JDK 7 Features

Sang Shin
JPassion.com
"Learn with Passion!"

Topics

- JDK 7 Features overview
- Small language changes (Project Coin)
- NIO.2
- Fork/Join
- Dynamic language support (invokedynamic)

JDK 7 Feature Overview

JDK 7 Features: Big

- Small language changes
  - Project Coin (JSR 334)
- More new IO APIs
  - NIO.2 (JSR 203)
- VM Support for dynamic languages
  - Da Vinci Machine project (JSR 292)

Features: Swing UI

- JLayer Class
  - > Add the SwingLabs JXLayer component decorator to the platform
- Nimbus Look & Feel
  - > A next-generation cross-platform look-and-feel for Swing
- Heavyweight and Lightweight Components
- Shaped and Translucent Windows
- Hue-Saturation-Luminance (HSL) Color Selection in JColorChooser Class

Features: Fast

- Garbage-First GC (G1)
  - A server-style garbage collector, targeted for multi-processors with large memories, that meets a soft real-time goal with high probability, while achieving high throughput.
  - G1 is the long term replacement of the Concurrent Mark-Sweep Collector (CMS)
  - Lower and predictable pause time
- Compressed 64-bit object pointers
  - A technique for compressing 64-bit pointers to fit into 32 bits
- Yet more HotSpot runtime compiler enhancements

Features from Others

- XRender pipeline for Java 2D
  - A new Java2D graphics pipeline based upon the X11 XRender extension
- Concurrency update (JSR 166y)
  - Fork/Join

Small Language Changes (Project Coin)

Binary Integral Literals

```
// An 8-bit 'byte' literal.
byte aByte = (byte) 0b00100001;
System.out.println("aByte = " + aByte);

// A 16-bit 'short' literal.
short aShort = (short) 0b1010000101000101;
System.out.println("aShort = " + aShort);
```

Underscores in numeric literals

```
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_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Better Type Inference

```
// Before JDK 7
Map<String, Integer> foo =
    new HashMap<String, Integer>();

// With JDK 7
Map<String, Integer> foo =
    new HashMap<>();
```

Strings in Switch

- Strings are constants too

```
String s = ...;
switch (s) {
   case "foo":
      return 1;

   case "bar":
      return 2;

   default:
      return 0;
}
```

Multi-catch Exceptions

```
try {
    boolean test = true;
    if (test) {
        throw new Exception1();
    } else {
        throw new Exception2();
    }
} catch (Exception1 | Exception2 e) {
    System.out.println("Exception type = " + e.getClass());
}
```

Lab:

Exercise 1.1-1.5: Small Language Changes
1607_jdk7_features.zip

Resource Management: pre-JDK7

- Manually closing resources is tricky and tedious

```
public void copy(String src, String dest) throws IOException {
    InputStream in = new FileInputStream(src);
    try {
        OutputStream out = new FileOutputStream(dest);
        try {
            byte[] buf = new byte[8 * 1024];
            int n;
            while ((n = in.read(buf)) >= 0)
                out.write(buf, 0, n);
        } finally {
            out.close();
        }
    } finally {
        in.close();
}}
```

Automatic Resource Management

```
static void copy(String src, String dest) throws IOException {
    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);
    }
   //in and out closes
}
```

`Closable` Interface

- Implemented by all auto close resources

```
package java.lang.auto;
/**
 * A resource that must be closed
 * when it is no longer needed.
 */
public interface AutoCloseable {
    void close() throws Exception;
}

package java.io;
public interface Closeable extends AutoCloseable {
    void close() throws IOException;
}
```

Lab:

Exercise 1.6: Small Language Changes
1607_jdk7_features.zip

NIO.2

NIO.2 Features

- Path operations (relative etc.)
- File operations (copy, check access, symlink)
- Directories (iterate)
- Recursive operations
- File change notification
- File attributes (NFSv4 ACL, Posix)
- Provider interface

FileSystem, FileSystems, FileStore

- java.nio.file.FileSystem
  - Provides an interface to a file system
  - Provides methods for accessing files and other objects in the file system
  - getPath(), getPathMatcher(), getFileStores(), etc
- java.nio.file.FileSystems
  - Defines the *getDefault()* method to get the default file system and factory methods to construct other types of file systems.
- java.nio.file.FileStore
  - A FileStore represents a storage pool, device, partition, volume, concrete file system or other implementation specific means of file storage

Path, Paths, Files

- java.nio.file.Path
  - A Path represents a path that is hierarchical and composed of a sequence of directory and file name elements separated by a special separator or delimiter
- java.nio.file.Paths
  - Consists exclusively of static methods that return a Path by converting a path string or URI.
- java.nio.file.Files
  - Consists exclusively of static methods that operate on files, directories, or other types of files
  - copy(..), createDirectory(..), CreateLink(..), CreateSymbolicLink(..), getFileStore(..), probeContentType

Path Example

- Paths may be used with the *Files* class to operate on files, directories, and other types of files.
- For example, suppose we want a BufferedReader to read text from a file "access.log". The file is located in a directory "logs" relative to the current working directory and is UTF-8 encoded

