

JDK 5.0 Overview

Alexis Moussine-Pouchkine
Client Services
Sun Microsystems France

Jean-Christophe Collet
Staff Engineer
Java Security & Networking
Sun Microsystems



Agenda

- Overview
- New Language features
- Core Libraries
- VM Improvements
- Conclusion
- Q & A

Overview of 5.0

- Developed using the JCP
- 15 component JSRs for major new features
- Over 100 Features
- Lots of bug fixing and performance work
- Available since late September 2004
- Themes:
 - Quality (compatibility!)
 - Performance & scalability
 - Ease of Development
 - Monitoring & Manageability
 - Desktop Client

New Language Features

- Generics
- Covariant Return Types
- Enhanced **for** loop
- Enumerated Types
- Autoboxing (and unboxing)
- Varargs
- Static imports
- Annotations (metadata)
- Concurrency & Threading Utilities

Generics*

- When you get an element from a collection, you have to cast
 - Casting is a pain
 - Casting is unsafe — casts may fail at runtime
- Wouldn't it be nice if you could tell the compiler what type a collection holds?
 - Compiler could put in the casts for you
 - They'd be guaranteed to succeed

***: compile-time type safety**

Filtering a Collection — Today

```
// Removes 4-letter words from c
// elements must be strings
static void expurgate(Collection c) {
    for (Iterator i = c.iterator(); i.hasNext(); )
        if(((String) i.next()).length() == 4)
            i.remove();
}

// Alternative form - a bit prettier?
static void expurgate(Collection c) {
    for (Iterator i = c.iterator(); i.hasNext(); ) {
        String s = (String) i.next();
        if(s.length() == 4)
            i.remove();
    }
}
```

Collection With Generics

```
// Removes 4-letter words from c
static void expurgate(Collection<String> c) {
    for (Iterator<String> i = c.iterator(); i.hasNext(); )
        if (i.next().length() == 4)
            i.remove();
}
```

- Clearer and Safer
- No cast, no extra parentheses, no temporary variables
- Provides compile-time type checking
- Once compiled, typing information is erased

Enhanced for loop

- Simplified iteration over collections
- Applying a method to each element in a Collection Today :

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

Collection With Enhanced for

```
void cancelAll(Collection c) {  
    for (Object o : c)  
        ((TimerTask)o).cancel();  
}
```

- Clearer and Safer
- No iterator-related clutter
- No possibility of using the wrong iterator
- No access to the loop variable

Combined With Generics

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

- Much shorter, clearer and safer
- Code says exactly what it does

It Works for Arrays, Too

```
// Returns the sum of the elements of an array
int sum ( int[] a ) {
    int result = 0;
    for (int val : a)
        result += val;
    return result;
}
```

- Eliminates array index rather than iterator
- Similar advantages

Enumerated Types

Standard approach – **int** enum pattern

```
public class Almanac {  
    public static final int SEASON_WINTER = 0;  
    public static final int SEASON_SPRING = 1;  
    public static final int SEASON_SUMMER = 2;  
    public static final int SEASON_FALL = 3;  
  
    ... // Remainder omitted  
}
```

Enum Construct

- Compiler support for Typesafe Enum pattern
- Looks like traditional enum (C, C++, Pascal)

```
enum Season { winter, spring, summer, fall }
```
- But more powerful
 - All advantages of (pre-Tiger) Typesafe Enum pattern*
 - Allows programmer to add arbitrary methods, fields
- Can be used in **switch** statements

*: See Bloch's "Effective Java"

With Generics and enhanced for

```
enum Suit {clubs, diamonds, hearts, spades}
enum Rank {deuce, three, four, five, six, seven,
           eight, nine, ten, jack, queen, king,
           ace}
```

```
List<Card> deck = new ArrayList<Card>();
for (Suit suit : Suit.VALUES)
    for (Rank rank : Rank.VALUES)
        deck.add(new Card(suit, rank));
```

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

Would require pages of code today!

Autoboxing/Unboxing

- You can't put an `int` into a collection
 - Must use `Integer` instead
- It's a pain to convert back and forth
- Wouldn't it be nice if compiler did it for you?

