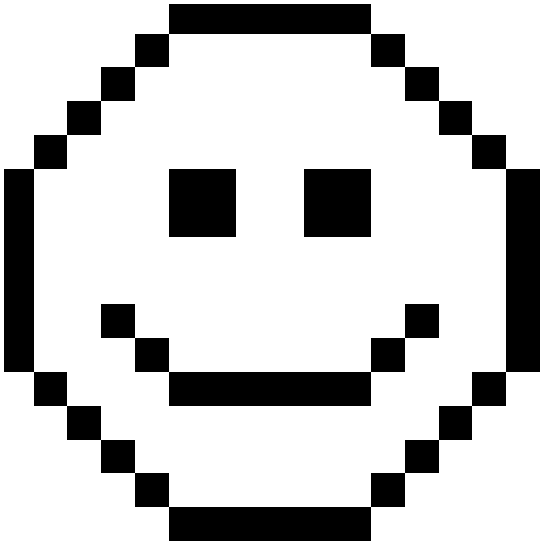
The 55 New Java 7 Features You (Probably) Didn't Hear About

- Slides from deck presented at EclipseCon Europe 2011, on November 2nd in Ludwigsburg, Germany.
- Please ping DonaldOJDK on Twitter with any comments.
- Many more details to most topics can be found at http://download.oracle.com/javase/7/docs/webnotes/adoptionGuide/index.html
- Note the SAO disclaimer at end of deck.

Binary Literals

```
int mask = 0b101010101010;
aShort = (short)0b1010000101000101;
long aLong = 0b1010000101000101101000010100010110100001010001011010000101000101L;
```

```
HAPPY_FACE = {
  (short)0b0000011111100000;
  (short)0b0000100000010000;
  (short)0b0001000000001000;
  (short)0b0010000000000100;
  (short)0b0100000000000010;
  (short)0b1000011001100001;
  (short)0b1000011001100001;
  (short)0b1000000000000001;
  (short)0b1000000000000001;
  (short)0b1001000000001001;
  (short)0b1000100000010001;
  (short)0b0100011111100010;
  (short)0b0010000000000100;
  (short)0b0001000000001000;
  (short)0b0000100000010000;
  (short)0b0000011111100000;}
```



Underscores in Numeric Literals

- Valid:

```
int mask = 0b1010_1010_1010;
long big = 9_223_783_036_967_937L;
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi =  3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BFFE;
```

- Invalid:

```
float pi1 = 3_.1415F;
long ssn = 999_99_9999_L;
int x1 = _52;
int x2 = 0_x52;
```

```
float pi2 = 3._1415F;

int x1 = 52_;
int x2 = 0x_52;
```

Strings in Switch Statements

```
int monthNameToDays(String s, int year) {
  switch(s) {
    case "April": case "June":
    case "September": case "November":
      return 30;

    case "January": case "March":
    case "May": case "July":
    case "August": case "December":
      return 31;

    case "February”:
      ...
    default:
      ...
}
```

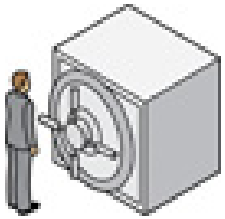
Did you know it produces generally more efficient byte codes than an if-then-else statement? Case Sensitive!

Automatic Resource Management

```
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);
}
```

- New superinterface **`java.lang.AutoCloseable`**
- All **`AutoCloseable`** (throws Exception) and by extension java.io.Closeable (throws IOException) types useable with try-with-resources
- Anything with a **`void close()`** method is a candidate
- JDBC 4.1 retrofitted as **`AutoCloseable`** too

Suppressed Exceptions



```
java.io.IOException
      at Suppress.write(Suppress.java:19)
      at Suppress.main(Suppress.java:8)
      Suppressed:  java.io.IOException
          at Suppress.close(Suppress.java:24)
          at Suppress.main(Suppress.java:9)
      Suppressed:  java.io.IOException
          at  Suppress.close(Suppress.java:24)
          at  Suppress.main(Suppress.java:9)

Throwable.getSupressed(); // Returns Throwable[]
Throwable.addSupressed(aThrowable);
```

Multi-Catch

```
try {
...
} catch (ClassCastException e) {
  doSomethingClever(e);
  throw e;
} catch(InstantiationException |
    NoSuchMethodException |
    InvocationTargetException e) {
  // Useful if you do generic actions
  log(e);
  throw e;
}
```

More Precise Rethrow

```
public void foo(String bar)
 throws FirstException, SecondException {
   try {
     // Code that may throw both
     // FirstException and SecondException
   }
   catch (Exception e) {
     throw e;
   }
 }
```

- Prior to Java 7, this code would not compile, the types in throws would have to match the types in catch – foo would have to "throws Exception"
- Java 7 adds support for this as long as try block calls all the exceptions in the throws clause, that the variable in the catch clause is the variable that is rethrown and the exceptions are not caught by another catch block.

Diamond Operator works many ways…

With diamond (**<>**) compiler infers type…

```
List<String> strList = new ArrayList<>();

OR

List<Map<String, List<String>> strList =
  new ArrayList<>();

OR

Foo<Bar> foo = new Foo<>();
foo.mergeFoo(new Foo<>());
```

Varargs Warnings – Erasure

```
class Test {
   public static void main(String... args) {
      List<List<String>> monthsInTwoLanguages =
        Arrays.asList(Arrays.asList("January",
                                    "February"),
                      Arrays.asList("Gennaio",
                                    "Febbraio" ));
   }
}
```

```
Test.java:7:  warning:
[unchecked]  unchecked  generic  array  creation
for  varargs parameter  of  type  List<String>[]
         Arrays.asList(Arrays.asList("January",
                               ^
1  warning
```

```
@SuppressWarnings(value = “unchecked”) // at call
@SafeVarargs   // at declaration
```

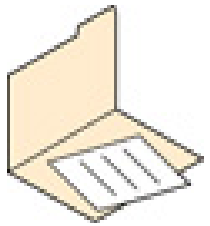
Java NIO.2 – File Navigation Helpers Slide 1 of 2

Two key navigation Helper Types:

- Class java.nio.file.Paths
  - Exclusively static methods to return a Path by converting a string or Uniform Resource Identifier (URI)
- Interface java.nio.file.Path
  - Used for objects that represent the location of a file in a file system, typically system dependent.

Typical use case:

Use Paths to get a Path. Use Files to do stuff.



