The new Java 1.5

- murphee (Werner Schuster)
- http://jroller.com/page/murphee

Overview

- Language features in 1.5
  - Short introduction
  - Use/Don't use?
- Standard API changes
  - Monitoring and Management
  - Concurrent Utils
  - Swing and AWT updates
  - Changes for Unicode 4.0
  - Instrumentation
  - ...
- Sun JVM Changes
  - Class Data Sharing
  - ...

New Language features

- Metadata (JSR 175)
- Generics (JSR 14)
- Autoboxing/Unboxing (JSR 201)
- Enhanced for Loop (JSR 201)
- Typesafe Enums (JSR 201)
- Static Import (JSR 201)
- Varargs (JSR 201)

Language features – General

- First languages changes since 1.1 (except for assert)
- Only compatible with JVMs >= 1.5
  - Reason: they use 1.5 standard lib methods
  - Always make sure to use the “-source” args
  - “-source 1.4” if you don't use new features
  - Retroweaver http://retroweaver.sourceforge.net/

Metadata - Intro

- Probably most important new feature (Sorry Generics... but it's true)
- Started out with Javadoc @param,...
- .NET/C# extended it
- XDoclet introduced Metadata as well
- Better solution than “Marker Interfaces”

Metadata – Syntax

- Annotations (java.lang.annotation)

Sample:

```
// Annotation
@Retention(SOURCE) @Target(METHOD)
public @interface Overrides { }

// Usage
@Overrides
public boolean equals(Foo that) { ... }
```

Metadata - Syntax

- Retention
  - SOURCE
  - CLASS
  - RUNTIME
- Target
- Member types
  - Primitives
  - String
  - Enum
  - Annotation types
  - Arrays of all the preceding types

Metadata – another Sample

```
// Declaration
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer()  default "[unassigned]";
}


// Sample usage
@RequestForEnhancement(
    id       = 2868724,
    synopsis = "Provide time-travel functionality",
    engineer = "Mr. Peabody"
)
```

Metadata – Use/Don't use?

- Tools Tools Tools – are necessary to add value
- Possible use in Aspect Oriented Programming (explicit markup for pointcuts?)
- Problems
  - Missing mindset in developers
  - No samples for good usage
  - Tools Tools Tools

Definitely: Use!

Generics

- Java Generics != C++ Templates
- Ancestor: GJ
- Reason
  - Better documentation
  - Increase type safety
  - Less typing (no explicit casting)

Generics - syntax

Simple Example:

```
class FooMap<K,V>{
   public void put(K key, V value){...}
   //... other stuff
}
```

```
// usage
Map<String, Date> mapper = new FooMap<String, Date>();
mapper.put("foo", new Date());
Date x = mapper.get("foo");
```

Generics - Advanced

- Constraints
  - Not explicitely available in C++
  - Wildcards
  - Necessary because of strict typing
- Problems
  - No: T[] x = new T[10];
  - Basically just removes need for explicit casting
- http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf

Generics – Use/Don't use?

- Enhances documentation if used
  - `public Map<String, Date> getDates()`
    better than
    `public Map getDates()`
- Sun made sure it integrates with legacy software

Definitely: Use

Autoboxing/-Unboxing

- Java has
  - Primitives
  - Reference types (classes)
- Non-unified type model
  - Reason: speed tradeoff
- Problem: use of primitives in Collection classes
- Current solution: explicitely wrapping primitives

Autoboxing – Explicit Wrapping

```
List intList = new ArrayList();
for(int i=0; i<someListLength; i++){
   // Wrapping
     intList.add(new Integer(i));
}


// Un-Wrapping - yikes!
int foo = ((Integer)intList.get(0)).intValue();
```

// Disclaimer: this is just supposed to show wrapping,
// normally code like that should *not* use a
// Collection, but an array!

Autoboxing/AutoUnBoxing

```
List<Integer> intList = new ArrayList<Integer>();
for(int i=0; i<someListLength; i++){
   // AutoBoxing
     intList.add(i);
}


// AutoUnBoxing and Generics at work...
int foo = intList.get(0);
```

// Disclaimer: this is just supposed to show wrapping,
// normally code like that should *not* use a
// Collection, but an array!

AutoBoxing - Performance?

- AutoBoxing make Java easier to use
- Problem
  - It “hides” object creation from the developers mind
- Actually, AutoBoxing is not that bad
  - Values from -127 to 128 (byte, short, int) are *cached*
  - Running this code only results in object creation *the first time it's run:*

```
for(int i=0; i<20;i++){
  list.add(i);
}
```

  - Autoboxing does not call `new` for, but `valueOf()` of Wrapper classes

Auto(Un)Boxing – Use/Don't Use?