Autoboxing/Unboxing (2)

- Before

```
list.add(0, new Integer(42));  
int total = (list.get(0)).intValue();  
Integer i = new Integer (36);  
i = new Integer ( i.intValue()++ );
```

- After

```
list.add(0,42);  
int total = list.get(0);  
Integer i = new Integer (36);  
i++;
```

- Operations work on wrapper types

- Watch performance (based on unboxing)
- Be careful with equality (JVM instance recycling)

Annotations (metadata)

- Key new Java programming language idea
- Modifiers to decorate packages, classes, methods, and fields for simpler declarative programming
- Language lets you annotate code so tools can generate boilerplate and side files.
- Can be retained in class file and also be visible at runtime
- Designed for use by tools and reflection
 - `apt` tool part of JDK

Annotations

- Simple Use

```
@Debug public void dumpState() { ... }
```

- Predefined annotations

```
@Deprecated public void foo() { ... }
```

```
@Override public void toString() { ... }
```

```
@SuppressWarnings({"unchecked",  
    "fallthrough"})
```

- Custom declared using `@interface`

```
@Retention(RUNTIME) @interface Debug {  
    boolean devbuild() default false;  
}
```

- Meta-annotations

- Target, Retention, Documented, and Inherited

APT Tool

- Command-line utility for annotation processing
 - First, runs annotation processors that can produce new source code and other files
 - Second, can cause compilation of both original and generated source files
- Compared to using a doclet to generate the derived files based on annotations, apt
 - uses a more contemporary API design, such as returning generic collections instead of arrays
 - supports recursive processing of newly generated files and can automatically cause compilation of all files
 - was built to process JDK 5.0 annotations

Annotation & Reflection

- `java.lang.reflect.AnnotatedElement` interface is implemented by all program element (Class, Constructor, Field, ...)

```
public boolean isAnnotationPresent ( Class  
    annotationType );
```

```
public Annotation getAnnotation ( Class  
    annotationType );
```

```
Class c = MyClass.class  
AnnotatedElement el = c.getMethod  
    ("dumpState");  
Debug debug = el.getAnnotation(Debug.class);  
boolean b = debug.devbuild();
```

Java Web Services

Tutorial Example: JAX-RPC Server

Before

```
public interface HelloIF extends Remote {
    public String sayHello(String s) throws
        RemoteException;
}
```

```
public class HelloImpl implements HelloIF {
    public String sayHello(String s) {
        return "Hello "+s;
    }
}
```

After

```
public class Hello {
    public @remote String sayHello(String s) {
        return "Hello "+s;
    }
}
```

New Language Features

- Generics
- Covariant Return Types
- Enhanced **for** loop
- Enumerated Types
- Autoboxing (and unboxing)
- Varargs
- Static imports
- Annotations (metadata)
- Concurrency & Threading Utilities

Some core libs new features

- Swing enhancements
 - Gnome L&F
 - Windows L&F
- XAWT toolkit with XDnD support
 - Default on Linux, option on Solaris
- OpenGL[®] acceleration

```
java -Dsun.java2d.opengl=true -jar Java2D.jar
```

- Other
 - Hyper-compression Pack200 support
 - Consolidated end-user consoles (Java Web Start and Plugin)
 - `out.printf("%s %5d\n", user, total);`

VM Improvements

- New 64 bits platform: AMD64
 - Windows, Linux, and Solaris 10 (soon)
- Faster startup time
 - Shared class data
- Hotspot tuning
 - “SmartTuning”
- Manageability
 - JMX Management API (JSR 3/160)
 - JVM monitoring & Management API (JSR 174)
- Error handling
 - **-XX:OnError="command"**

Performance in 5.0

Smart Tuning™

- Heuristics based on machine inspection
 - Server-class machine starts at 2GB/2CPU
 - Parallel GC, Server Hotspot VM compiler
 - Max Heap $\frac{1}{4}$ Max Memory or 1GB
 - Initial Heap $\frac{1}{64}$ Max Memory or Max Heap
 - 64bit => server VM / Windows => client VM
 - Mem sizes, Choice of collector, compiler, TLABs...
- Hyperthreading (HT) and NPTL support
 - Linux (if BIOS and kernel allow) and Solaris
- System class data sharing (OS-specific)
 - `java -Xshare:off` to disable
 - `java -Xshare:dump` to re-generate

Monitoring and Manageability

- JVM Monitoring and Management API (JSR-174)
 - Low memory detection
 - Allow access to internal VM status
 - Heap sizes, gc info, threads, etc.
 - Industry standard SNMP and JMX based
 - Plugs into existing management consoles
- JMX API (JSR-003)
 - MBeanServer part of JRE
 - Support for remote management JSR-160
 - Works with existing J2EE™ application servers

M&M Tools

- Monitoring (process, core or remote)
 - **jps** - list of all available running JVMs
 - **jstat** - monitors class compile, loading & GC stats
 - **jconsole** - monitors all of the above in a GUI
- Troubleshooting (process, core or remote)
 - **jinfo** - configuration information (vm version, compiler, options used, classpath, locale, flags, ...)
 - **jmap** - memory heap details (size, shared libs)
 - **jstack** - stack trace of threads (trace and state)
- All exposed as JMX MBeans, no extra instrumentation