Java NIO.2 – File Navigation Helpers Slide 2 of 2

```
//Make a reference to a File
Path src = Paths.get(“/home/fred/readme.txt”);
Path dst = Paths.get(“/home/fred/copy_readme.txt”);

//Make a reference to a path
Path src = Paths.get(“/home/fredSRC/”);
Path dst = Paths.get(“/home/fredDST/”);


//Navigation /home/fredSRC -> /home/fredSRC/tmp
Path tmpPath = src.resolve(“tmp”);

//Create a relative path from src -> ..
Path relativePath = tmpPath.relativize(src);

// Convert to old File Format for your legacy apps
File file = aPathPath.toFile();
```

Java NIO.2 Features – Files Helper Class

- Class java.nio.file.Files
  - Exclusively static methods to operate on files, directories and other types of files
- Files helper class is feature rich:
  - Copy
  - Create Directories
  - Create Files
  - Create Links
  - Use of system “temp” directory
  - Delete
  - Attributes – Modified/Owner/Permissions/Size, etc.
  - Read/Write

```
Files.move(src, dst, StandardCopyOption.ATOMIC_MOVE);
Files.copy(src, dst,
     StandardCopyOption.COPY_ATTRIBUTES,
     StandardCopyOption.REPLACE_EXISTING);
```

Java NIO.2 Directories

- DirectoryStream iterate over entries
  - Scales to large directories
  - Uses less resources
  - Smooth out response time for remote file systems
  - Implements **`Iterable`** and **`Closeable`** for productivity
- Filtering support
  - Build-in support for glob, regex and custom filters

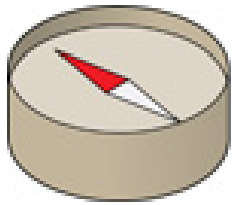
```
Path srcPath = Paths.get("/home/fred/src");

try (DirectoryStream<Path> dir =
   srcPath.newDirectoryStream("*.java")) {
for (Path file : dir)
   System.out.println(file.getName());
}
```

Java NIO.2 Symbolic Links

Path and Files are “link aware”

```
Path newLink = Paths.get(. . .);
Path existingFile = Paths.get(. . .);
try {
    Files.createLink(newLink, existingFile);
} catch (IOException x) {
    System.err.println(x);
} catch (UnsupportedOperationException x) {
     //Some file systems or some configurations
     //may not support links
    System.err.println(x);
}
```



Java NIO.2 Walking A File Tree

- A FileVisitor interface makes walking a file tree for search, or performing actions, trivial.
- SimpleFileVisitor implements

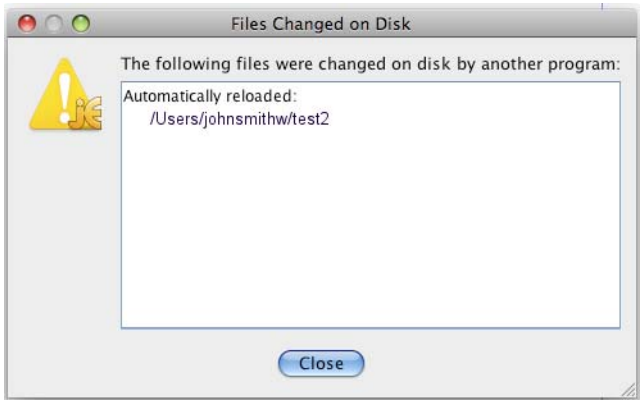
```
preVisitDirectory(T dir, BasicFileAttributes attrs);
visitFile(T dir, BasicFileAttributes attrs);
visitFileFailed(T dir, IOException exc);
postVisitDirectory(T dir, IOException exc);
```

SAMPLE:

```
Path startingDir = ...;
PrintFiles pf = new PrintFiles(); // SimpleFileVisitor sub
  // visitFile(Path p, BasicFileAttributes bfa) {
  //       System.out.println(file.getFileName());}
Files.walkFileTree(startingDir, pf);
```

Java NIO.2 Watching A Directory

- Create a WatchService “watcher” for the filesystem
- Register a directory with the watcher
- “Watcher” can be polled or waited on for events
  - Events raised in the form of Keys
  - Retrieve the Key from the Watcher
  - Key has filename and events within it for create/delete/modify
- Ability to detect event overflows



NIO.2 Custom FileSystems

- FileSystems class is factory to FileSystem (interface)
- Java 7 allows for developing custom FileSystems, for example:
  - Memory based or zip file based systems
  - Fault tolerant distributed file systems
  - Replacing or supplementing the default file system provider
- Two steps:
  - Implement `java.nio.file.spi.FileSystemProvider`
    - URI, Caching, File Handling, etc.
  - Implement `java.nio.file.FileSystem`
    - Roots, RW access, file store, etc.

NIO.2 filesystem provider for zip/jar archives

A fully-functional and supported NIO.2 filesystem provider for zip and jar files

```
Map<String, String> env = new HashMap<>();
   env.put("create", "true");
   // locate file system by using the syntax
   // defined in java.net.JarURLConnection
   URI u= URI.create("jar:file:/foo/zipfs/zipfstest.zip");
   try (FileSystem z = FileSystems.newFileSystem(u, env)) {
    Path externalTxtFile = Paths.get("/foo/zipfs/Sample.txt");
    Path pathInZipfile = z.getPath("/Sample.txt");
   // copy a file into the zip file
   externalTxtFile.copyTo(pathInZipfile);
}
```

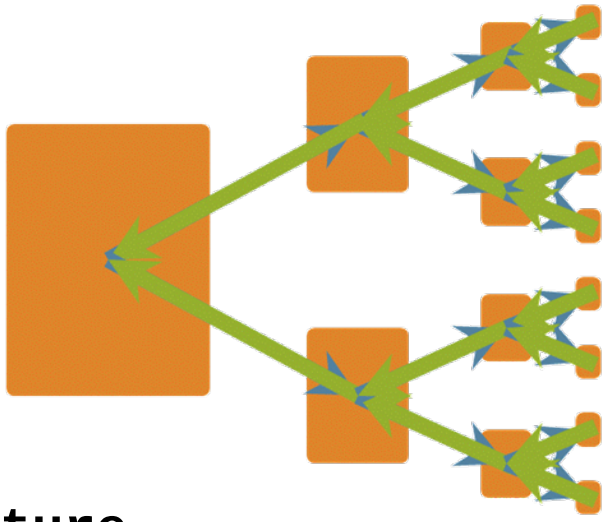
Concurrency APIs JSR 166y - Phasers

- **`Phaser`**
  - Barrier similar to **`CyclicBarrier`** and **`CountDownLatch`**
  - Used for many threads to wait at common barrier point
    - For example, use this to create N threads that you want to do something simultaneously – “start gun” metaphore
  - How is Phaser an improvement?
    - Dynamic add/remove “parties” to be sync’d
    - Better deadlock avoidance
    - Arrival “counting” and phase advance options, etc
    - Termination api’s
    - Tiering (tree structure)
      - Rather than sync 100 threads, sync 2x50 then 2x.

Concurrency APIs JSR 166y - TransferQueue

- **`TransferQueue`** interface
  - Extension to **`BlockingQueue`**
  - Implemented by **`LinkedTransferQueue`**
  - Additional Benefits:
    - Adds methods:
      - `transfer(E e), tryTransfer(E e), tryTransfer(E e, long timeout), hasWaitingConsumer(), getWaitingConsumerCount()`
    - Allows for smarter queues to be built – sidestep the data structure if it's known there are consumers waiting.

