
JavaOne™

JDK7 In Action

Using New Core Platform Features In Real Code

Joe Darcy @jddarcy
Mike Duigou @mjduigou
Stuart Marks @stuartmarks

Oracle JDK Team

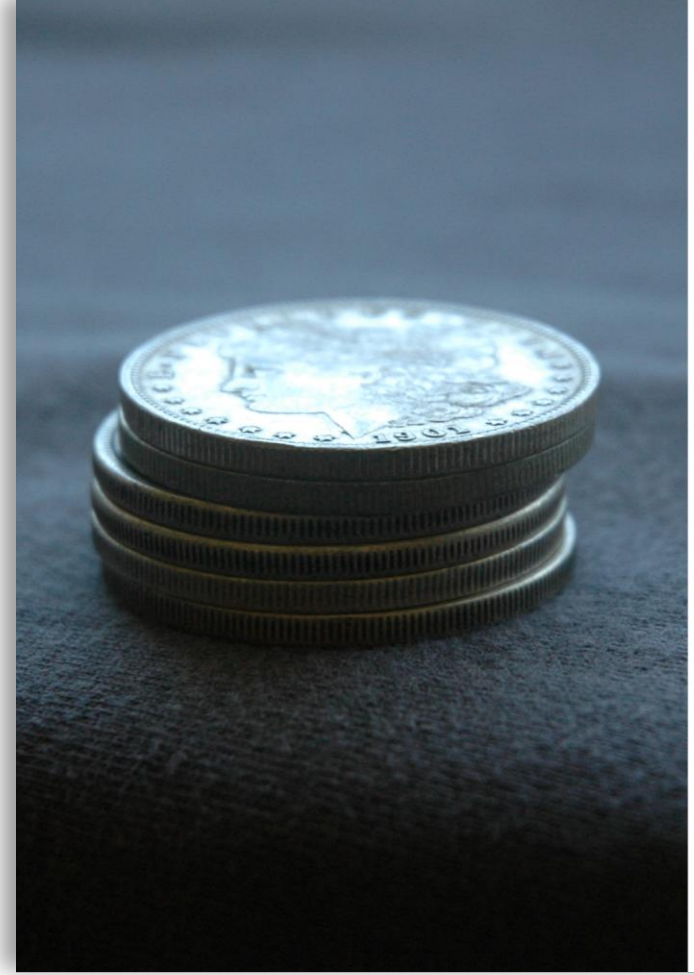
Program Agenda

- Project Coin — Small Language Features
- NIO.2 File System APIs
- Fork/Join Framework
- Core Library Morsels

Audience Poll

- Which JDK version are you using?
  - pre-1.4.2
  - 1.4.2
  - 5
  - 6
  - 7
  - 8 preview builds
- Uptake of 7 features?
- Which IDE?

Project Coin



Project Coin is a suite of language and library changes to make things programmers do every day easier.

Project Coin Benefits

Enjoy improved free code flow today!

- Remove extra text to make programs more *readable*
- Encourage writing programs that are more *reliable*
- Integrate well with past and future changes

Coin Constraints

- *Small* language changes
  - Specification
  - Implementation
  - Testing
- No JVM changes!
- Coordinate with forthcoming larger language changes
- Beware the hazards of language interactions!

The Six Coin Features and How They Help

- Consistency and clarity
  - 1. Improved literals
  - 2. Strings in switch
- Easier to use generics
  - 3. SafeVarargs (removing varargs warnings)
  - 4. Diamond
- More concise error handling
  - 5. Multi-catch and precise rethrow
  - 6. Try-with-resources

1. Improved Literals

Improved Literals

- Binary integral literals
  - new “0b” prefix
- Underscores in numeric literals
  - can have ***multiple*** underscores ***between*** digits

Integral Binary Literals

```
// An 8-bit 'byte' value:
byte aByte = (byte)0b00100001;

// A 16-bit 'short' value:
short aShort = (short)0b1010000101000101;

// Some 32-bit 'int' values:
int anInt1 = 0b10100001010001011010000101000101;
int anInt3 = 0B101; // The B can be upper or lower case.

// A 64-bit 'long' value. Note the "L" suffix:
long aLong =
0b10100001010001011010000101000101101000010100010110100001010001 01L;
```

Underscores in Literals

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;

long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;

byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Underscores in Literals

- Grammar changes a bit tricky to get right; **multiple** underscores **between** digits:

Digits:
Digit
Digit DigitsAndUnderscores$_{opt}$ *Digit*

DigitsAndUnderscores:
DigitOrUnderscore
DigitsAndUnderscores DigitOrUnderscore

Implication of Multiple Underscores

Do we want this to be allowed?

```
// Courtesy Josh Bloch
int bond =
        0000_____________0000_______0000000000000000__0000000000000000+
      00000000_________00000000_____0000000000000000__0000000000000000+
     000____000_______000____000_____000_______0000__00_____0+
    000_____000_____000_____000_____________0000___00_____0+
   0000_____0000___0000_____0000_____________0000_____0_____0+
   0000_____0000___0000_____0000_____________0000___________0+
   0000_____0000___0000_____0000_____________000+__0000000000+
   0000_____0000___0000_____0000_____________0000+
    000_____000_____000_____000_____________0000+
     000____000_______000____000_____________00000+
      00000000_________00000000_____0000000+
        0000_____________0000_____________000000007;
```

2. Strings in Switch

Strings in Switch

```
switch (name) {
    case "Athos":
    case "Porthos":
    case "Aramis":
        System.out.println("One of the Three Musketeers");
        break;
    case "d'Artagnan":
        System.out.println("Not a Musketeer");
        break;
    default:
        throw new IllegalArgumentException();
}
```

Strings in Switch Specification

- JLS §14.11 The switch Statement

  “The type of the switch *expression* must be `char`, `byte`, `short`, `int`, `Character`, `Byte`, `Short`, `Integer`, `String`, or an enum type (§8.9), or a compile-time error occurs.”

Strings in Switch Specification

What is there to discuss?

- What does switching on a null do? ***(NullPointerException)***
- Can null be a case label? ***(No.)***
- Case-insensitive comparisons? ***(No.)***
- Implementation
  - relies on a particular algorithm be used for **`String.hashCode`**
  - on average faster than if-else chain with >3 cases

3. Safe Varargs

Safe Varargs

```
List<String> list1 = ...
List<String> list2 = ...
List<String> list3 = ...

Set<List<String>> setOfLists = new HashSet<List<String>>();

Collections.addAll(setOfLists, list1, list2, list3);
                   ^
warning: [unchecked] unchecked generic array
creation of type java.util.List<java.lang.String>[]
for varargs parameter
```

“If your entire application has been compiled without unchecked warnings, it is type safe.”