*: for 1.4.x, use 'jvmstat' technology

JConsole SnapShots

J2SE 1.5 Monitoring & Management Console: localhost:0 (Monitoring Self)

Connection

Summary Memory **Threads** Classes MBeans VM

Number of Live Threads



Threads

Reference Handler	Name	State
Finalizer	AWT-EventQueue-0	WAITING
Signal Dispatcher		
AWT-Shutdown		
AWT-Window		
AWT-EventQueue-0		
DestroyJavaVM		
Java2D Disposer		
TimerQueue		
Worker-Memory-localhost:0		
Worker-MBeans-localhost:0		
Timer-localhost:0		
Worker-Threads-localhost:0		

J2SE 1.5 Monitoring & Management Console: localhost:0 (Monitoring Self)

Connection

Summary **Memory** Threads Classes MBeans VM

Chart: **Heap Memory Usage** Time Range: All Perform GC



Details

Time: 2004-08-29 14:49:21
 Used: 4862 kbytes
 Committed: 5368 kbytes
 Max: 65088 kbytes

J2SE 1.5 Monitoring & Management Console: localhost:0 (Monitoring Self)

Connection

Summary **Memory** Threads Classes MBeans VM

Summary

Uptime: 1 minute Process CPU time: 11.286 seconds
 Total compile time: 1.071 seconds

Threads

Live threads: 13 Peak: 16
 Daemon threads: 7 Total started: 18

Memory

Current heap size: 3838 kbytes Committed memory: 5368 kbytes
 Maximum heap size: 65088 kbytes
 Objects pending for finalization: 0
 Garbage collector: Name = 'MarkSweepCompact', Collections = 0, Total time spent = 0.000 seconds
 Garbage collector: Name = 'Copy', Collections = 143, Total time spent = 1.260 seconds

Classes

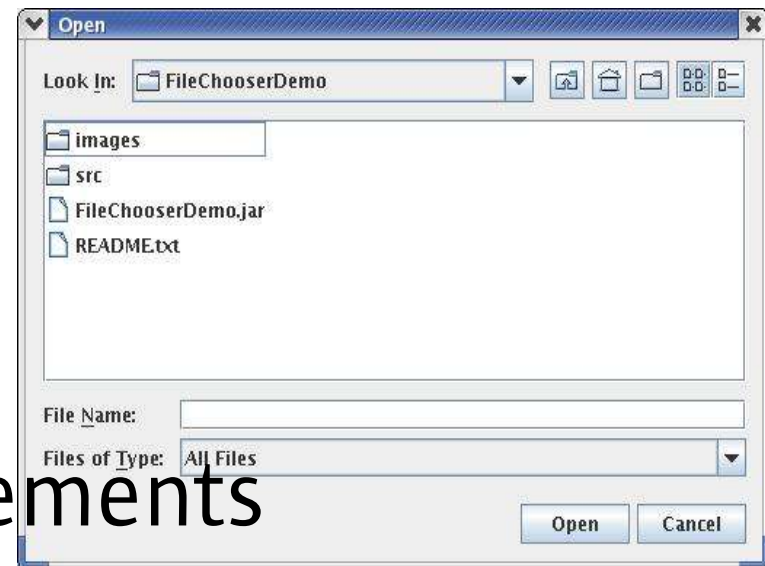
Current classes loaded: 2174 Total classes unloaded: 0
 Total classes loaded: 2174

Operating System

Total physical memory: 785200 kbytes Free physical memory: 372788 kbytes
 Committed virtual memory: 33344 kbytes

Desktop Client

- Improved GUI look-and-feel support
 - Improved Gnome skins support
 - New Synth skinnable API and XML config
 - Improved Windows XP look and feel
 - Fresh new cross-platform Java technology look and feel — “Ocean”
 - Printable JTable
 - Font rearchitecture
 - CUPS Printing Support
 - More...
- Java Web Start enhancements



Networking features

- HTTP
 - Connect & Read timeouts
 - Cookie Management
 - Caching support
 - Streaming output
- Proxy API
- Reachability API (ping)
- IPv6 support
- Misc

Read & Connect timeouts

- New methods in **URLConnection**:
 - `setConnectTimeout(int timeout)`
 - `int getConnectTimeout()`
 - `setReadTimeout(int timeout)`
 - `int getReadTimeout()`
- Works for HTTP, FTP, and any other appropriate protocol handler.
- **SocketTimeoutException** thrown if timeout expires.

