
ORACLE®

OpenJDK

ORACLE

Coin in Action

Using Java 7 Language Features in Real Code

Stuart W. Marks — JDK Core Libraries

O'REILLY OSCON java open source convention

import static com.oracle.disclaimer.*;

The following 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.

Java 7 Features Illustrated Today

- Project Coin
  1. Diamond
  2. Try-with-resources
  3. Multi-catch with more precise rethrow
  4. Enhanced integer literals
  5. Strings in switch
  6. Safe varargs
- Other Java 7 Features
  - NIO.2 utilities

Java 7 Features Illustrated Today

- Project Coin
  1. Diamond
  2. Try-with-resources
  3. Multi-catch with more precise rethrow
  4. Enhanced integer literals
  5. Strings in switch
  6. Safe varargs
- Other Java 7 Features
  - NIO.2 utilities

Scenario – URLJarFile.java

- "Based on a true story"
- Coinification: apply Project Coin features to the JDK
  - Concentrated mostly on core libraries
- Bug 7018392
  - "update URLJarFile.java to use try-with-resources"
    - http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=7018392
  - Fix integrated into JDK 7
    - http://hg.openjdk.java.net/jdk7/jdk7/jdk/rev/6e33b377aa6e
  - Simplified for purposes of presentation

Requirements for URLJarFile.retrieve()

- Given a URL …
  - Open it
  - Download contents into a temporary file
  - Create and return a JarFile instance backed by that temp file
  - Remove temp file if there was an error
  - Don’t leak anything
  - Handle all errors without loss of information

Original Code (One Slide)

```
JarFile retrieve(URL url) throws IOException {
    InputStream in = url.openStream();
    OutputStream out = null;
    File tmpFile = null;
    try {
        tmpFile = File.createTempFile("jar_cache", null);
        out = new FileOutputStream(tmpFile);
        int read = 0;
        byte[] buf = new byte[BUF_SIZE];
        while ((read = in.read(buf)) != -1) {
            out.write(buf, 0, read);
        }
        out.close();
        out = null;
        return new JarFile(tmpFile);
    } catch (IOException e) {
        if (tmpFile != null) {
            tmpFile.delete();
        }
        throw e;
    } finally {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
    }
}
```


Original Code (1 of 3)

```
JarFile retrieve(URL url) throws IOException {
    InputStream in = url.openStream();
    OutputStream out = null;
    File tmpFile = null;
    try {
        tmpFile = File.createTempFile("jar_cache", null);
        out = new FileOutputStream(tmpFile);
        ...
```


Original Code (2 of 3)

```
...
int read = 0;
byte[] buf = new byte[BUF_SIZE];
while ((read = in.read(buf)) != -1) {
    out.write(buf, 0, read);
}
out.close();
out = null;
return new JarFile(tmpFile);
...
```


Original Code (3 of 3)

```
            ...
        } catch (IOException e) {
            if (tmpFile != null) {
                tmpFile.delete();
            }
            throw e;
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
}
```


Code Review

- Has bugs!
  - If in.close() fails, ***out*** will remain open
  - If non-IOException is thrown, temp file will not be deleted
  - Suppressed exceptions are mishandled
- Other Issues
  - Uses null references to keep track of what needs cleanup
  - Messy, but alternative is to use nested try-statements
    ... which is arguably worse
  - Pathology: trying to do too much in a single try/catch/finally block

Improvement #1 – Use NIO

- Replace copy loop with Files.copy(InputStream, Path)
- Add various java.nio.file.* imports
- Change java.io.File to java.nio.file.Path
- Call Path.toFile() where necessary to convert back to java.io.File
- Get rid of BUF_SIZE and OutputStream variables
- Use Files.copy(..., REPLACE_EXISTING)

Improvement #1 – Use NIO

Allows us to replace this...

```
out = new FileOutputStream(tmpFile);
int read = 0;
byte[] buf = new byte[BUF_SIZE];
while ((read = in.read(buf)) != -1) {
    out.write(buf, 0, read);
}
out.close();
```

With this...

```
Files.copy(in, temp, REPLACE_EXISTING);
```


Improvement #1 – Use NIO (1 of 2)

```
JarFile retrieve(URL url) throws IOException {
    InputStream in = url.openStream();
    Path tmpFile = null;
    try {
        tmpFile = Files.createTempFile("jar_cache", null);
        Files.copy(in, tmpFile, REPLACE_EXISTING);
        return new JarFile(tmpFile.toFile());
        ...
```


Improvement #1 – Use NIO (2 of 2)

```
        ...
        } catch (IOException e) {
            if (tmpFile != null) {
                Files.delete(tmpFile);
            }
            throw e;
        } finally {
            if (in != null) {
                in.close();
            }
        }
}
```