Fork Join Framework - JSR 166y - Pools

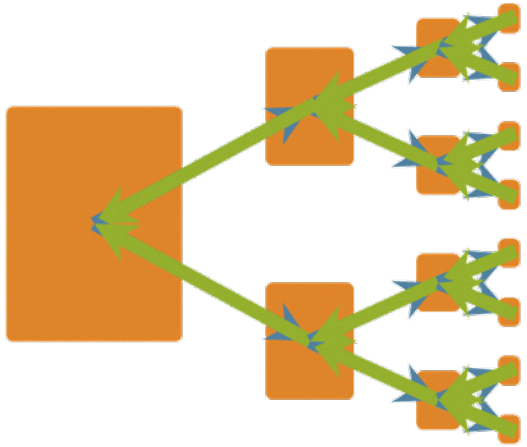


- **ForkJoinPool**
  - Service for running **ForkJoinTasks**
  - **aFjp.execute(aTask); // async**
  - **aFjp.invoke(aTask); // wait**
  - **aFjp.submit(aTask); // async + future**

  - **ForkJoinPool(); // default to platform**
  - **ForkJoinPool(int n); // # concurrent threads**
  - **ForJoinPool(n,aThreadFactory,exHandler,FIFOtasks); // Create your own thread handler, exception handler, and boolean on task ordering (default LIFO)**

Fork Join Framework - JSR 166y - Tasks

- **`ForkJoinTask`**
  - The abstract base class for:
    - **`RecursiveAction`**
      - A recursive resultless task
      - Implements **`compute()`** abstract method to perform calculation
    - **`RecursiveTask`**
      - Similar to **`RecursiveAction`** but returns a result



```
ForkJoinPool p = new ForkJoinPool();
    MyTask mt = new MyTask(n); // implements compute
    p.submit(mt);
while (!mt.isDone()) {/*THUMPER!*/ }
System.out.println(mt.get());
```

Fork Join Framework - JSR 166y - compute()

- **RecursiveAction** example to increment an entire array

```
protected void compute() {
    if (hi - lo < THRESHOLD) {
       for (int i = lo; i < hi; ++i) array[i]++; }
    else {
       int mid = (lo + hi) >>> 1;
       invokeAll(new IncrementTask(array, lo, mid),
                 new IncrementTask(array, mid, hi));}
```

- **RecursiveTask example for Fibonacci numbers**

```
protected Integer compute() {
    if (n <= 1) return n;
    Fibonacci f1 = new Fibonacci(n - 1);
    Fibonacci f2 = new Fibonacci(n - 2);
    f1.fork(); f1.fork();
    return f2.join() + f1.join();}
```

Concurrent Random Numbers - JSR 166y

- Existing RNG becomes unwitting source of contention between threads in concurrent apps
- Expected more needs of concurrent RNG with advent of Fork Join Framework

- **Class java.util.ThreadLocalRandom**
  - `ThreadLocalRandom.current().nextDouble(…)`
  - `ThreadLocalRandom.current().nextInt (…)`
  - `ThreadLocalRandom.current().nextLong(…)`

JSR 166y – ConcurrentLinkedDeque Class

- Unbound concurrent deque based on linked nodes
  - Like a Queue, but allows front and rear removal of elements
- Concurrent insert, remove and access on multiple threads
- Iterators are weakly consistent

ClassLoader Improvements – Deadlock Avoidance

Class Hierarchy:

class A extends B ; class C extends D ;

Custom Classloader CL1:

directly loads class A

delegates to custom ClassLoader CL2 for class B

Custom Classloader CL2:

directly loads class C

delegates to custom ClassLoader CL1 for class D

Thread 1:

Use CL1 to load class A (locks CL1)

defineClass A triggers loadClass B (try to lock CL2)

Thread 2:

Use CL2 to load class C (locks CL2)

defineClass C triggers loadClass D (try to lock CL1)

ClassLoaders were "not sufficiently granular" and prone to deadlock

Java 7 has a "parallel capable classloader"

(in SE 7 – lock CL1+A)

(in SE7 – lock CL2+B)

(in SE 7 – lock CL2+C)

(in SE7 – lock CL1+D)

URLClassLoader Improvements – close()

```
// create a class loader loading from "foo.jar"
URL url = new URL("file:foo.jar");
URLClassLoader loader = new URLClassLoader (new URL[] {url});
Class cl = Class.forName ("Foo", true, loader);
Runnable foo = (Runnable) cl.newInstance();
foo.run();
loader.close ();

// foo.jar gets updated somehow
loader = new URLClassLoader (new URL[] {url});
cl = Class.forName ("Foo", true, loader);
foo = (Runnable) cl.newInstance();

// run the new implementation of Foo
foo.run();
```

Unicode 4 -> Unicode 6.0

- Unicode standard was originally 16 bit
- 16 bits not sufficient for Unicode 6, but backward compatibility needs to be maintained
- Use String “U+hex” to express char in Unicode
- Unicode 6.0 adds thousands of new characters
- Support for properties and data files (mostly interesting to Japanese Telcos and Indic scripts)
- Full Unicode 6.0 REGEX support!

Character	Unicode Code Point	Glyph
Latin A	U+0041	A
Latin sharp S	U+00DF	ß
Han for East	U+6771	東
Deseret, LONG I	U+10400	𐐀

Extensible Currency Codes (ISO 4217)

- ISO 4217 Defines Currency Codes
- Possible to supersede default currencies with `<JAVA_HOME>/lib/currency.properties` file
- Allows for supporting global changes without updating Java
- Format: ISO 3166 Country code = ISO 4217 Codes

```
# Sample currency property if Canada adopts USD
# CA=CAD,124,2 is default ISO 4217 code
CA=USD,840,2
```

Number Shaper Enhancements

- NumericShaper used to map numbers to non Latin char sets (since 1.4)
- NumericShaper traditionally used an int bitmask for defaults
  - Fine when there were only 19 defaults
  - In Java 7 there are 34 (> 32 bits!!)
- Java 7 now has an `Enum NumericShaper.Range`
- Backward compatibility maintained, new API's added for Enum use where desired

Locale enhancement – Categories

- Default Locale can be set independently for *format* resources (dates, numbers, currencies) and *display* resources (menus and dialogs)
- For example, an application for Japanese speakers who deal with US financial transactions may:

```
//Enum Locale.Category – DISPLAY and FORMAT
//Default no arg get/set is DISPLAY
Locale.setDefault(DISPLAY, Locale.JAPAN);
Locale.setDefault(FORMAT, Locale.US);
```