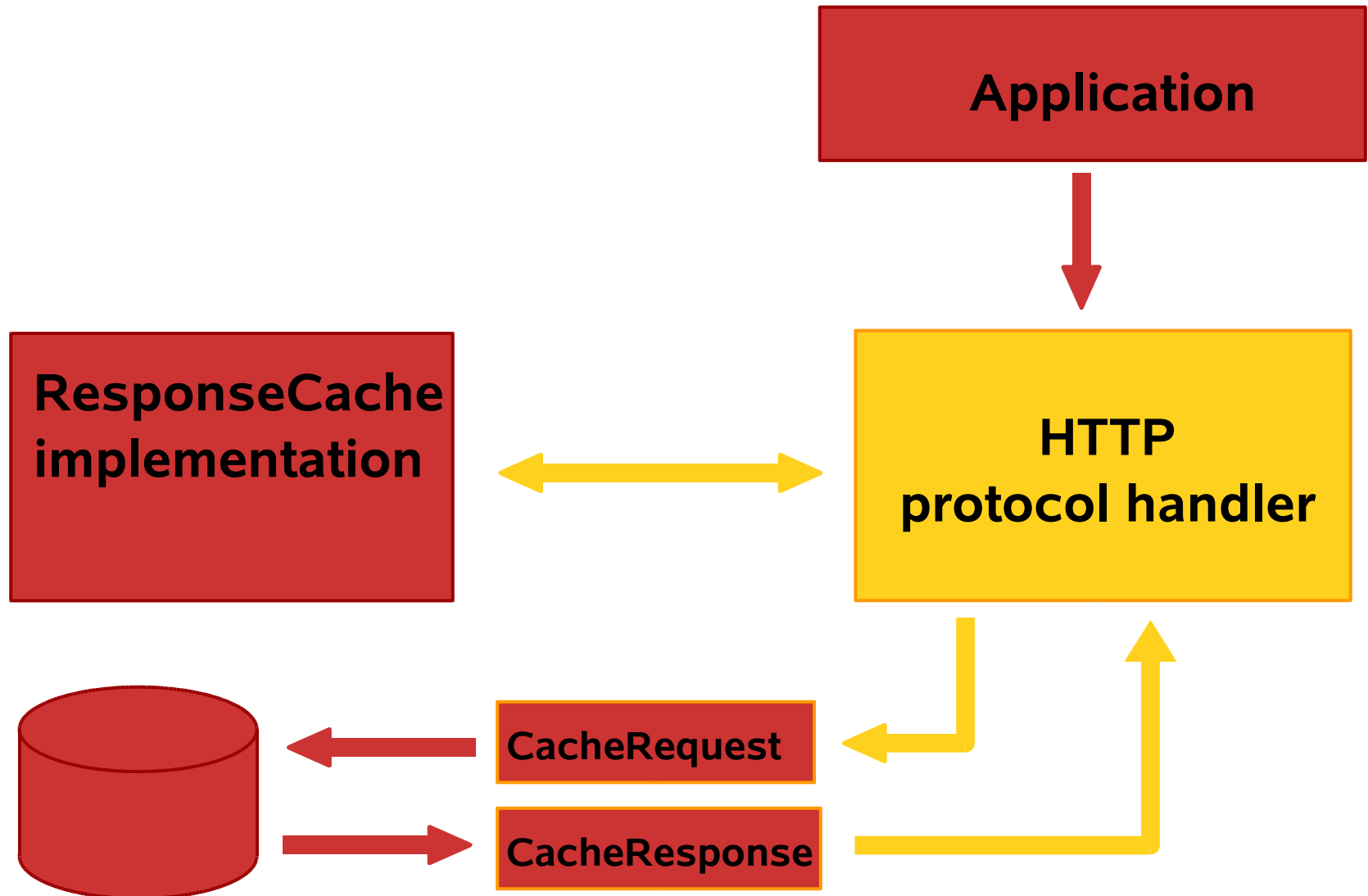
Cookie Management

- New `java.net.CookieHandler` class
- Cookies represented as:
`Map<String, List<String>>`
- Methods in `CookieHandler`:
 - `get` // returns a list of cookies (by URI)
 - `put` // stores a list of cookies
 - `getDefault` // get systems cookie handler
 - `setDefault` // set systems cookie handler
- No default handler in J2SE

URLConnection content caching

- Three new classes in java.net:
 - ResponseCache
 - CacheRequest
 - CacheResponse
- **ResponseCache** represents the cache
 - abstract class to be extended by a real impl.
 - `get()` //returns a CacheResponse based on URI and request headers
 - `put()` //allows cache to decide if resource should be cached, returns a CacheRequest
 - `setDefault()` `getDefault()` // set/get cache