Improvement #2 – Use Try-With-Resources

- Declares a resource variable
  - Automatically closed within a finally block
  - Ignored if null
- Suppressed exceptions from close() are added to a suppressed exception list
- Lets us drop our own finally block

Improvement #2 – Use Try-With-Resources

```
JarFile retrieve(URL url) throws IOException {
    Path tmpFile = null;
    try (InputStream in = url.openStream()) {
        tmpFile = Files.createTempFile("jar_cache", null);
        Files.copy(in, tmpFile, REPLACE_EXISTING);
        return new JarFile(tmpFile.toFile());
    } catch (IOException e) {
        if (tmpFile != null) {
            Files.delete(tmpFile);
        }
        throw e;
    }
}
```


Improvement #3 – Get Rid of Null Sentinel

- The ***in*** and ***out*** resources are handled for us now
  - ***in*** is a resource variable
  - ***out*** is buried inside of Files.copy()
- We can create the temp file first and get rid of special case null handling

Improvement #3 – Get Rid of Null Sentinel

```
JarFile retrieve(URL url) throws IOException {
    Path tmpFile = Files.createTempFile("jar_cache", null);
    try (InputStream in = url.openStream()) {
        Files.copy(in, tmpFile, REPLACE_EXISTING);
        return new JarFile(tmpFile.toFile());
    } catch (IOException e) {
        Files.delete(tmpFile);
        throw e;
    }
}
```


Improvement #4 – Catch/Rethrow Throwable

- We want to delete the temp file on any error
  - Catch and rethrow ***Throwable***
  - The method still declares ***throws IOException***
  - How is this possible?
- This is the “more precise rethrow” feature of Java 7
  - If the catch block simply rethrows a caught exception,
  - The checked exceptions that ***can be thrown*** from the catch block are inferred from what ***can be thrown*** by the try block.
- ***A subtle but significant change in Java 7!***

Improvement #4 – Catch/Rethrow Throwable

```
JarFile retrieve(URL url) throws IOException {
    Path tmpFile = Files.createTempFile("jar_cache", null);
    try (InputStream in = url.openStream()) {
        Files.copy(in, tmpFile, REPLACE_EXISTING);
        return new JarFile(tmpFile.toFile());
    } catch (Throwable t) {
        Files.delete(tmpFile);
        throw t;
    }
}
```

Improvement #5 – Suppressed Exceptions

- An exception from Files.delete() could still suppress an earlier exception
- Add explicit code to catch them and add them to the suppressed exception list
- Code gets a bit longer but closes a big hole in exception handling

Improvement #5 – Suppressed Exceptions

```
JarFile retrieve(URL url) throws IOException {
    Path tmpFile = Files.createTempFile("jar_cache", null);
    try (InputStream in = url.openStream()) {
        Files.copy(in, tmpFile, REPLACE_EXISTING);
        return new JarFile(tmpFile.toFile());
    } catch (Throwable t) {
        try {
            Files.delete(tmpFile);
        } catch (Throwable t2) {
            t.addSuppressed(t2);
        }
        throw t;
    }
}
```


Before vs After

```
JarFile retrieve(URL url) throws IOException {
    InputStream in = url.openStream();
    OutputStream out = null;
    File tmpFile = null;
    try {
        tmpFile = File.createTempFile("jar_cache", null);
        out = new FileOutputStream(tmpFile);
        int read = 0;
        byte[] buf = new byte[BUF_SIZE];
        while ((read = in.read(buf)) != -1) {
            out.write(buf, 0, read);
        }
        out.close();
        out = null;
        return new JarFile(tmpFile);
    } catch (IOException e) {
        if (tmpFile != null) {
            tmpFile.delete();
        }
        throw e;
    } finally {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
    }
}
```

```
JarFile retrieve(URL url) throws IOException {
    Path tmpFile = Files.createTempFile("jar_cache", null);
    try (InputStream in = url.openStream()) {
        Files.copy(in, tmpFile, REPLACE_EXISTING);
        return new JarFile(tmpFile.toFile());
    } catch (Throwable t) {
        try {
            Files.delete(tmpFile);
        } catch (Throwable t2) {
            t.addSuppressed(t2);
        }
        throw t;
    }
}
```

Summary

- Java 7 Features Demonstrated
  - NIO Files utilities
  - Try-with-resources
  - More precise rethrow (separate from multi-catch)
- Benefits
  - Code gets ***more concise*** and ***more correct***

What You Should Do Next

- Learn about Java 7 features:
  - There's lots of stuff I haven't mentioned today
  - http://download.oracle.com/javase/7/docs/index.html
- Download JDK 7 and try it out!
  - http://jdk7.java.net/

Q&A

Hardware and Software

ORACLE

Engineered to Work Together


ORACLE®