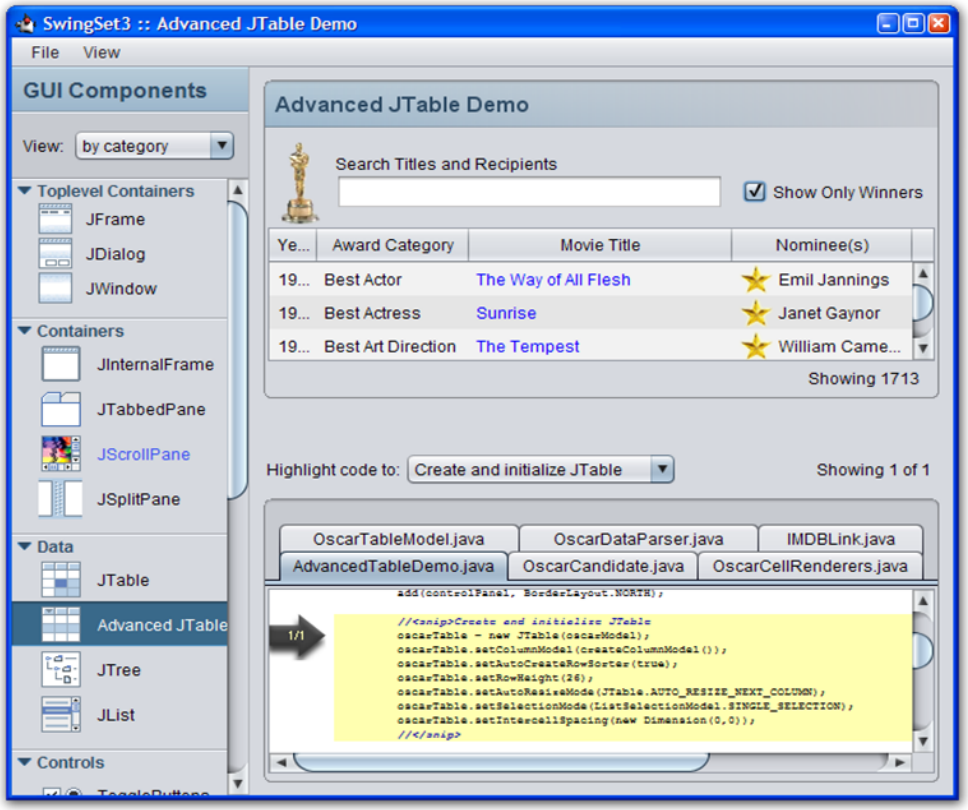
Locale enhancement – BCP 47 Extensions

- Java 7 confirms to IETF BCP 47 (refs UTS #35)
  - Specify extensions to a Locale (get/set)
  - i.e., de-DE-co-phonebk
  - No guarantee the underlying platform can honour extension

Key	Description	Example	Example Description
ca	calendar algorithm	ca-buddhist	Thai Buddhist calendar
co	collation type	co-pinyin	Pinyin ordering for Latin
k*	collation parameters	kf-upper	Donald before donald
cu	currency type	cu-usd	U.S. dollars
nu	number type	nu-jpanfin	Japanese financial numerals
tz	timezone	tz-aldav	Europe/Andorra
va	common variant type	va-posix	POSIX style locale variant

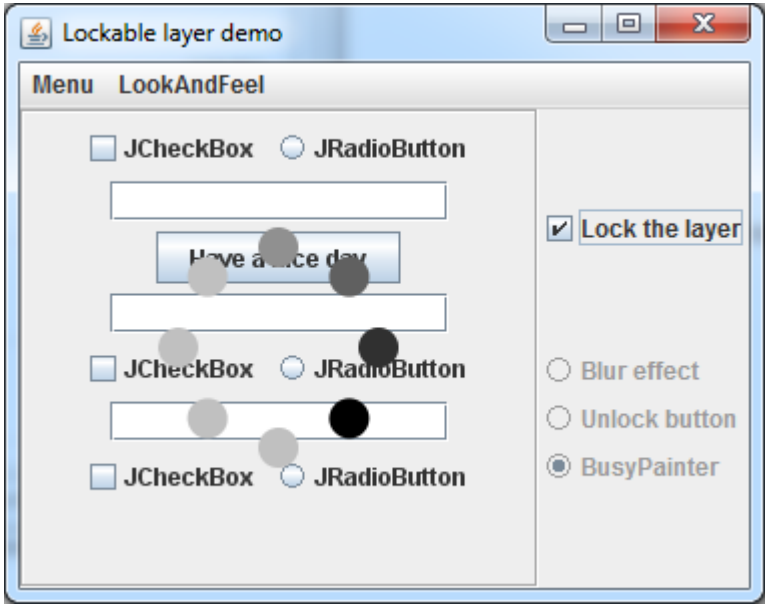
Standardize Nimbus Look and Feel

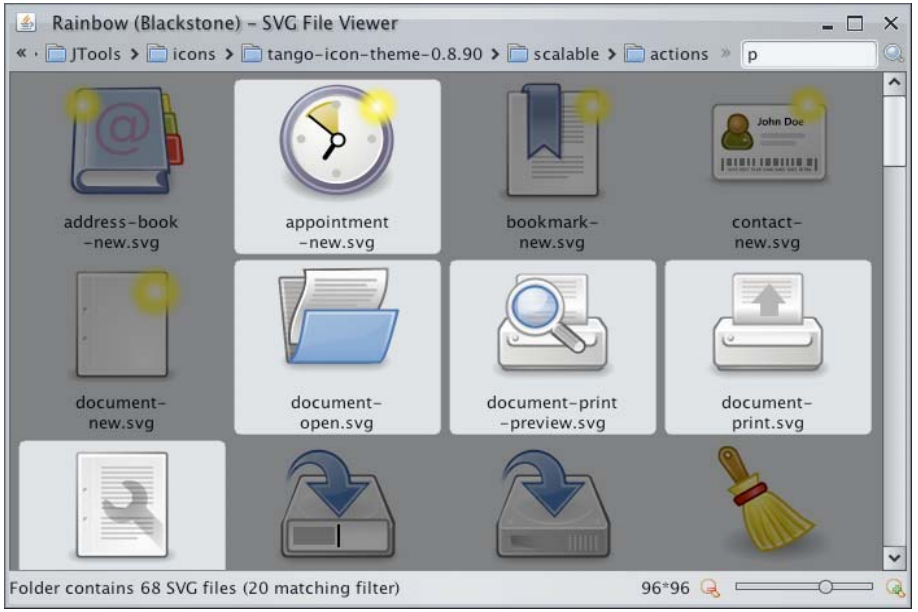
- Better than Metal for cross platform look-and-feel
- Introduced in Java SE 6u10, now part of Swing
- Not the default L&F
- Scalable Java 2D impl



