



J2SE™ 5.0: WATCH AND HEAR THE TIGER ROAR!

Matt Hosanee

Partner Engineering

Sun Microsystems

1. J2SE 5.0 design themes
2. Language/Compiler changes
 - > Generics & metadata
3. Library API changes
 - > Concurrency utilities
4. Virtual machine
 - > Self tuning
5. Conclusions and resources

1. J2SE 5.0 Design Themes

- Reliability, Availability, Serviceability
- Monitoring and Manageability
- Scalability and Performance
- XML and Client Web Services
- Ease of Development

- Generics
- Autoboxing/Unboxing
- Enhanced for loop (“foreach”)
- Type-safe enumerations
- Varargs
- Static import
- Metadata



Generics - Boxing - foreach - new enums - varargs - static imports - metadata

- The problem:

```
Vector v = new Vector();  
v.add(new Integer(4));  
OtherClass.expurgate(v);
```

...

```
static void expurgate(Collection c) {  
    for (Iterator it = c.iterator(); it.hasNext(); )  
        /* ClassCastException */  
        if (((String)it.next()).length() == 4)  
            it.remove();  
}
```

Generics - Boxing - foreach - new enums - varargs - static imports - metadata

- Using Generics

Instantiate a generic class to create type specific object

Example:

```
Vector<String> x = new Vector<String> ();  
x.add(new Integer(5)); // Compiler error!
```

```
Vector<Integer> y = new Vector<Integer> ();  
return x.getClass() == y.getClass(); // ?
```

Generics - Boxing - foreach - new enums - varargs - static imports - metadata

- Compatability of generics

Raw Types for existing code

```
/* Old class */  
public Vector getVector() { return new Vector(); }
```

```
/* New class */  
public Vector<String> s = oldCode.getVector();  
/* Generates compiler warning, not error */
```

Generics - **Boxing** - **foreach** - **new enums** - **varargs** - **static imports** - **metadata**

- **Generic classes**

Add type parameters in class definition

Typically use one capital letter for types

Use anywhere in class that type is required

```
public class Pair<F, S> {  
    F first; S second;  
  
    public Pair(F f, S s) {  
        first = f; second = s;  
    }  
}
```

Generics - **Boxing** - **foreach** - **new enums** - **varargs** - **static imports** - **metadata**

- **Wildcards**

How to print the contents of any collection?

```
void printCollection(Collection<Object> c) {  
    for (Object o : c)  
        System.out.println(o);  
}
```

<?> = wildcard = unknown

```
void printCollection(Collection<?> c) {  
    for (Object o : c)  
        System.out.println(o);  
}
```

Generics - Boxing - foreach - new enums - varargs - static imports - metadata

- Bounded wildcards

Parameterised types do not have inheritance like objects

A wildcard can be specified with an upper bound

```
public void drawAll(List<? extends Shape>s) {  
    ...  
}
```

```
List<Circle> c = getCircles();  
drawAll(c);  
List<Triangle> t = getTriangles();  
drawAll(t);
```

Generics - Boxing - foreach - new enums - varargs - static imports - metadata

- **Methods & generics**

```
static void aToC(Object[] a, Collection<?> c) {  
    for (Object o : a)  
        c.add(o);    /* COMPILER ERROR */  
}
```

```
static <T> void aToC(T[] a, Collection<T> c) {  
    for (T o : a)  
        c.add(o);    /* No compiler error */  
}  
  
String[] sa = new String[100];  
Collection<Object> co = new ArrayList<Object>();  
Collection<String> cs = new ArrayList<String>();  
aToC(sa, cs);    /* T inferred to be String */  
aToC(sa, co);    /* T inferred to be Object */
```

Generics - Boxing - foreach - new enums - varargs - static imports - metadata

- Bounded wildcards (again)

Wildcards do not use type inference

They can have lower bounds

```
class CopyUtil {  
    public <T>void copy(List<T> src,  
                       List<? super T> dest) {  
        for (T o : src)  
            dest.add(o);  
    }  
}
```

Generics - Boxing - foreach - new enums - varargs - static imports - metadata

- Type erasure

The compiler "erases" type information

- > Bytecodes the same as pre J2SE 5
- > Security is therefore unaffected
 - > Verifier unchanged
- > Performance unaffected
 - > No gains, no losses
- > Type information **is** recorded in class file
 - > It must be there to enable linking of class files