Generics Tutorial Extra
http://docs.oracle.com/javase/tutorial/extra/generics/index.html

Gilad Bracha, Computational Theologist (emer.)

Safe Varargs

Background

- Desirable to have a ***sound*** system of warnings
  - No missed cases (no false negatives)
  - But may have false positives

Unchecked Warnings & Heap Pollution

"To make sure that potential violations of the typing rules are always flagged, some accesses to members of a raw type will result in compile-time unchecked warnings." — JLS §4.8 Raw Types

"Heap pollution can only occur if the program performed some operation involving a raw type that would give rise to a compile-time unchecked warning or if the program aliases an array variable of non-reifiable element type through an array variable of a supertype which is either raw or non-generic." — JLS §4.12.2 Variables of Reference Type

Unchecked Warnings and Soundness

- Unchecked warnings are intended to be a sound analysis
- From a certain point of view, it would be correct (but unhelpful!) to *always* emit an unchecked warning
- Pre-JDK 7, all callers always got an unchecked warning when calling certain varargs library methods (e.g., Collections.addAll)
  - Bad, and complicated interaction between generics and arrays
  - But usually nothing actually dangerous happens!

The @SafeVarargs Annotation

- Added to the problematic library calls in Java 7
  - java.util
    - Arrays.asList
    - Collections.addAll
    - EnumSet.of
  - javax.swing
    - SwingWorker.publish
- Shuts off annoying warnings to the callers of these methods
  - No changes necessary to callers!

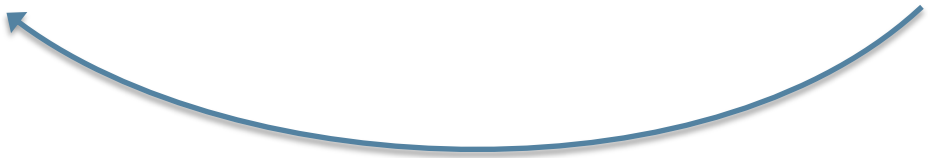
@SafeVarargs Design Considerations

- Annotations on methods are *not* inherited
- **`@SafeVarargs`** can therefore only be applied to
  - Static methods
  - Constructors
  - **`final`** instance methods
- Additional checks on varargs status, etc.
- Runtime retention policy

4. Diamond <>

Diamond <>

```
Set<List<String>> setOfLists = new HashSet<List<String>>();

Set<List<String>> setOfLists = new HashSet<>();
```



the type in the diamond is inferred from the declaration

Diamond Use: Variable Initializer

```
Set<List<String>> setOfLists = new HashSet<>();
```

Diamond Use: Assignment Statement

```
List<Map<String,Integer>> listOfMaps;
...
...
...
listOfMaps = new ArrayList<>();
```

Diamond Use: Return Statement

```
public Set<Map<BigInteger,BigInteger>> compute() {
    ...
    ...
    ...
    return new Set<>();
}
```

Diamond <>

- Diamond uses ***type inference*** to figure out types so the programmer doesn't have to write them
- Primary input to type inference is the ***target type***
  - Determined the context within which the expression occurs
- Type inference is a ***constraint satisfaction*** problem
  - What are the constraints?
  - How can they be satisfied?
- Want a ***unique*** answer returned by the algorithm

More Than One Facet

- Two inference schemes proposed, differing in how they gathered constraints
- Each sometimes more useful than the other
- Use *quantitative analysis* to help resolve the issue
  - Prototype both schemes
  - Analyze results on millions of files of code

Quantitative Results

- Both schemes equally effective
  - Each could eliminate a different 90% of the explicit type parameters to constructor calls
  - Verified diamond was a worthwhile feature!
- Choose inference scheme with better evolution and maintenance properties
- *Language designer's notebook: Quantitative language design* http://www.ibm.com/developerworks/java/library/j-ldn1/

5. Multi-Catch and Precise Rethrow

Multi-Catch and Precise Rethrow

- Multi-catch:
  - ability to catch multiple exception types in a single catch clause

```
try {
    ...
} catch (FirstException | SecondException) { ... }
```

- Precise rethrow:
  - change in ***can-throw analysis*** of a catch clause

Multi-Catch and Precise Rethrow

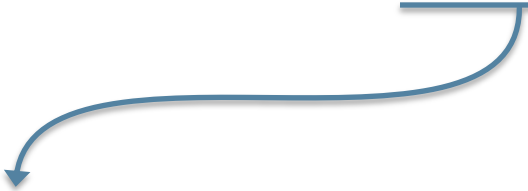
- Java's checked exceptions must either:
  - Be handled by a ***catch*** clause; or
  - Be declared in the ***throws*** clause of the containing method.
- Where do checked exceptions come from?
  - The ***throw*** statement
  - The ***throws*** clause of called methods
- ***Can-throw*** analysis
  - Determines what exceptions can be thrown by a block of code

Multi-Catch and Precise Rethrow

- The ***can-throw*** analysis for a catch block has changed
- Java 6 and earlier:
  - the declared type of the exception variable
- Java 7 and later:
  - ***If*** the exception variable is effectively final (not assigned),
  - Only the checked exceptions that can be thrown by the try-block

Multi-Catch and Precise Rethrow

```
void exampleMethod(Future future) throws
    InterruptedException, ExecutionException, TimeoutException
{
    Object result = future.get(5, SECONDS);
}
```



Declaration has:

```
throws InterruptedException, ExecutionException, TimeoutException
```

How would we catch, clean up, and rethrow?

Multi-Catch and Precise Rethrow

```
void exampleMethod(Future future) throws
    InterruptedException, ExecutionException, TimeoutException
{
    try {
        Object result = future.get(5, SECONDS);
    } catch (InterruptedException ex) {
        cleanup();
        throw ex;
    } catch (ExecutionException ex) {
        cleanup();
        throw ex;
    } catch (TimeoutException ex) {
        cleanup();
        throw ex;
    }
}
```

Java 6: multiple catch clauses

Multi-Catch and Precise Rethrow

```
void exampleMethod(Future future) throws
    Exception
{
    try {
        Object result = future.get(5, SECONDS);
    } catch (Exception ex) {
        cleanup();
        throw ex;
    }
}
```

Java 6: catch "wider" exception type (poor style)

Multi-Catch and Precise Rethrow

```
void exampleMethod(Future future) throws
    InterruptedException, ExecutionException, TimeoutException
{
    try {
        Object result = future.get(5, SECONDS);
    } catch (InterruptedException | ExecutionException |
             TimeoutException ex) {
        cleanup();
        throw ex;
    }
}
```

Java 7: multi-catch

Multi-Catch and Precise Rethrow

```
void exampleMethod(Future future) throws
    InterruptedException, ExecutionException, TimeoutException
{
    try {
        Object result = future.get(5, SECONDS);
    } catch (Throwable th) {
        cleanup();
        throw th;
    }
}
```

Java 7: precise rethrow (is this good style now?)