Standardize JLayer Component

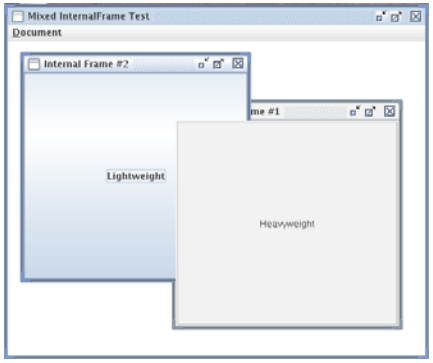
Easy enrichment for Swing components

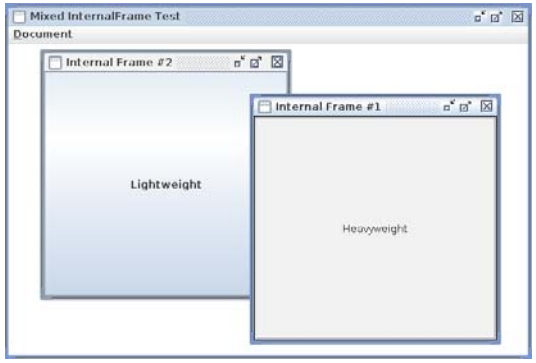





```
// wrap your component with JLayer
JLayer<JPanel> layer = new JLayer<JPanel>(panel);
// custom ui provides all extra functionality
layer.setUI(myLayerUI);
// add the layer as usual component
frame.add(layer);
```

Mixing of AWT and Swing – Works*

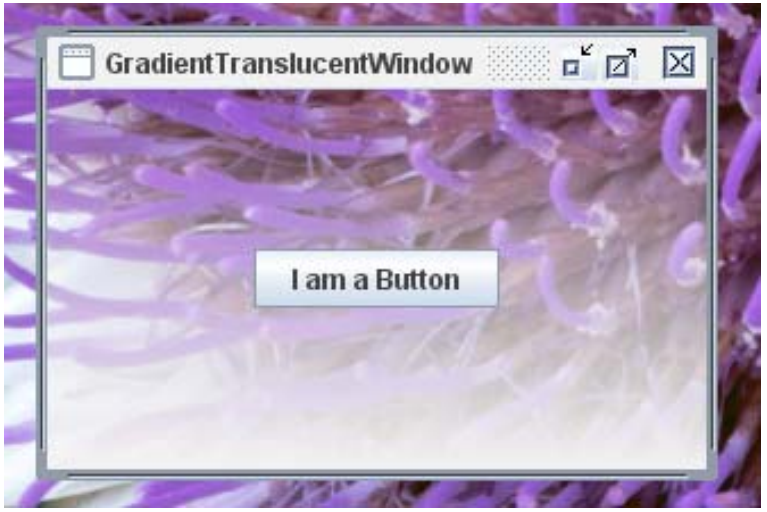




- As of 6u12 and 7u1, some caveats for scroll bars

Standardize Translucent Windows

- Private API added in 6u10, made public in Java 7
- Support (based on platform) for:
  - Uniform Translucency
  - Per Pixel Translucency
  - Per Pixel Transparency




```
// simple uniform:
aWindow.setOpacity(0.5f);
```

```
// Per pixel g2d is the g2d of a Jpanel on paintComponent(g)
Paint p = new GradientPaint(0.0f, 0.0f, new Color(R, G,
   B,0), 0.0f, getHeight(), new Color(R, G, B, 255), true);
g2d.setPaint(p);
g2d.fillRect(0, 0, getWidth(), getHeight());
```

Xrender-based Java 2D for modern X11

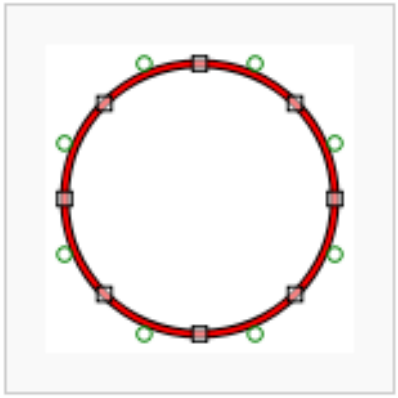
- Improved Graphics Performance
- Off by default (backward compatibility)

- Quiet:
  `-Dsun.java2d.xrender=true`

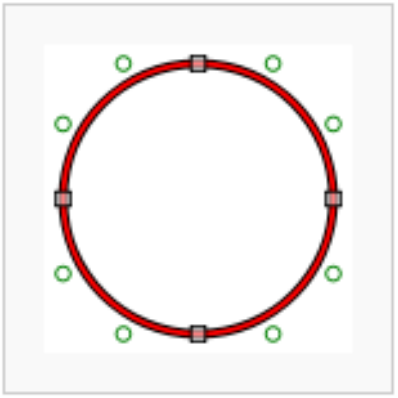
- Verbose (log on stdout if successful or not)
  `-Dsun.java2d.xrender=True`

OpenType/CFF Fonts

- Java Platform must support TrueType fonts, other font technologies is implementation dependent
- Java 7 adds support for “Compact Font Format” - OpenType/CFF.



TrueType outlines use quadratic Bézier splines.