Generics - **Boxing** - foreach - new enums - varargs - static imports - metadata

- Problem:
 - > Conversion between primitive types and wrapper objects (and vice-versa)
 - > Needed when adding primitives to a collection
- Solution: Let the compiler do it

```
Byte byteObj = 22;           // Boxing conversion
int i = byteObj              // Unboxing conversion
```

```
ArrayList al = new ArrayList();
al.add(22); // Boxing conversion
```

Generics - Boxing - **foreach** - new enums - varargs - static imports - metadata

- Problem:
 - > Iterating over collections is tricky
 - > Often, iterator only used to get an element
 - > Iterator is error prone
(Can occur three times in a for loop)
 - > Can produce subtle runtime errors
- Solution: Let the compiler do it
 - > New for loop syntax
`for (variable : collection)`
 - > Works for Collections and arrays

Generics - Boxing - **foreach** - new enums - varargs - static imports - metadata

- Old code

```
void cancelAll(Collection c) {  
    for (Iterator i = c.iterator(); i.hasNext(); ){  
        TimerTask task = (TimerTask)i.next();  
        task.cancel();  
    }  
}
```

- New Code

```
void cancelAll(Collection<TimerTask> c) {  
    for (TimerTask task : c)  
        task.cancel();  
}
```

Generics - Boxing - foreach - **new enums** - varargs - static imports - metadata

- Problem:
 - Variable needs to hold limited set of values
 - e.g. card suit can only be spade, diamond, club, heart
- Solution: New type of class declaration
 - enum type has public, self-typed members for each enum constant
 - New keyword, enum
 - Works with switch statement

Generics - Boxing - foreach - **new enums** - varargs - static imports - metadata

```
public enum Suit { spade, diamond, club, heart };  
public enum Value { ace, two, three, four, five,  
                  six, seven, eight, nine, ten,  
                  jack, queen, king };
```

```
List<Card> deck = new ArrayList<Card>();
```

```
for (Suit suit : Suit.values())  
    for (Value value : Value.values())  
        deck.add(new Card(suit, value));
```

```
Collections.shuffle(deck);
```

Generics - Boxing - foreach - new enums - **varargs** - static imports - metadata

- Problem:
 - > To have a method that takes a variable number of parameters
 - > Can be done with an array, but not nice
 - > Look at `java.text.MessageFormat`
- Solution: Let the compiler do it for you
 - > New syntax:

```
public static String format  
(String fmt, Object... args);
```

Generics - Boxing - foreach - new enums - varargs - **static imports** - metadata

- Problem:
 - > Having to fully qualify every static referenced from external classes
- Solution: New import syntax
 - > `import static TypeName.Identifier;`
 - > `import static Typename.*;`
 - > Also works for static methods and enums

e.g `Math.sin(x)` becomes `sin(x)`

Generics - Boxing - foreach - new enums - varargs - static imports - **metadata**

Metadata (JSR 175)

- Provide standardised way of adding annotations to Java code
- Like Serializable interface, javadoc comments and Xdoclets, but better
- Annotations are used by tools that work with Java code:
 - > Compiler
 - > IDE
 - > Runtime tools

Generics - Boxing - foreach - new enums - varargs - static imports - **metadata**

Defining Metadata

- Defined like an interface
 - > @interface
- Definition contains parameters that can be specified when using the annotation
- Default values can be provided
- For annotations with one parameter, the name 'value' is special

Generics - Boxing - foreach - new enums - varargs - static imports - **metadata**

Meta-annotations

- @Retention
 - > How long is annotation information kept
 - > **Enum RetentionPolicy**
 - > **SOURCE, CLASS, RUNTIME**
- @Target
 - > Restrictions on use of this annotation
 - > **Enum ElementType**
 - > **TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL VARIABLE, ANNOTATION_TYPE, PACKAGE**

Generics - Boxing - foreach - new enums - varargs - static imports - **metadata**

Example:

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Accessor {
    String variableName();
    String variableType() default "String";
}
```

Generics - Boxing - foreach - new enums - varargs - static imports - **metadata**

Code annotation

- Marker annotation
 - > `public int @BetaVersion getValue()`
- Single value annotation
 - > `@Copyright(value = "Sun Microsystems")`
 - > `@Copyright("Sun Microsystems")`
 - > This can only be used if member is called value
- Normal annotation
 - > `@Author(@Name(first="fred", last="bloggs"))`
 - > `@Contributors({"fred", "joe", "bill"})`

Generics - Boxing - foreach - new enums - varargs - static imports - **metadata**

Reflection

- Marker annotation