```
Path path = FileSystems.getDefault().getPath("logs", "access.log");
BufferReader reader = Files.newBufferedReader(path,
StandardCharsets.UTF_8);
```

Directory Tree Traversal

- java.nio.file.FileVisitor
  - A visitor of files
  - An implementation of this interface is provided to the *Files.walkFileTree* methods to visit each file in a file tree
- java.nio.file.FileVisitResult
  - The result type of a FileVisitor
- java.nio.file.FileVisitOption
  - Defines the file tree traversal options

File Change Notification Classes

- java.nio.file.WatchEvent
- java.nio.file.WatchKey
- java.nio.file.WatchService
- java.nio.file.Watchable

File Change Notification

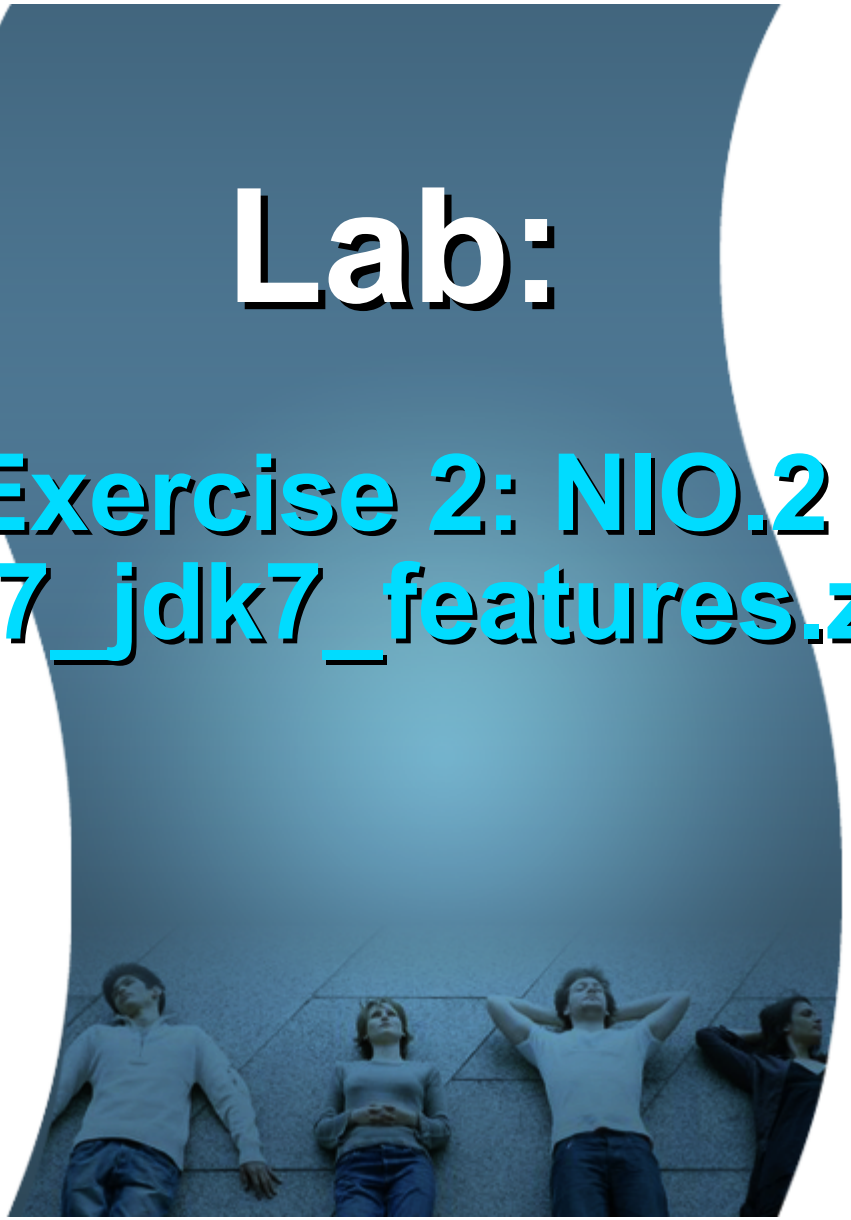
- Improve performance of applications that are forced to poll the file system today
- *WatchService*
  - Watch registered objects for events and changes
  - For example a file manager may use a watch service to monitor a directory for changes so that it can update its display of the list of files when files are created or deleted.
- A Watchable object is registered with a watch service by invoking its register method, returning a *WatchKey* to represent the registration
- *Path* interface extends *Watchable interface*
  - Register directory to get events when entries are created, deleted, or modified

User Defined Attributes

- User-defined file attributes are used to store metadata with a file that is not meaningful to the file system.
  - It is primarily intended for file system implementations that support such a capability directly but may be emulated.
  - The details of such emulation are highly implementation specific and therefore not specified.
- *java.nio.file.attribute.UserDefinedFileAttributeView*
  - Represents a file attribute view that provides a view of a file's user-defined attributes, sometimes known as extended attributes.

Lab:

Exercise 2: NIO.2