Precise Rethrow — Compatibility

Does it matter if this code doesn't compile in JDK 7?

```
try {
    throw new DaughterOfFoo();
} catch (Foo exception) {
    try {
        throw exception;
            // in JDK6, exception is Foo
            // in JDK7, exception is DaughterOfFoo
    } catch (SonOfFoo anotherException) {
        ;   // Reachable?  JDK6=yes, JDK7=no
    }
}
```

6. Try-with-resources

6. Try-with-resources

Largest of the Coins

- A variation of the try-catch-finally statement
- Allows initialization of a ***resource variable***
  - Must be of type AutoCloseable
  - Its close() method is called from a generated finally-block
  - Exceptions thrown by close() added to ***suppressed exception list***
- Useful for avoiding leaks of external objects
  - Files, channels, sockets, SQL statements, ...
  - Many JDK classes retrofitted to be AutoCloseable

Try-with-resources

You type this:

```
try (Resource r = aa()) {
    bb();
} catch (Exception e) {
    cc();
} finally {
    dd();
}
```

Compiler generates this:

```
try {
    Resource r = null;
    try {
        r = aa();
        bb();
    } finally {
        if (r != null)
            r.close();
    }
} catch (Exception e) {
    cc();
} finally {
    dd();
}
```

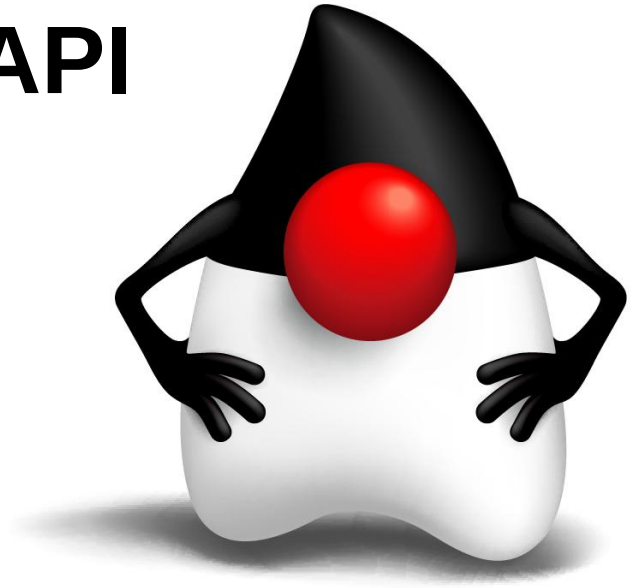
It's actually more complicated because of the way exceptions from close() are handled.

Project Coin Summary

- Features add clarity, conciseness, and convenience
- Methodical and quantitative design approach
  - Decide today what needs to be decided today
  - Consciously leave room for future decisions
- Language + library co-evolution (but no VM changes)
- Smooth transition to new features
  - Widespread tool support
  - Use of new features reads well

DEMO

NIO.2 File System API



Background and Motivation

- The platform was long overdue something better than java.io.File
  - Doesn’t work consistently across platforms
  - Lack of useful exceptions when a file operation fails
  - Missing basic operations, no file copy, move, ...
  - Limited support for symbolic links
  - Very limited support for file attributes
  - No bulk access to file attributes
  - Badly missing features that many applications require
  - No way to plug-in other file system implementations

New File System API

- Package java.nio.file
- Also java.nio.file.attribute and java.nio.file.spi
- Some additions to java.io and java.nio.channels

New File System API

- `Path` – used to locate a file in a file system
- `Files` – defines static methods to operate on files, directories and other types of files
- `FileSystem`
  - Provides a handle to a file system
  - Factory for objects that access the file system
  - `FileSystems.getDefault` returns a reference to the default FileSystem
- `FileStore` – the underlying storage/volume

Path

- Represents an absolute or relative path
- Create from path String or URI or File.toPath()
- Consists of one or more name elements, or a root component and zero or more name elements
- Immutable
- Defines methods to access elements of the path
- Defines methods to combine paths, return a new path

Creating a Path

```
Path path = FileSystems.getDefault().getPath("/foo");
```

Creating a Path

```
Path path = Paths.get("/foo");
```

Creating a Path

```
URI u = URI.create("file:///foo");

Path path = Paths.get(u);
```

Creating a Path

```
File f = new File("foo");

Path path = f.toPath();
```

Accessing Components

```
//  foo/bar/gus/baz
Path path = Paths.get("foo", "bar", "gus", "baz");

Path name = path.getFileName();    // baz

Path parent = path.getParent();    // foo/bar/gus

Path subpath = path.subpath(1,3);  // bar/gus
```

Combining Paths

```
Path dir = Paths.get("/foo/bar");

Path gus = dir.resolve("gus");              //  /foo/bar/gus

Path baz = dir.resolveSibling("baz");       //  /foo/baz
```

Testing Paths

```
Path dir = Paths.get("foo/bar");

boolean isAbsolute = dir.isAbsolute();

boolean isFoo = dir.startsWith("foo");
```

Other Methods

```
Path np = path.normalize();

Path rel = path.relativize(other);

Path rp = path.toRealPath();

URI u = path.toUri();
```

Files class

- Consists exclusively of static methods that operate on files
- Most methods take a Path as a parameter to locate the file

```
Files.createFile(path);
```

Files class

- In many cases the return value is a Path too

```
Path foo = Files.createDirectory(dir.resolve("foo"));
```

Files class

- Methods that access the file system throw a useful IOException if they fail
- Defines range of methods for working with regular files, directories

Regular Files

- Simple operations

```
Path path = ...

byte[] bytes = Files.readAllBytes(path);
```

Regular Files

- Simple operations

```
import static java.nio.charsets.StandardCharsets.*;

Path path = ...

List<String> lines =  Files.readAllLines(path, UTF_8);
```

Text Files

```
import static java.nio.charsets.StandardCharsets.*;

BufferedReader reader =
    Files.newBufferedReader(path, UTF_8);

BufferedWriter writer =
    Files.newBufferedWriter(path, ISO_8859_1);
```