```
boolean isBeta = MyClass.class.isAnnotationPresent  
    (BetaVersion.class);
```

- Single value annotation

```
String copyright = MyClass.class.getAnnotation  
    (Copyright.class).value();
```

- Normal annotation

```
Name author = MyClass.class.getAnnotation(Author.class).  
    value();
```

```
String first = author.getFirst();
```

```
String last = author.getLast();
```

Generics - Boxing - foreach - new enums - varargs - static imports - **metadata**

Annotation Processing Tool

- Annotated base file generates derived files
 - > **apt** is tool to generate derived files
 - > Derived files can be new source files, deployment descriptor, etc
- Uses annotation processors
 - > Processors can be developed using the **com.sun.mirror** packages
 - > **process()** method processes specific annotations
 - > Access to **File** to create derived files
- **apt** can compile the derived files if required

3. Library API changes

- Simple formatted I/O & Scanner
- Concurrency utilities (JSR 166)
 - > Executors, Thread pool, callable/future, locks, read/write locks, semaphores, atomics
- Desktop Client
 - > Accessibility, Deployment, Performance (OpenGL/XAWT), GUI (GNOME skin, better Windows look, updated Java look)

Main changes - Performance - Monitoring - Compatability - Java.next

- Class data sharing
 - > Improved startup time (up to 30% faster)
 - > Reduced memory footprint
 - > -Xshare:on, -Xshare:dump
- Thread priority changes (JSR-133)
- Fatal error handling
 - > -XX:OnError="gcore %p; dbx - -%p"