1607_jdk7_features.zip



Fork/Join

What is Fork/Join?

- New in the Java SE 7 release, the fork/join framework is an implementation of the *ExecutorService* interface that helps you take advantage of multiple processors
- It is designed for work that can be broken into smaller pieces recursively.
  - The goal is to use all the available processing power to make your application run faster

How Does Fork/Join Work?

- As with any *ExecutorService*, the fork/join framework distributes tasks to worker threads in a thread pool
  - The fork/join framework is distinct because it uses a work-stealing algorithm
  - Worker threads that run out of things to do can steal tasks from other threads that are still busy
- The center of the fork/join framework is the *ForkJoinPool* class, an extension of *AbstractExecutorService*
- *ForkJoinPool* implements the core work-stealing algorithm and can execute *ForkJoinTask's*.

How to Implement Fork/Join?

- Write some code that does the “division or work”
  - if (my portion of the work is small enough)
  - do the work directly
  - else
  - split my work into two pieces
  - invoke the two pieces and wait for the results
- Wrap this code as a *ForkJoinTask* subclass, typically as one of its more specialized types *RecursiveTask*(which can return a result) or *RecursiveAction*
- Call *invoke()* method of a *ForkJoinPool* instance

Lab:

Exercise 3: Fork/Join
1607_jdk7_features.zip

Dynamic Language Support

JVM Specification, First Edition (1997)

"The Java Virtual Machine knows nothing about the Java programming language, only of a particular binary format, the class file format."

"A class file contains Java Virtual Machine instructions (or bytecodes) and a symbol table, as well as other ancillary information."

"Any language with functionality that can be expressed in terms of a valid class file can be hosted by the Java virtual machine."

