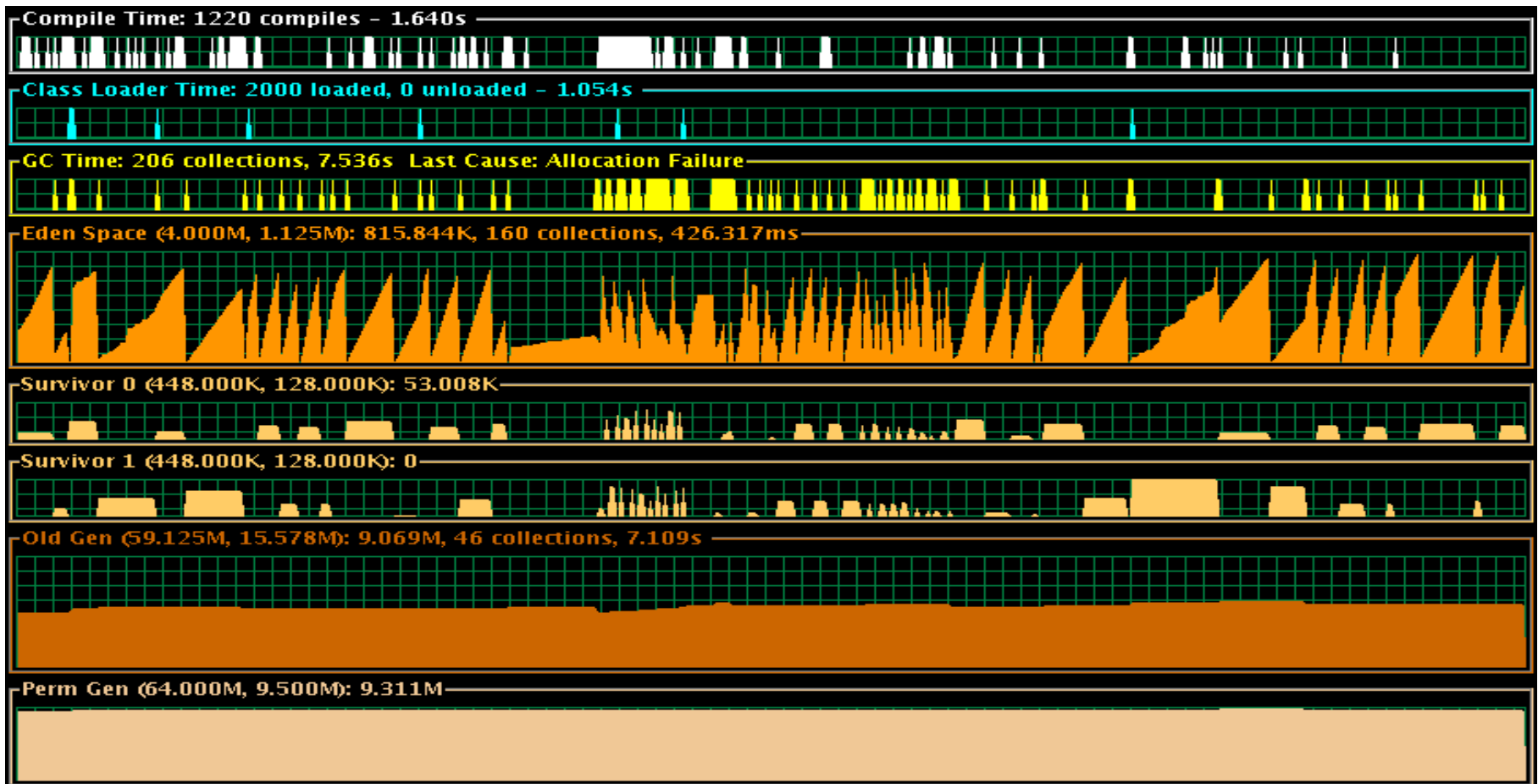
Performance

jvmstat

```
% jvmstat -gcutil -h10 6617 1000
  S0      S1      E      O      P      YGC      YGCT  FGC      FGCT      GCT
  0.00    9.18    78.92   51.67   98.21   133      0.322   34      4.596    4.918
 10.84    0.00    25.84   51.67   98.21   134      0.323   34      4.596    4.919
 10.84    0.00    68.71   51.67   98.21   134      0.323   34      4.596    4.919
  0.00    8.74    18.80   51.67   98.21   135      0.324   34      4.596    4.920
  0.00    8.74    68.11   51.67   98.22   135      0.324   34      4.596    4.920
 17.72    0.00    14.86   51.67   98.22   136      0.325   35      4.596    4.921
  0.00   82.43    68.01   51.74   99.11   137      0.329   35      4.735    5.064
 28.41    0.00    80.99   52.10   99.11   138      0.331   35      4.735    5.066
 18.41    0.00     0.00   52.10   99.11   140      0.334   35      4.735    5.069
  0.00   21.11   40.14   52.10   99.11   141      0.335   35      4.735    5.070
  S0      S1      E      O      P      YGC      YGCT  FGC      FGCT      GCT
  0.00   17.04     3.30   52.10   99.11   143      0.337   35      4.735    5.073
 14.98    0.00   40.80   52.10   99.11   144      0.339   35      4.735    5.074
  0.00   13.61   50.33   52.10   99.11   145      0.341   35      4.735    5.076
 10.33    0.00   99.53   52.10   99.11   146      0.342   35      4.735    5.077
...
```