- Simplified stack trace access
 - > `Thread.getStackTrace()`

Main changes - **Performance** - Monitoring - Compatability - Java.next

Server class machine

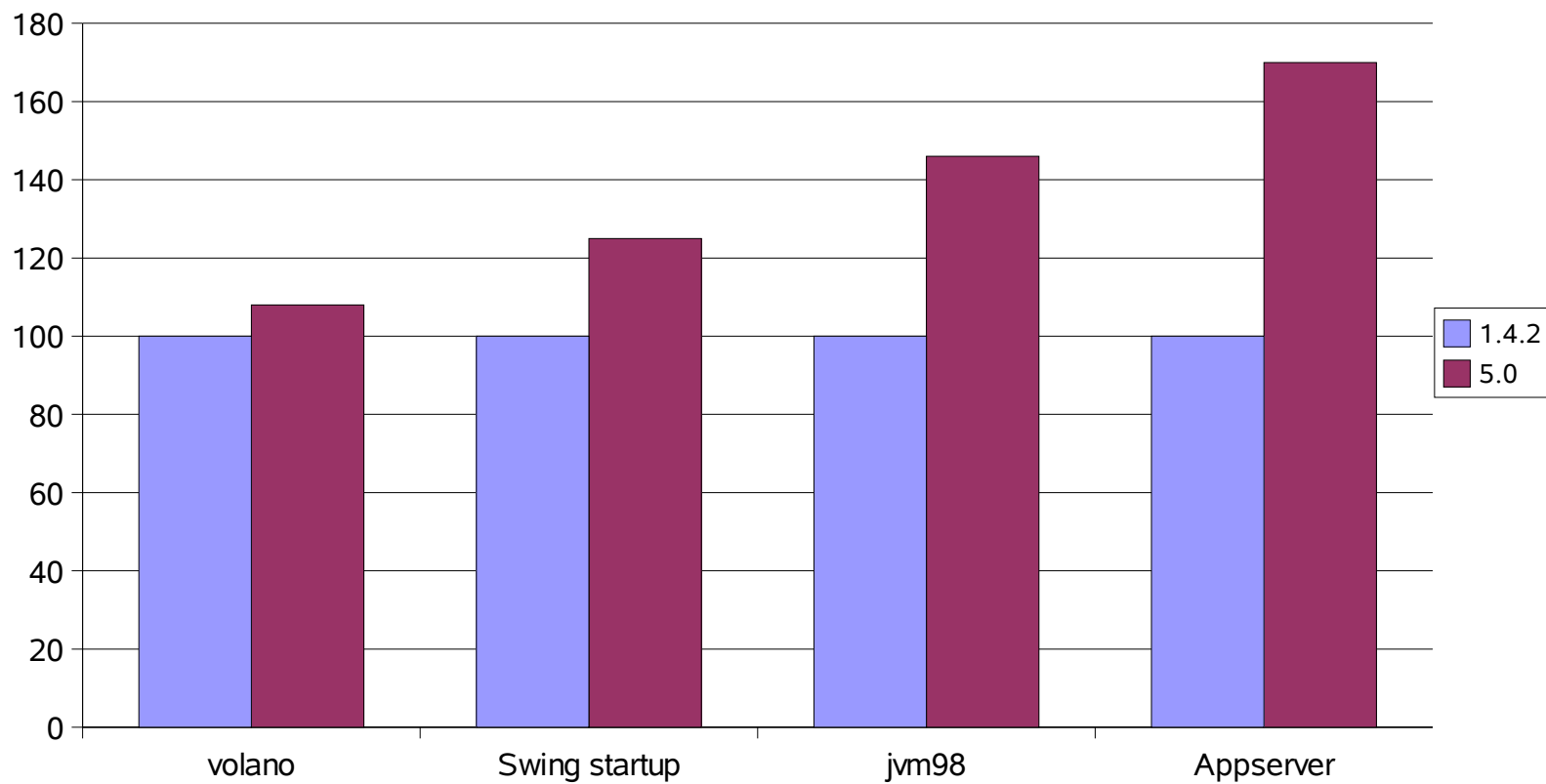
- Auto-detected
- 2 CPU, 2GB mem (except windows)
 - > Uses server compiler
 - > Uses parallel collector
 - > Initial heap size is 1/64 of physical memory up to 1GB
 - > Max heap size is 1/4 of physical memory up to 1GB

Main changes - **Performance** - Monitoring - Compatability - Java.next
Self tuning (ergonomics)

- Maximum pause time goal
 - > `-XX:MaxGCPauseMillis=<n>`
 - > This is a hint, not a guarantee
 - > GC will adjust parameters to try and meet goal
 - > Can adversely effect applicaiton throughput
- Throughput goal
 - > `-XX:GCTimeRatio=<n>`
 - > Percentage GC Time = $1 / (1 + n)$
 - > e.g. `-XX:GCTimeRatio=19` (5% of time in GC)

Main changes - **Performance** - Monitoring - Compatability - Java.next

Solaris Sparc



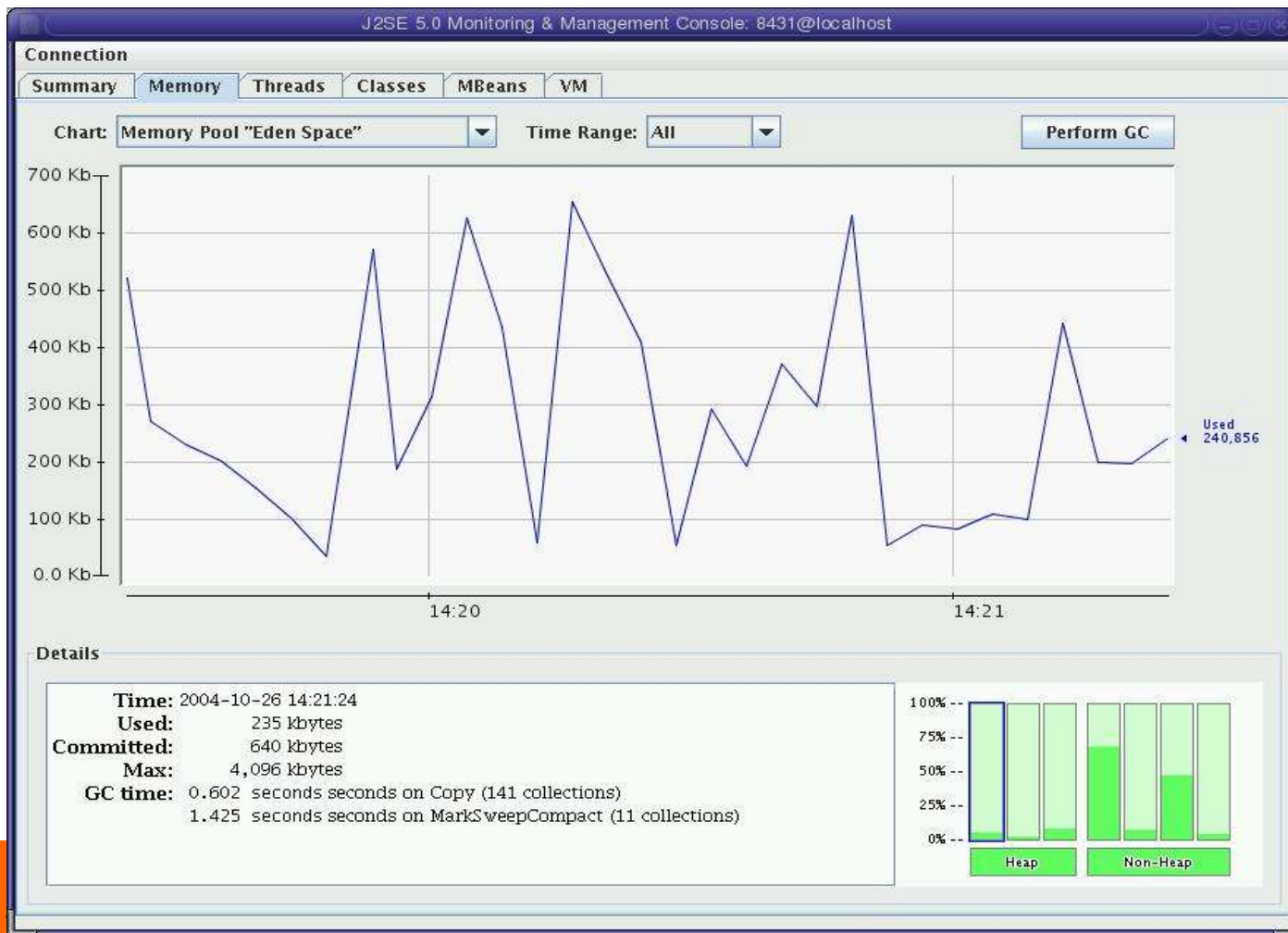
Main changes - Performance - **Monitoring** - Compatability - Java.next

- jinfo
 - > Config info from running JVM or crash dump
- jmap
 - > Lots of useful memory information on running JVM or crash dump
- jstack
 - > Prints stack traces for all threads in running VM or stack trace
- These are not supported on Windows

Main changes - Performance - **Monitoring** - Compatability - Java.next

- jstat
 - > Statistics of GC, dynamic compilation, class loader
 - > Not available on Windows 98, ME