Input and Output Streams

```
Path = ...

InputStream in = Files.newInputStream(path);

OutputStream out = Files.newOutputStream(path);
```

Input and Output Streams

```
import java.nio.file.StandardOpenOption.*;

Path = ...

OutputStream out =
    Files.newOutputStream(path, CREATE, APPEND);
```

Channels

- Files.newByteChannel to open file, returning channel
- SeekableByteChannel
  - ByteChannel that maintains a file position
  - Channel equivalent of RandomAccessFile

```
import java.nio.file.StandardOpenOption.*;

SeekableByteChannel sbc =
    Files.newByteChannel(path, READ, WRITE);
```

Channels

- FileChannel for more advanced features
  - memory mapped I/O, file locking, ...
  - Implements SeekableByteChannel
  - Now defines open methods to open a file directly
- AsynchronousFileChannel class for asynchronous file I/O
  - Supports asynchronous read, write, locking

Directories

- DirectoryStream to iterate over entries
  - Scales to large directories
  - Uses less resources
  - Smooth out response times for remote file systems
  - Provides handle to open directory
  - Extends Iterable and Closeable
- Filtering
  - Built-in support for glob and regex patterns
  - Can also use custom filters

DirectoryStream

```
Path dir = ...

DirectoryStream<Path> stream =
    Files.newDirectoryStream(dir);
```

DirectoryStream

```
Path dir = ...

try (DirectoryStream<Path> stream =
         Files.newDirectoryStream(dir)) {
    for (Path entry: stream) {
        System.out.println(entry.getFileName());
    }
}
```

DirectoryStream

```
try (DirectoryStream<Path> stream =
        Files.newDirectoryStream(dir, "*.java")) {

}
```

DirectoryStream

```
DirectoryStream.Filter<Path> filter =
        new DirectoryStream.Filter<Path>() {
    public boolean accept(Path entry) throws IOException {
        return Files.size(entry) > 8192L;
    }
};

try (DirectoryStream<Path> stream =
    Files.newDirectoryStream(dir, filter)) {

}
```

Symbolic Links

- Special file that is a reference to another file
- Optionally supported based on long standing Unix semantics
- Followed by default, with some exceptions (detete, move, ...)

Symbolic Links

```
Path path = ...

boolean isSymLink = Files.isSymbolicLink(path);
```

Symbolic Links

```
Path link = ...
Path target = ...

Files.createSymbolicLink(link, target);
```

Symbolic Links

```
Path link = ...

Path target = Files.readSymbolicLink(link);
```

Symbolic Links

```
Path path1 = ...
Path path2 = ...

boolean isSame = Files.isSameFile(path1, path2);
```

Symbolic Links

```
Path path = ...

BasicFileAttributes attrs =
    Files.readAttributes(path,
                         BasicFileAttributes.class);
```

Symbolic Links

```
import static java.nio.file.LinkOption.*;

Path path = ...

BasicFileAttributes attrs =
    Files.readAttributes(path,
                         BasicFileAttributes.class,
                         NOFOLLOW_LINKS);
```

Other File Operations

```
Path source = ...
Path target = ...

Files.copy(source, target);
```

Other File Operations

```
import static java.nio.file.StandardCopyOption.*;

Path source = ...
Path target = ...

Files.copy(source, target, REPLACE_EXISTING);
```

Other File Operations

```
import static java.nio.file.StandardCopyOption.*;

Path source = ...
Path target = ...

Files.copy(source, target,
    REPLACE_EXISTING, COPY_ATTRIBUTES);
```

Other File Operations

```
Path source = ...
URI u = ...

try (InputStream in = u.toURL().openStream()) {
    Files.copy(in, path);
}
```

Other File Operations

```
Path source = ...
Path target = ...

Files.move(source, target);
```

File Attributes

- Meta-data associated with file
- Highly platform and file system specific
- Many applications need access to
  - File permissions
  - File owner
  - Timestamps
  - Extended attributes
- Long standing performance issues

File Attributes

- Group related attributes
- Define a *view* that provides
  - Defines the attributes in the group (name and type)
  - Provides bulk access where appropriate
  - Provides type safe access
- All implementations required to support a *basic* view
- Implementation may support additional views.

Basic Attributes

```
import java.nio.file.attribute.BasicFileAttributes;

BasicFileAttributes attrs =
    Files.readAttributes(path, BasicFileAttributes.class);

long size = attrs.size();

boolean isDirectory = attrs.isDirectory();

FileTime lastModified = attrs.lastModifiedTime();
```

Basic Attributes

```
interface BasicFileAttributes {
    FileTime lastModifiedTime();
    FileTime lastAccessTime();
    FileTime creationTime();
    long size();
    boolean isRegularFile();
    boolean isDirectory();
    boolean isSymbolicLink();
    boolean isOther();
    Object fileKey();
}
```

File Attributes

- API defines several other views
  - POSIX
  - ACL support based on NFSv4 ACL model (RFC 3530)
  - User-defined attributes
- Implementations may support additional views
- FileStore defines supportsFileAttributeView method to test if attributes are supported by underlying file store.

```
PosixFileAttributes attrs =
    Files.readAttributes(path, PosixFileAttributes.class);

UserPrincipal owner = attrs.owner();

UserPrincipal group = attrs.group();

Set<PosixFilePermission> perms = attrs.permissions();
```

Setting Initial Attributes

```
Set<StandardOpenOption> opts =
    EnumSet.of(CREATE_NEW, WRITE);

Set<PosixFilePermission> perms =
    PosixFilePermissions.fromString("rw-r-----");

WritableByteChannel wbc = Files.newByteChannel(path, opts,
    PosixFilePermissions.asFileAttribute(perms);
```

Recursive Operations

- Files.walkFileTree
  - Internal iterator to walk a file tree from a given starting point
  - FileVisitor invoked for each file/directory encountered
  - SimpleFileVisitor with default behavior
  - Depth first, invoked twice for each directory (pre/post)
  - Return value controls iteration
  - When following symbolic links then cycles are detected and reported
- Most of the samples in the JDK samples directory use it

```
Path start = ...

Files.walkFileTree(start, new SimpleFileVisitor<Path>() {









});
```

```
Path start = ...

Files.walkFileTree(start, new SimpleFileVisitor<Path>() {




    public FileVisitResult
        visitFile(Path file, BasicFileAttributes attrs)
    {
        System.out.println(file);
        return FileVisitResult.CONTINUE;
    }
});
```

```
Path start = ...

Files.walkFileTree(start, new SimpleFileVisitor<Path>() {
    public FileVisitResult
        preVisitDirectory(Path dir, BasicFileAttributes attrs)
    {
        System.out.format("%s/%n", dir);
        return FileVisitResult.CONTINUE;
    }
    public FileVisitResult
        visitFile(Path file, BasicFileAttributes attrs)
    {
        System.out.println(file);
        return FileVisitResult.CONTINUE;
    }
});
```