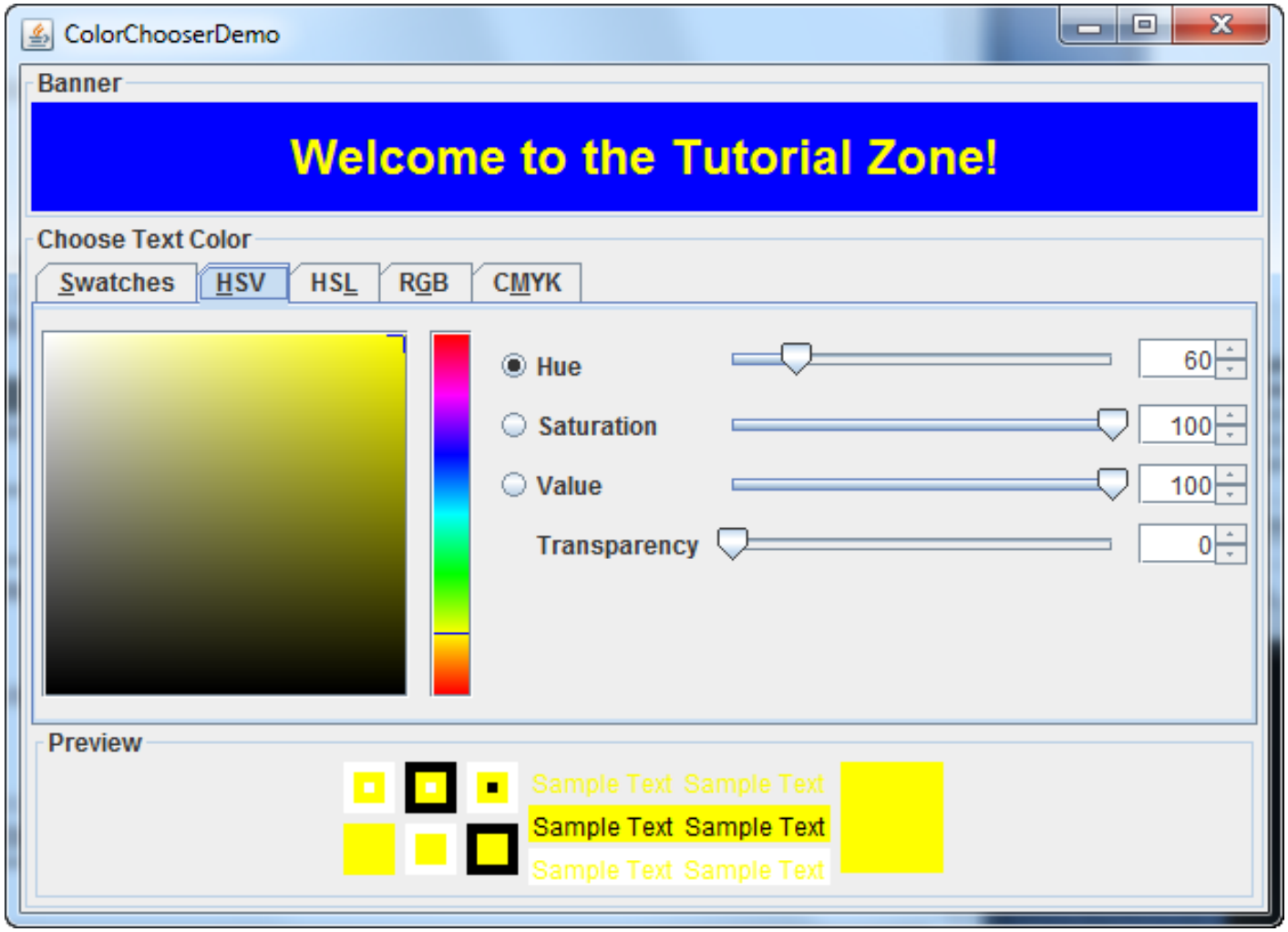


CFF outlines use cubic Bézier splines.

Better Support for Linux Fonts

- Five logical fonts since Java 1.0:
  - Serif, Sans-serif, Monospaced, Dialog, and DialogInput
  - Must map to physical font on your system
- No consistency in fonts in Linux
- Required editing fontconfig.properties
- Java 7 on Linux (and Solaris 11) uses system “libfontconfig”, reflecting what Gnome/KDE desktop applications use

HSV/HSL Tab on JColorChooser Class



JDBC 4.1

- Try-with-resources statement to automatically close resources of type `Connection`, `ResultSet`, and `Statement`

```
try (Statement stmt = con.createStatement()) { // ... }
```

- RowSet 1.1 introduces `RowSetFactory` and `RowSetProvider`

```
//Factory options (impl) set on cmd line or metainf
myRowSetFactory = RowSetProvider.newFactory();
jdbcRs = myRowSetFactory.createJdbcRowSet();
jdbcRs.setUrl("jdbc:myDriver:myAttribute"); //etc
jdbcRs.setCommand("select COF_NAME, SUP_ID, PRICE,
   SALES, TOTAL from COFFEES");
jdbcRs.execute();
```

Java DB Enhancements (Derby)

- JDK 7 includes Java DB 10.8.1.2
- New Since JDK 6
  - BOOLEAN data type
  - Table truncation
  - Query plan browsing
    - Automatic calc of index stats
  - Unicode database names
  - Improved interrupt handling
    - Can now interrupt connection threads
  - MAX optimization (faster!)
  - XML operator portability

Embedding JNLP File in Applet Tag

- Saves a network round trip first time applet is loaded
- Base64 Encode the JNLP contents into a Javascript call:

```
<script src="http://www.java.com/js/deployJava.js"></script>
<script>
   var attributes = {} ;
<!-- Base64 encoded string trunc'd below for readability -->
   var parameters = {jnlp_href: 'dynamictree-applet.jnlp',
    jnlp_embedded: 'PCEtLSAKLyoKICogQ29weX ... HA+Cg==' } ;
   deployJava.runApplet(attributes, parameters, '1.7');
</script>
```

Ability to detect applet init status on load

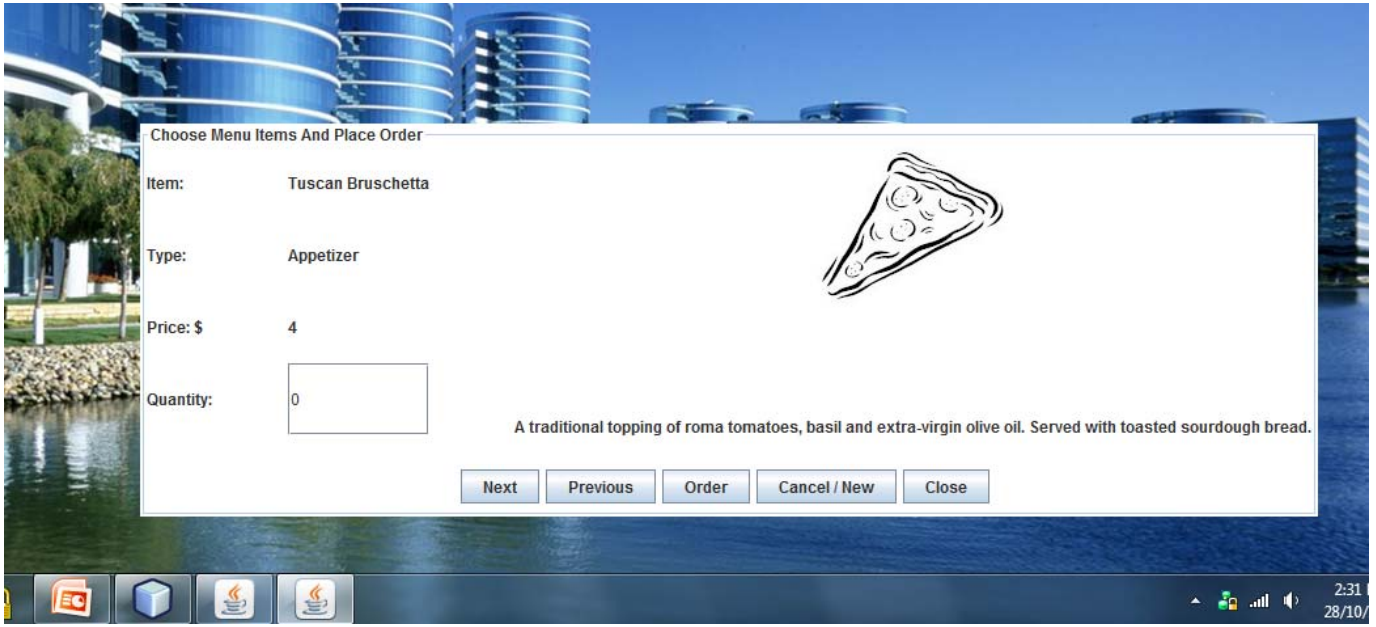
```
<script>
function registerAppletStateHandler() {
   switch (drawApplet.status) {
   case 1: <!-- applet is loading -->
    drawApplet.onLoad = onLoadHandler;
   case 2: <!-- applet is loaded -->
   case 3: <!-- error -->
    document.getElementById("mydiv")
        .innerHTML =“No need to onload";
   }
}
```

```
function onLoadHandler(){
   document.
   getElementById("mydiv“)
   .innerHTML =
   "Applet has loaded";
   draw();
}
```

```
<!-- assume java.com/js/deployJava.js is loaded ->
var parameters = {java_status_events: 'true'};
<!-- set other params like jnlp->
deployJava.runApplet(attributes, parameters, '1.7');
...
</script>
```

Draggable Applet Decoration

- Applet decoration settings apply equally to in browser and out of browser launches – borderless, etc.