URLConnection content caching



URLConnection content caching

- **CacheRequest** used to write an entry in cache
- Two methods:
 - `WritableByteChannel` `getBody()`
 - `void abort()`
- `getBody()` returns a channel for writing the request body into cache
- `abort()` abandons a cache write

URLConnection content caching

- **CacheResponse** returns an entry from cache
- Two methods:
 - `ReadableByteChannel getBody()`
 - `Map<String,List<String>> getHeaders()`
- `getBody()` returns a channel for reading the request body from cache.
- `getHeaders()` returns the headers stored

HTTP Streaming output

- Allows application to stream large POST data without any buffering.
- Trade-off is: redirection and authentication not automatic, if streaming enabled.
- 2 new methods in **HttpURLConnection**:
 - `setFixedLengthStreamingMode(int contentLength)`
 - `setChunkedStreamingMode(int chunkSize)`
- New **HttpRetryException**
 - `int responseCode()`
 - `String getLocation()`

Example (known size)

```
urlc.setRequestMetod ("POST");
URLConnection urlc = url.openConnection();
urlc.setDoOutput(true);
File f = new File ("file2send.txt");
urlc.setFixedLengthStreamingMode (f.length());
InputStream is = new FileInputStream (f);
OutputStream os = urlc.getOutputStream();
byte[] buffer = new byte [10 * 1024];
while ((l=is.read (buffer))>0) {
    os.write (buffer, 0, l);
}
os.close();
int response = urlc.getResponseCode();
```

Example (unknown size)

```
InputStream str ... // str provided externally
URLConnection urlc = url.openConnection();
urlc.setRequestMethod ("POST");
urlc.setDoOutput (true);
urlc.setChunkedStreamingMode (0); /* default
                                   chunk size */
OutputStream os = urlc.getOutputStream();
byte[] buffer = new byte [10 * 1024];
while ((l=str.read (buffer))>0) {
    os.write (buffer, 0, l);
}
os.close();
str.close();
int response = urlc.getResponseCode();
```

Proxy Management

- Until 1.4.x only properties:
 - `Dhttp.proxyHost=webcache.domain.com`
- New **Proxy** class allows for selection on a per connection basis
- `Proxy.Type` is an enum
 - {Direct, HTTP, SOCKS}
- Methods:
 - `SocketAddress address() // proxy server`
 - `Proxy.Type type() // Proxy type`

Proxy Management

- Proxy class usage:
 - `URL.openConnection(Proxy p)`
 - `Socket(Proxy)`

- Examples:

```
SocketAddress addr = new InetSocketAddress
    ("webcache.mydomain.com", 8080);
Proxy proxy = new Proxy(Proxy.Type.HTTP, addr);
URL url = new URL("http://java.sun.com/");
URLConnection conn = url.openConnection(proxy);
URL url2 = new URL("http://infos.mydomain.com/");
URLConnection conn2 = url2.openConnection
    (Proxy.NO_PROXY);
```

Proxy Management

- New **ProxySelector** class allows for automated proxy selection:
- Four methods:
 - `List<Proxy> select (URI)`
 - `void connectFailed (URI, SocketAddress, IOException)`
 - `getDefault()`, `setDefault()`
- Default implementations in J2SE, plugin

Reachability API

- New Method in **InetAddress**:
`boolean isReachable(int timeout)`
- Uses ICMP/Ping if possible
- If not, uses TCP connect to port 7
- Timeout expressed in milli-seconds

IPv6 enhancements

- Support for Windows XP SP1 / 2003
- Support for scoped addresses
- As always:
 - Automatic
 - Transparent
 - No code change needed

Misc networking enhancements

- Java applications from **inetd**
 - `java.nio.Channel`
 - `System.inheritedChannel()`
- Extension to Authenticator
- API for performance preferences
- Improved SOCKS support
- Many, many bug fixes

Mustang/Dolphin themes

- Compatibility, Stability, Quality
- Monitoring and Management
- Corporate/Enterprise Desktop
- Ease of Development
- XML and Web Services
- Becoming even more open

Conclusion

- JDK 5.0 is Big!
 - This was only an overview
 - Over 100 new features
- Biggest language change since 1.0
 - But only 1 new keyword
- Ease of development
- Performance & Scalability
- Compatibility
- Available now (<http://java.sun.com>)

JDK 5.0 Overview

Alexis.MP@Sun.COM

Jean-Christophe.Collet@Sun.COM