Performance

Visual GC



Performance

Other performance projects

- **Profiling**
- **Java2D on OpenGL**
- **Global startup & footprint**
- **Unsynchronized StringBuffer's**
- **“Crunched” class files**
- **Refined memory model**
- **And much more ...**



What comes next

Quality

Ease of development

Performance

Integration

How you can help



Integration

Unicode Surrogates

- **Unicode 3.1 expanded into 21-bit codes**
- **Unicode 4.0 added more**
- **New API's needed to deal with characters larger than 16 bits**
- **Difficult to not break compatibility**



Integration

Unicode Surrogates

```
// Solution: map surrogates as code-points  
// into int's (32 bits) - Examples:
```

```
// Existing String access method  
char charAt(int index)
```

```
// New String access method for possibly extended  
// characters  
int codePointAt(int index)
```



```
// Search String for an extended character  
int indexOf(int ch)
```





Integration

Unicode Surrogates

- **Disadvantages** 
 - Exposes code-point mappings
 - Makes special cases of characters outside the BMP
 - Breaks standardization with UTF-8
 - Makes “character” indexing uncertain
 - Still leaves normalization an open question
- **Preserves existing API's** 
(see Compatibility Rules 2 and 3).



Integration

getenv

```
out.println(System.getenv("PATH"));
```



Integration

getenv

```
out.println(System.getenv("PATH"));
```

```
for( Map.Entry<String, String> me : System.getenv() )  
    out.printf("%s=%s%n",  
              me.getKey(), me.getValue());
```



Integration

Multiple JRE support *(current)*

```
% /usr/local/java/1.4.1/bin/java Foo
```



Integration

Multiple JRE support

```
% /usr/local/java/1.4.1/bin/java Foo
```

```
% java -version:1.4.1+ Foo
```



Integration

Multiple JRE support

```
% /usr/local/java/1.4.1/bin/java Foo
```

```
% java -version:1.4.1+ Foo
```

```
% vi manifest →
```

```
Main-Class: Foo  
JRE-Version: 1.4.1+
```

```
% jar cfm manifest foo.jar Foo.class
```

```
% java -jar foo.jar
```



Integration

Other integration features

- **Enum's added to serialization**
- **Inherited channels**
- **XML & JDBC**
- **Monitoring & Manageability**
- **New Swing look-and-feels**
- **Auto update**
- **And much more ...**



What comes next

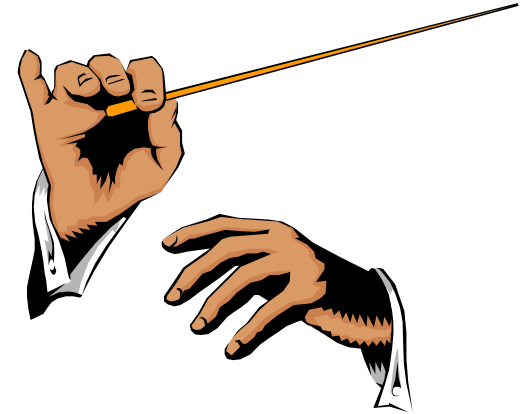
Quality

Ease of development

Performance

Integration

How you can help



How you can help

Three things to avoid (1)

```
// The GC knows better than you do  
System.gc()
```



How you can help

Three things to avoid (2)

```
// The GC knows better than you do  
System.gc()
```



```
// Finalize methods are "unreliable"  
// and degrade performance  
public void finalize() { ... }
```



How you can help

Three things to avoid (3)

```
// The GC knows better than you do  
System.gc()
```



```
// Finalize methods are "unreliable"  
// and degrade performance  
public void finalize() { ... }
```

```
// All sun.* packages are J2SE internal  
// interfaces and should NEVER be used  
import sun.misc.*;
```



How you can help

Try out Tiger!

- **Beta1** **2004-02-04**
- **Beta2** **2004/Q2**
- **FCS** **2004/Q3**



<http://java.sun.com/j2se/1.5>



Sources

- **Java Community Process**
 - <http://www.jcp.org/>
- **Java2 Standard Edition 1.5**
 - <http://java.sun.com/j2se/1.5>
- **Java “Hot-spot” JVM**
 - <http://java.sun.com/docs/hotspot/gc1.4.2>
- **Professor Doug Lea’s Workstation**
 - <http://gee.cs.oswego.edu/dl/>