Other Misc New JNLP Stuff...

- Partially signed JNLP
  - Simplifies build and deployment in some scenarios
  - External JNLP file may differ from one embedded in jar
- Targeting resources to particular version of OS

```
<resources os="Windows\ Vista Windows\ 7">
<jar href=“legacySound.jar"/> </resources>
```

- Better “install” preferences of an application
  - For example, attribute to determine if app appears on “Add or Remove Programs panel”

VM: Updates to Experimental GC – G1

- Garbage First - “G1” intended to replace* Concurrent Mark-Sweep (CMS) in Hotspot at some future release
- G1 is included for experimentation in Java 7
- Key benefits:
  - More predictably “soft real-time” – temporal configuration
  - High throughput
- Basics:
  - Heap partitioned into equal-sized heap regions
  - Compacts as it proceeds – looks for regions with no live objects for immediate reclamation

*not an official fwd looking statement

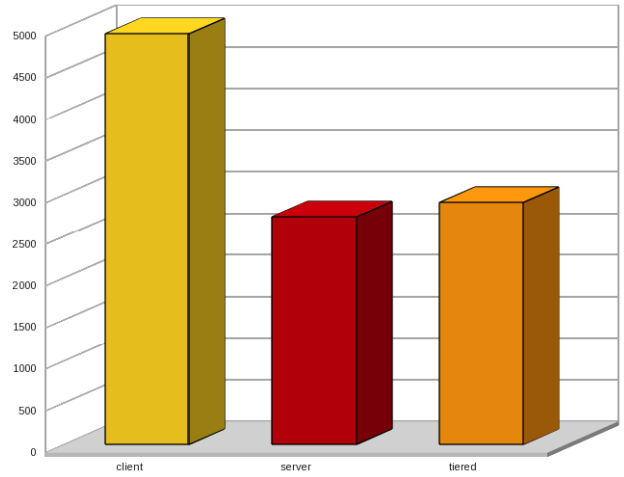
VM: Tiered Compilation

- Hotspot has 2 JIT's "client" and "server"
  - Client starts fast, but let optimizations – best for clients
  - Server starts slower, but provides better optimizations
- Java 7 adds Tiered Compiliation
  - JIT the code first with "client", and if it's really hot code, recompile with "server"
  - Has been around for a while, but not with a great implementation

```
-server -XX:+TieredCompilation
```

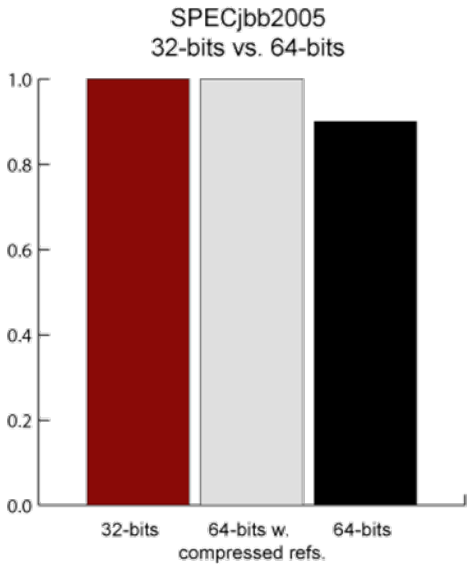
Image from Rémi Forax showing the DaCapo Jython benchmark.