- Caching makes it less memory intensive
- Makes code clearer
- Future versions may be even more efficient (better than explicitely calling `new`)

Use (but with caution)

Enums

- Long awaited feature
- Improve type safety
  - No more integer constants as enumeration replacement
- Real classes
- Support to be used in `switch`
- Runtime output of enum name with `toString`

Enum - Sample

Simple:

```
public enum Season { WINTER, SPRING, SUMMER, FALL }

Season mySeason = WINTER;

public static void main(String[] args) {
     for (Season s : Season.values())
        System.out.println(s);
}
```

Prints:

```
WINTER
SPRING
SUMMER
FALL
```

Enum – Another Sample

More elaborate sample (from JSR-201 text):

```
public enum Coin {
    PENNY(1), NICKEL(5), DIME(10), QUARTER(25);

    Coin(int value) { this.value = value; }

    private final int value;

    public int value() { return value; }
}
```

Enum – Use/Don't Use?

Definitely: Use!

New for loop

- Makes code more readable
- Uses java.lang.Iterable
- Works on arrays as well

```
List<String> myList = foo.getList();
for(String currElement : myList){
    // do something with currElement
}
```

Definitely: Use!

Varargs - Overview

- Syntax sugar for `new[]{a,b,c}`
- No overhead
- Implemented for `printf()`?
- Other classes/methods use it too
- `public void foo(Object...)`

Definitely: Use!

Static Imports

- Some developers don't like typing

```
import static java.lang.System.*;
  ...
public static void main(String args){
  out.printf(“My foo is no %s”, “bar”);
}
```

Static Imports

EEEEEEEEEK!

Static Imports – Don't Use

- Throw out *DOS Edit* and *edlin* and use a more modern way to write code
- Tip: add the following to your Coding standard:

"Using static imports will be punished with 25 whip lashes!"

Do not use static imports!

Monitoring/Management

- JMX (Java Management Extension)
  - Useful for representing settings at runtime
  - Various consoles available
  - Export *Mbeans from VM using RMI
- JVM monitoring using JMX
  - `java.lang.management` holds Mbeans with JVM information (Memory, Classloading, GC,...)
- ManagementBeans:
  - Easy to use
  - Can be made as flexible as possible
  - http://java.sun.com/j2se/1.5.0/docs/guide/jmx/overview/JMXoverviewTOC.html

Concurrency Utils

- `Synchronize` and `wait/notify` are not enough
- Written by Doug Lea
- java.util.concurrent
  - Executors (configurable Threadpools)
  - Queues
  - Synchronizers (Semaphores,...)
  - Concurrent Collections (not governed by a single lock like Hashtable, Vector,...)
- java.util.concurrent.locks
  - Locks
  - Condition Variables
- java.util.concurrent.atomic
  - Threadsafe, lockfree access to single vars

Bytecode Instrumentation

- java.lang.instrument
- Modifying a class file
  - When the class is loaded
  - While the class is being used
- Sun Research: JFluid VM
- Possible before 1.5, but no Java API available
- Part of JVM Tools Interface
  - Replacement of JVMDI and JVMPI)
- What is it for:
  - Debugging (change code)
  - Profiling (add and profiling code from classes)

AWT - Changes

- AWT on X now implemented using Xlib (not Motif)
- Java2D can now use OpenGL to improve render speed
  - On Windows this has been done since 1.4.x using DirectX
- http://java.sun.com/j2se/1.5.0/docs/guide/2d/new_features.html

Swing - Look&Feel changes

- Windows XP
- GTK
- New default theme for Metal (“Ocean”)
  ...now with more Gradients
- Synth – new skinning L&F
  - declaration with XML files
  - extensible with Java code as well
  - Problem: little documentation available - yet

Unicode 4.0

- 2^16 are not characters not enough
- Supplementary characters
- “Old” 65536 characters are now “BMP”
  - Basic Multilingual Plane
- What has to change?
  - Nothing - if you use high level classes (String, CharSequence,...)
  - The length of a `char[]` != number of characters
  - Low Level APIs now use `int`s to represent characters (2 surrogate `char`s = one `int`)
- http://java.sun.com/developer/technicalArticles/Intl/Supplementary/

Various Stuff...

- Arbitrary Precision Math
  - In BigDecimal
- Pack200 Jar Compression
  - Exploits characteristics of Jar files
  - Much better compression
- printf()
  - In java.io.PrintStream
  - Convenience method for Formatter

Sun JVM – Class Data Sharing

- Long requested feature
  - Problem: same classes loaded for each JVM instance
- When the first JVM is launched, it
  - Loads the rt.jar classes
  - Creates a file containing the loaded classes
- Any new JVM simply maps this file into memory
  - Reduced startup time
- Only system classes – application classes sharing in some future release

Garbage Collector

- Lots of GC algorithms and combinations available
- Optimization through twiddling with parameters
- GC can now consider these goals
  - A desired maximum GC pause
  - A desired application throughput goal
  - Minimum throughput

- http://java.sun.com/j2se/1.5.0/docs/guide/vm/gc-ergonomics.html

JVM - Misc

- AMD64 support
- Java Memory Model fixed (JSR-133)
  - Arcane stuff... read the article
  - http://www-106.ibm.com/developerworks/library/j-jtp02244.html

Future of the Java platform

Put the Standard library under some open license?
Murphee's opinion:

- http://jroller.com/page/murphee/20040225
- http://jroller.com/page/murphee/20040305
- http://jroller.com/page/murphee/20040426