File Change Notification

- Watch files and directories for changes
- Prime motivation is to eliminate the need to poll the file system
- WatchService
  - Watches registered objects for changes and events
  - Makes use of native event notification facility where available
  - Minimally required to support monitoring directories: events when files in directory created, modified or deleted
  - May support watching other types of objects or other events
  - Deliberately a low-level interface to be used in different contexts

Registration

```
import static java.nio.file.StandardWatchEventKinds.*;

WatchService watcher =
    FileSystems.getDefault().newWatchService();

Path dir = ..

WatchKey key =
    dir.register(watcher, ENTRY_CREATE, ENTRY_DELETE);
```

Retrieving Events

```
for (;;) {
    WatchKey key = watcher.take();




    key.reset();
}
```

Retrieving Events

```
for (;;) {
    WatchKey key = watcher.take();
    for (WatchEvent<?> event: key.pollEvents()) {




    }
    key.reset();
}
```

Retrieving Events

```
for (;;) {
    WatchKey key = watcher.take();
    for (WatchEvent<?> event: key.pollEvents()) {
        if (event.kind() == ENTRY_CREATE) {
            Path name = (Path)event.context();
            System.out.format("%s created%n", name);
        }
    }
    key.reset();
}
```

Provider Interface

- Use to develop and deploy custom file system implementations
- FileSystemProvider is a factory for FileSystem instances
- Deploy as JAR file on class path or install as extension
- java.nio.file.FileSystems defines methods to create and obtain references to custom file systems
- Can replace default provider
- Can interpose on default provider

ZIP Provider

- Sample file system provider shipped with JDK 7
- Treats the contents of zip or JAR file as a file system
- Works out of the box
- Also included as a demo

ZIP Provider

```
Path zipfile = Paths.get("foo.zip");

try (FileSystem zipfs =
        FileSystems.newFileSystem(zipfile, null)) {









}
```

ZIP Provider

```
Path zipfile = Paths.get("foo.zip");

try (FileSystem zipfs =
        FileSystems.newFileSystem(zipfile, null)) {
    Path top = zipfs.getPath("/");









}
```

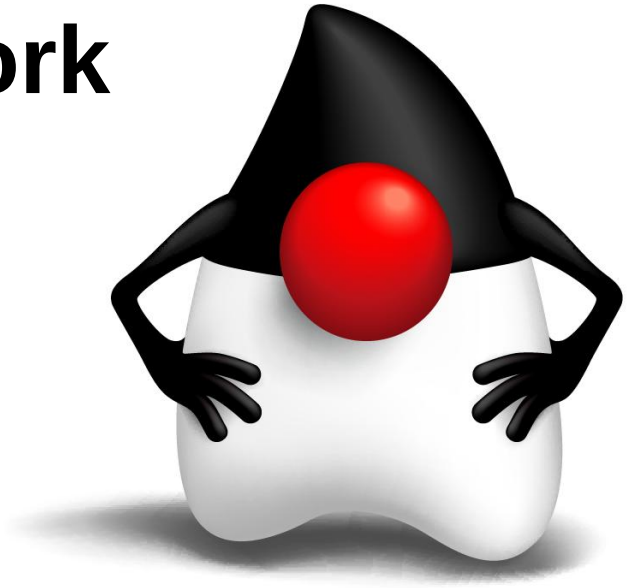
ZIP Provider

```
Path zipfile = Paths.get("foo.zip");

try (FileSystem zipfs =
        FileSystems.newFileSystem(zipfile, null)) {
    Path top = zipfs.getPath("/");
    try (DirectoryStream stream =
            Files.newDirectoryStream(top)) {
        for (Path entry: stream) {
            System.out.println(entry.getFileName());
        }
    }
}
```

Wrap-up

- The JDK finally gets a comprehensive interface to the file system
- JDK 7 shipping last July so you can use it now
- Easy to use, yet powerful
- Extensible via the provider interface

Fork-Join Framework

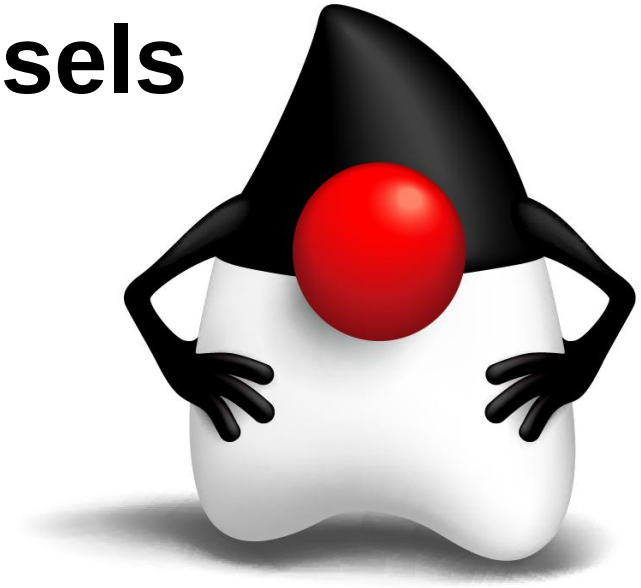


Fork-Join Framework

- Low overhead work stealing `ExecutorService`
- “Large tasks should be split into smaller subtasks, usually via recursive decomposition. If tasks are too big, then parallelism cannot improve throughput. If too small, then memory and internal task maintenance overhead may overwhelm processing.”

DEMO

Core Libraries Morsels



Core Libraries Morsels

- “Coins” are small language features
- Small library features are “morsels”

java.util.Objects

- Utility class of static methods
  - `compare(obj1,obj2,comparator)`
  - `requireNonNull(obj), requireNonNull(obj,msg)`
  - `hashCode(obj), hash(Objects … objs)`
  - `equals(obj1,obj2) deepEquals(obj1,obj2)`
  - `toString(obj), toString(obj,nullDefault)`

java.nio.charset.StandardCharsets

- Constants for six standard charsets
  - UTF8, UTF16, UTF16_BE, UTF16_LE, US_ASCII, ISO_8859_1
- Key advantage is ability to use Charset variant methods
  - No more constantly catching `UnsupportedCharsetException`!