http://www.java.net/blogs/forax



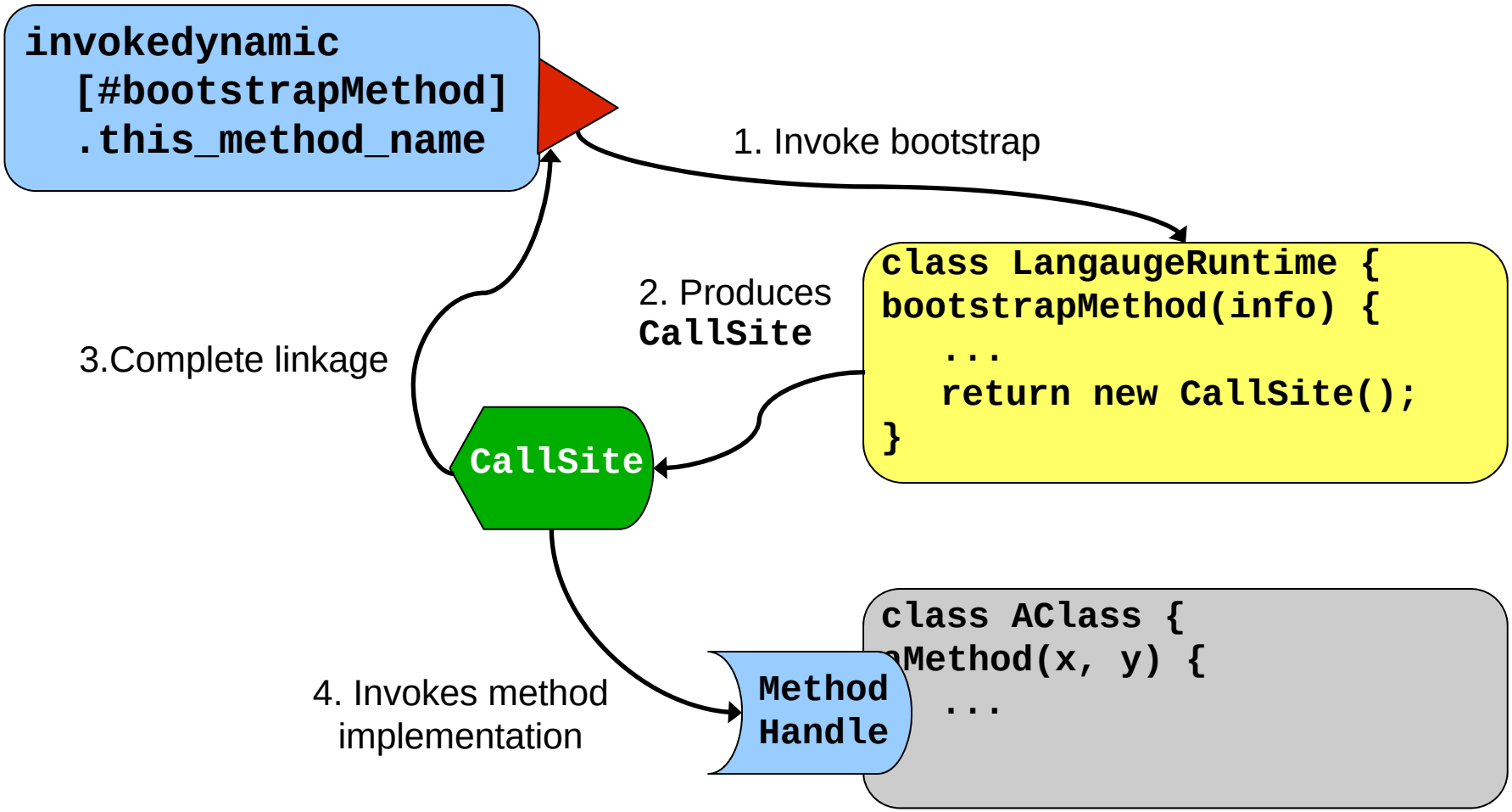
VM: Compressed OOPS by default

- Going from 32bit to 64bit system will grow the heap by ~1.5x simply because of bigger ordinary object pointers
- Memory is cheap, but bandwidth and cache is not
- Compressed OOPS:
  - Managed 32 bit pointers (similar heap sizes for 32/64 bit apps)
  - Scaled (8 x 4GB chunks) added to a 64 bit base
- Useful for heaps up to 32GB
  - Compressed OOPS will turn off when –Xmx > 32g



VM: invokedynamic Illustrated

```
this[method_name](x, y)
```



Java XML Technology Enhancements

- JAXP 1.4.5
  - Bug fixes and performance improvements
- JAX-WS 2.2.4
  - Bug fixes and performance improvements
- JAXB 2.2.3
  - Bug fixes and performance improvements

Elliptic Curve Cryptography (ECC)

- New Native Provider added to JDK 7
- ECC-based algorithms (ECDSA/ECDH)
- Enables ECC based Java Secure Sockets Extension (JSSE)
- Compared to traditional crypto systems like RSA, ECC offers equivalent security:
  - With smaller key sizes
  - Faster computations
  - Lower power consumption
  - Memory and bandwidth savings

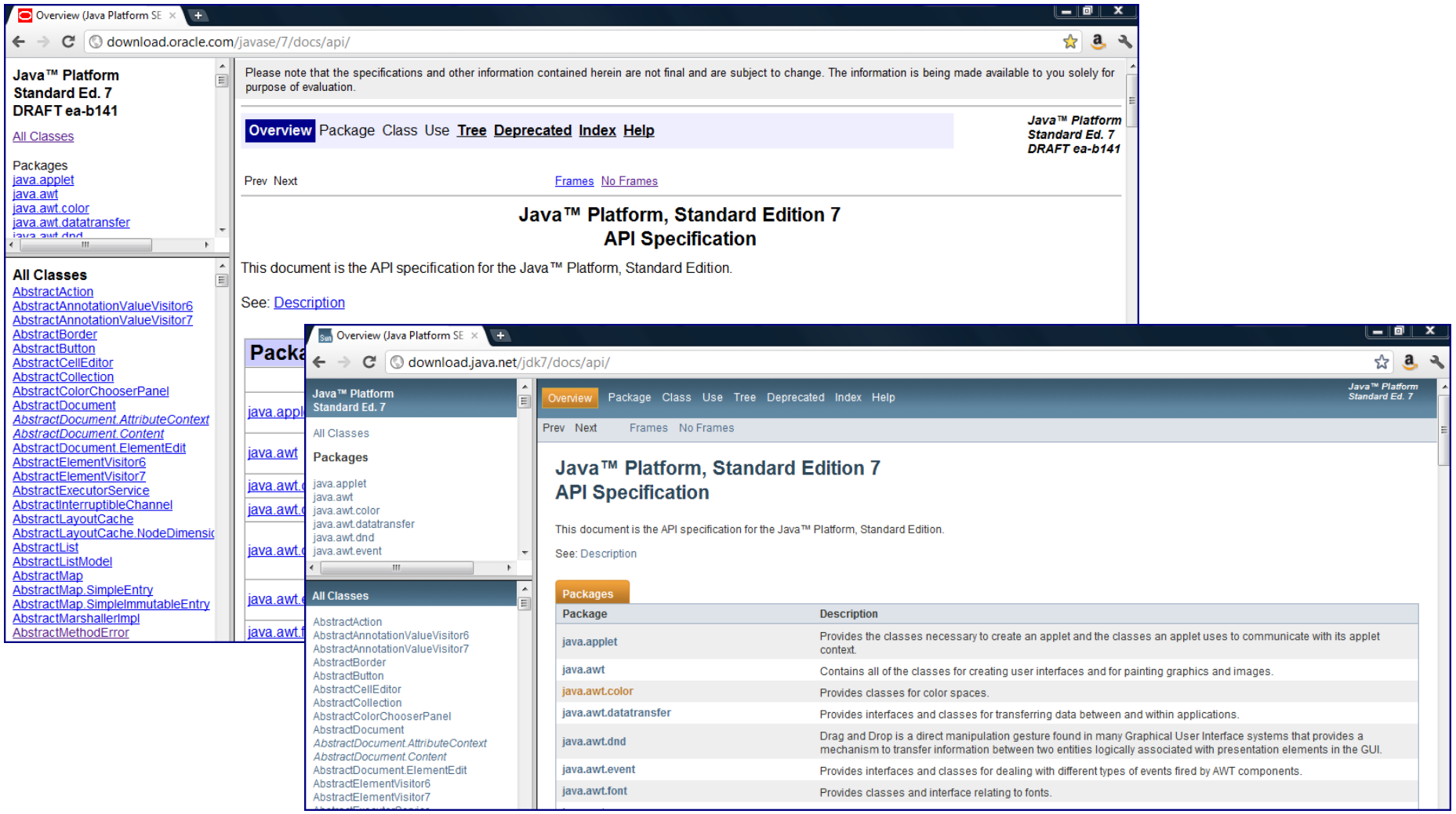
Transport Layer Security (TLS) Updates

- Support for TLS 1.1
  - Protection against cipher block chaining attacks
- Support for TLS 1.2
- TLS Renegotiation
- CertPath and TLS algorithm disabling
  - Can deny specific algorithms in path processing and handshaking, i.e., MD2

JavaDoc Improvements in Java 7

- Section 508 accessibility guidelines
  - Captions, headings, etc.
- Previously, JavaDoc wrote to an OutputStream on the fly meaning it built the document sequentially, imposing limitations
- Now uses internal “HTMLTree” classes
  - Generates compliant HTML
  - Allows for more advancements in the future
- Removes limitation of only being able to execute only once in any VM
  - Was fine when used as a command line tool
  - Continuous build, etc, made it necessary to address this!

CSS for JavaDoc - `stylesheet.css`



Feature 56 – Future Tech Duke!

Visit the Oracle Booth (by Theatre) for a Free Duke Plushie Keychain!

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.