 - > Not available on Windows NT, 2000, XP if using FAT32 filesystem
- jps
 - > Process ids of all instrumented HotSpot VMs

Main changes - Performance - **Monitoring** - Compatability - Java.next



Main changes - Performance – Monitoring - **Compatability** - Java.next

- J2SE 5.0 classes will not run on earlier JVMs
- **enum** is now a keyword
 - > Cannot be used as a variable identifier
- java.net.Proxy
 - > Can clash with java.lang.reflect.Proxy
- JAXP minor differences
 - > DOM Level 3, Xerces rather than Crimson
- For full details see
 - > java.sun.com/j2se/1.5.0/compatability.html

4. Virtual machine

Main changes - Performance - Monitoring - Compatability - **Java.next**

- J2SE 6: Codename “Mustang”
- More community based development
 - > j2se.dev.java.net
 - > Reference implementation still created through JCP
- Source code released under Java Research License
 - > Designed for universities and researchers
 - > Simpler and more relaxed terms than SCSL
- Get involved!

5. Conclusions & resources

- Over one hundred new features
- Lots of power
- Take time to learn the details
- Use the new features wherever you can
- Upgrade to the latest JVM wherever possible

- www.jcp.org
 - > JSR-014 Generics
 - > JSR-166 Concurrency utilities
 - > JSR-175 Metadata facility
 - > JSR-201 Enums, Autoboxing, For loop, Static import
- java.sun.com/j2se
- java.sun.com/j2se/1.5.0/compatibility.html
- j2se.dev.java.net
- www.netbeans.org



J2SE™ 5.0: WATCH AND HEAR THE TIGER ROAR!

Matt Hosanee

matt.hosanee@sun.com