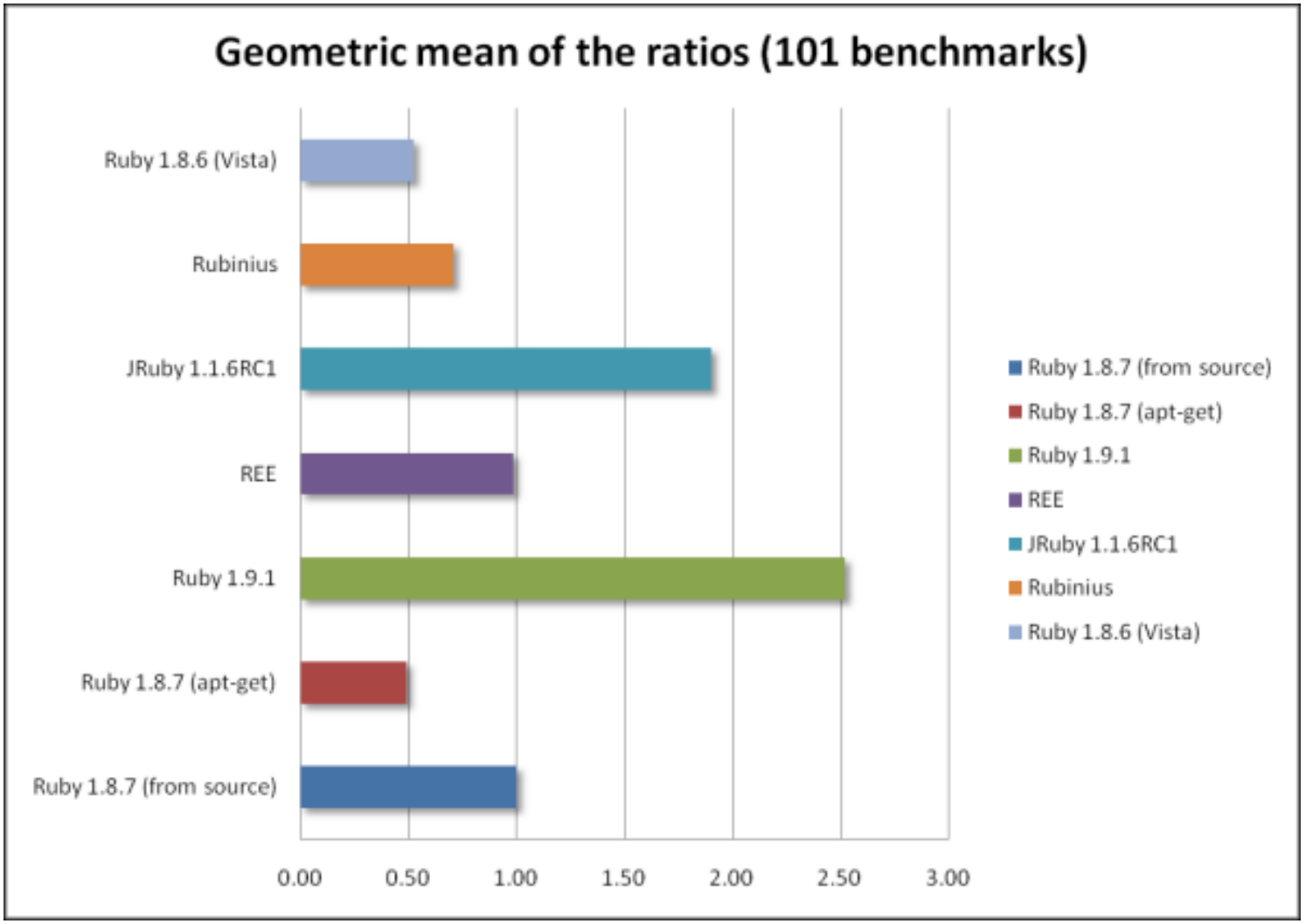
"Attracted by a generally available, machine-independent platform, implementors of other languages are turning to the Java Virtual Machine as a delivery vehicle for their languages."

"In the future, we will consider bounded extensions to the Java Virtual Machine to provide better support for other languages."

Languages on the Java Virtual Machine

Tea
iScript
Jickle
Zigzag
JESS
Modula-2
CAL
Lisp
JavaScript
Nice
Correlate
JudoScript
Basic
Simkin
Drools
Groovy
Eiffel
Luck
Icon
v-language
Pascal
Prolog
Mini
Scala
Tcl
PLAN
Hojo
Rexx
JavaFX Script
Funnel
FScript
Tiger
Anvil
Yassl
Oberon
E
Smalltalk
Logo
JHCR
JRuby
Tiger
Scheme
Ada
G
Clojure
Sather
Phobos
Processing
Dawn
WebL
Sleep
TermWare
LLP
Pnuts
Bex Script
BeanShell
PHP
Forth
C#
SALSA
ObjectScript
Yoix
Piccola
Jython

The Great Ruby Shootout 2008



http://antoniocangiano.com/2008/12/09/the-great-ruby-shootout-december-2008/

Pain Points for Dynamic Languages

- The current JVM is designed for static typed language
- Biggest mismatch between current JVM and dynamic languages is method selection and invocation
  - Calling a method is cheap; selecting the right method is expensive
  - Static languages do most of their method selection at compile-time
  - Dynamic languages do almost none at compile-time (obviously) instead during runtime
- What is the one change in the JVM that would make life better for dynamic languages?
  - Flexible method calls!

JSR 292 (Support for Dynamically Typed Languages in the Java Virtual Machine)

- Introduced a new bytecode called "*invokedynamic*" and new linkage mechanism called Method Handle
  - Dynamic language compiler designers no longer have to do backdoor work
- Enables even higher performance of the applications written in dynamic language
- Da Vinci Machine Project, an OpenJDK community provides the implementation

Learn with Passion!
JPassion.com