Java Collections Framework

New utilities

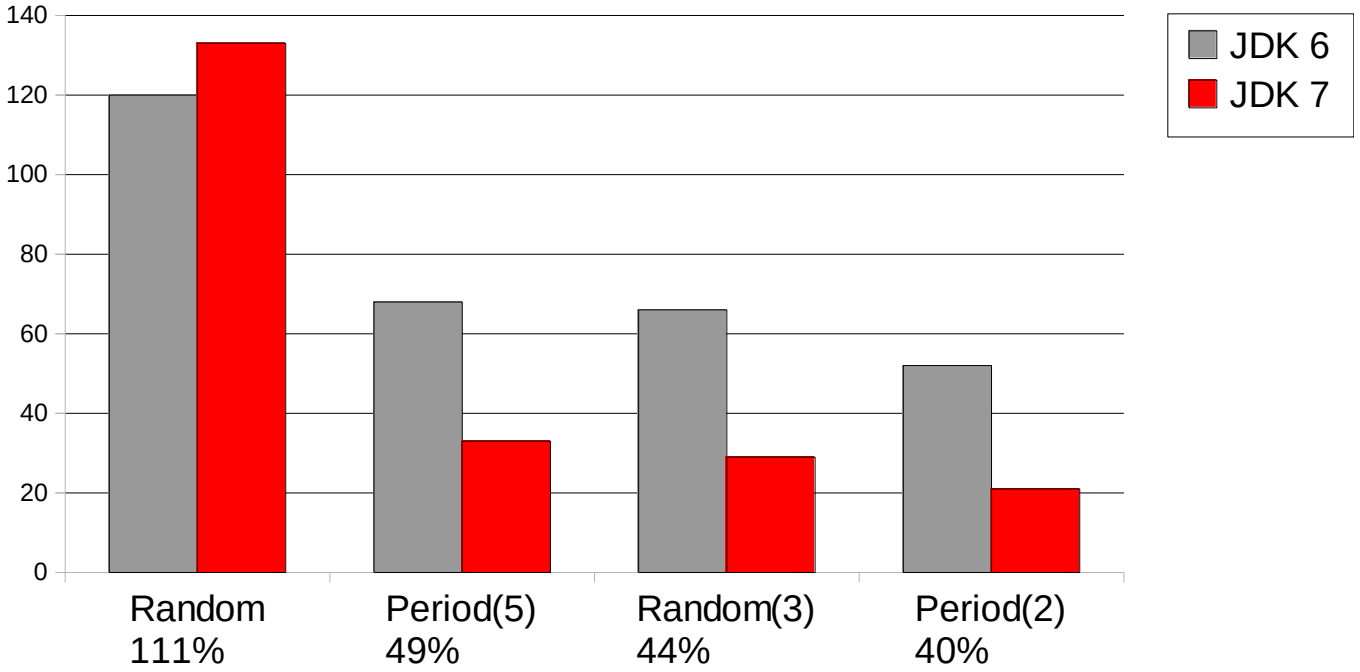
```
Collections.emptyIterator()

Collections.emptyListIterator()

Collections.emptyEnumeration()
```

Java Collections Framework

Major Sorting improvements

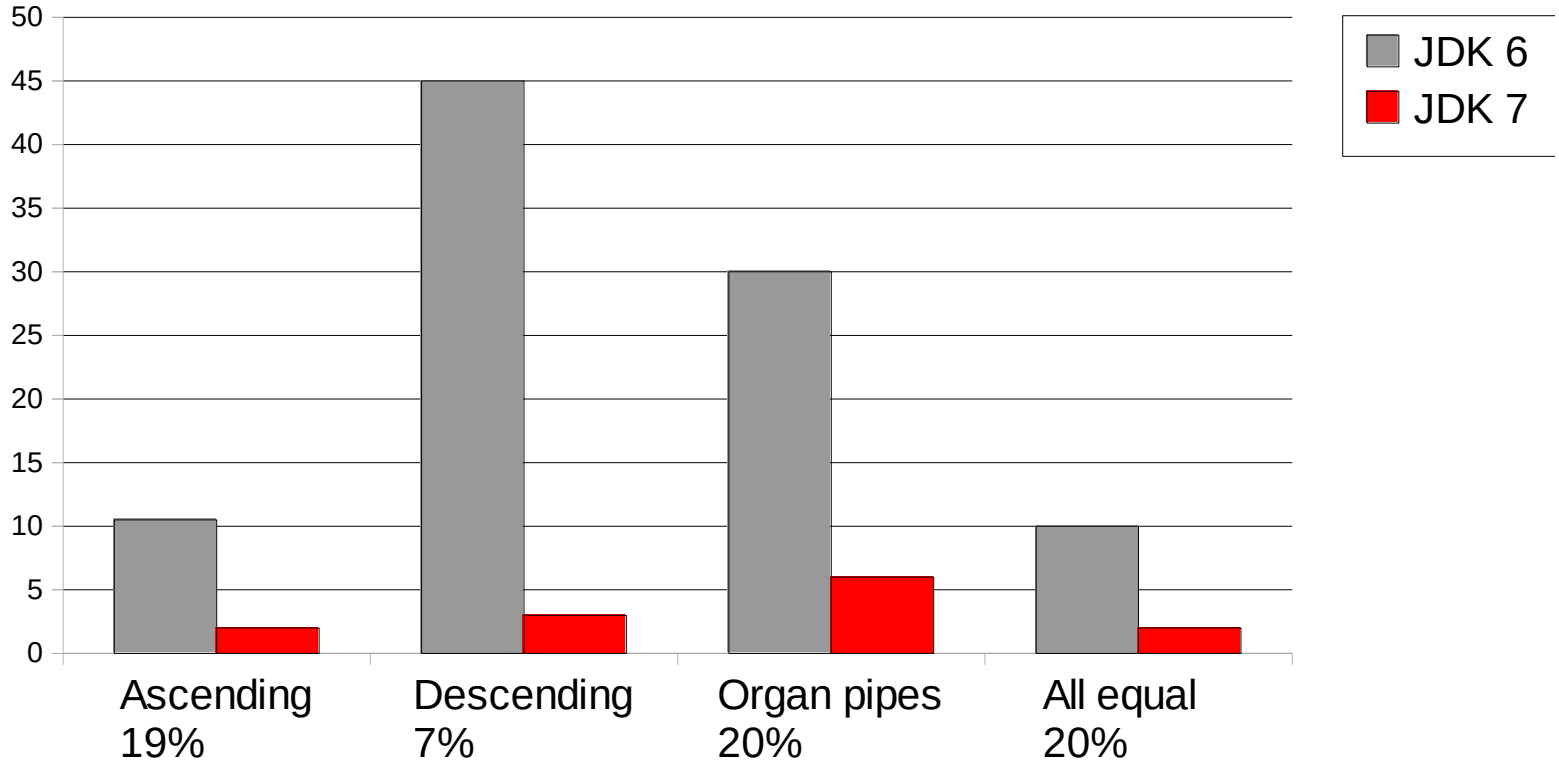
- TimSort
  - Capitalizes on runs of partially sorted data
  - Arrays of Object & ArrayList
  - Contributed by Joshua Bloch
  - Designed by Tim Peters
- Dual Pivot Quicksort
  - 0.8*n**ln(*n*) average comparisons vs Quicksort *n**ln(*n*)
  - Arrays of primitive values
  - Contributed by Vladimir Yaroslavskiy
  - Designed by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch

TimSort Performance



Results courtesy of Vladimir Yaroslavskiy

TimSort Performance



Results courtesy of Vladimir Yaroslavskiy

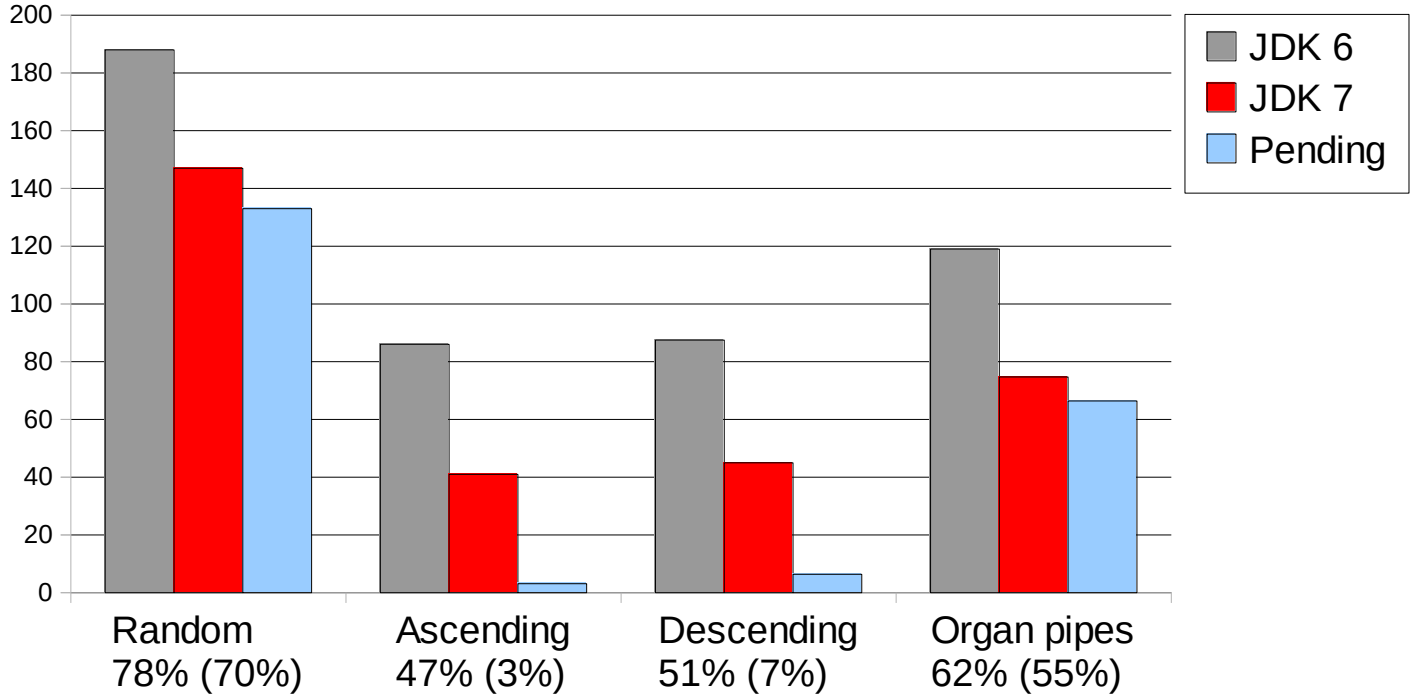
TimSort Summary

- Benchmarking from Bentley's Test Suite (n=1,000,000)

HotSpot	JDK 6 (Merge)	JDK 7 (TimSort)
Client	100%	**43%**
Server	100%	**26%**

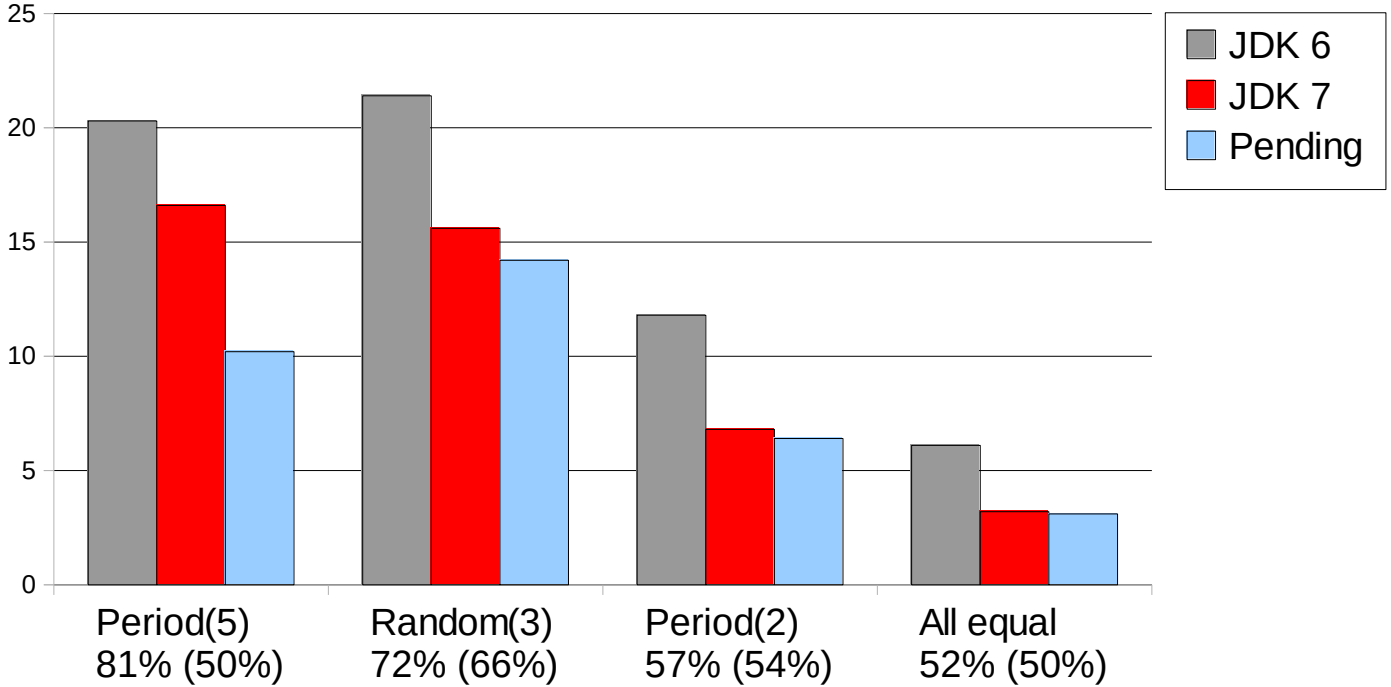
Results courtesy of Vladimir Yaroslavskiy

Dual Pivot Quicksort Performance



Results courtesy of Vladimir Yaroslavskiy

Dual Pivot Quicksort Performance



Results courtesy of Vladimir Yaroslavskiy

Dual-Pivot Quicksort Summary

- Benchmarking from Bentley's Test Suite (n=1,000,000)

HotSpot	JDK 6 (Bentley)	JDK 7 (Dual-Pivot)
Client	100%	**43%**
Server	100%	**28%**

Results courtesy of Vladimir Yaroslavskiy

Sorting Comparators

It was there all along

```
java.lang.IllegalArgumentException: Comparison method
violates its general contract!
    at java.util.TimSort.mergeHi(TimSort.java:868)
    at java.util.TimSort.mergeAt(TimSort.java:485)
    at java.util.TimSort.mergeCollapse(TimSort.java:408)
    at java.util.TimSort.sort(TimSort.java:214)
    at java.util.TimSort.sort(TimSort.java:173)
    at java.util.Arrays.sort(Arrays.java:659)
    at java.util.Collections.sort(Collections.java:217)
    ... your code ...
```

Sorting Comparators

Beware the overflow

```
class FooComparator implements Comparator<Foo> {
    public int compare(Foo o1, Foo o2) {
        return o1.serial – o2.serial;
    }
}
```

o1.serial	o2.serial	Result	Valid
1	1	0	✔
1	2	-1	✔
2	1	1	✔
-1	-2	1	✔
-2,000,000,000	-2,000,000,000	0	✔
2,000,000,000	-2,000,000,000	-294967296	✘

Sorting Comparators

Obey all signs

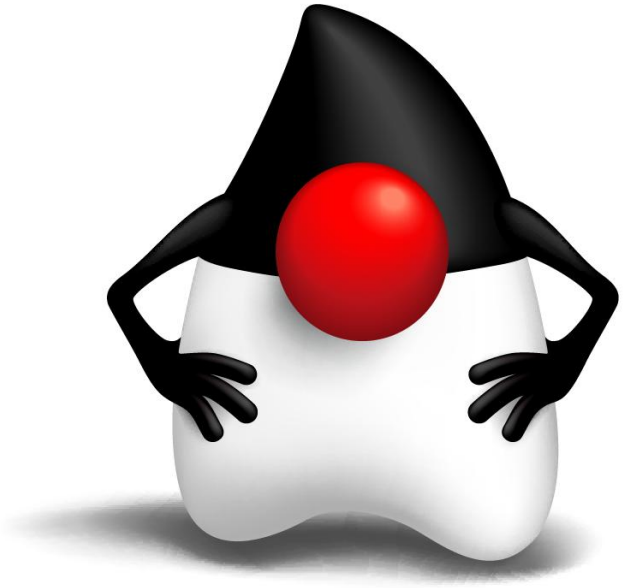
```
class FooComparator implements Comparator<Foo> {
    public int compare(Foo o1, Foo o2) {
        return o1.serial – o2.serial;
    }
}
```

- o1 > o2 → o2 < o1
- o1 == o2 → o2 == o1
- o1 == o2 → compare(o2,o3) == compare(o1,o3)
- o1 == o2 → o1.equals(o2) ***optional***

Alternative String Hashing

- Hash based Map performance depends strongly upon hash code quality
- Hash code collisions can lead to O(n) or worse performance
- Improved hashing for java.lang.String keys with HashMap, LinkedHashMap, HashSet, Hashtable, WeakHashMap, ConcurrentHashMap
- Optional behaviour added in Java 7u6. Not optional in Java 8
- Optional because iterator order becomes unpredictable
  - Smaller the map more likely it's iteration order is assumed

Wrap-Up



More Information — Java SE 7

- Main Java page
  - http://www.oracle.com/technetwork/java/index.html
- Downloads
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Features and enhancements (including Project Coin)
  - http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html

More Information — NIO

Java™ Tutorials
http://download.oracle.com/javase/tutorial/essential/io/fileio.html

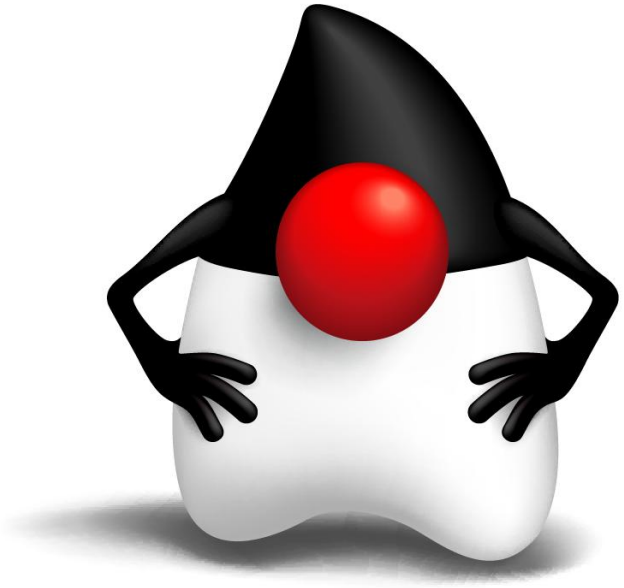
OpenJDK Project
http://openjdk.java.net/projects/nio

Mailing lists
nio-dev@openjdk.java.net
nio-discuss@openjdk.java.net

Summary

- JDK 7 is ready to use today!
  - Many productivity benefits in the language and library
  - Now on the Mac too
  - Regular updates every few months
- JDK 7 update releases
  - http://jdk7.java.net/
  - http://openjdk.java.net/projects/jdk7u/

Any Questions?



MAKE THE FUTURE JAVA

ORACLE

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.


JavaOne™