

Java SE 7 What's New

Eva Fulierová
Java Specialist
ciit (www.ciit.at)

9.10.2009 – Java

Day

Next Generation



- Modular Java Platform
- Support for Dynamically Typed Languages
- Java Language Syntax changes
- New File I/O (NIO.2)
- Swing API Additions
- Networking

- Sun + OpenJDK Community

Modularization

- Refactor the Java SE platform into smaller, separate, interdependent modules.
- Download size
- Start-up performance
- Runtime size
- JAR file concept drawbacks

Modularization

- Declaring that a class belongs to a module:

```
module M;  
package P;  
  
public class Foo {...}
```

- Defining a module in `module-info.java` file:

```
module M @1.0 {  
    requires N @2.1;  
    requires L @0.5;  
}
```

Support for Dynamically Typed Languages

- VM can execute every program that comes in a suitable `.class` format.
- It's difficult to compile a dynamically typed language into the Java VM bytecode
 - due to the static type concept of Java language.
- New bytecode instructions for VM that don't need to have the type information filled out.
- This allows for an effective way to run dynamic languages on platform-independent VM

Dynamically Typed Languages

- Creators of the engines for other languages like [Ruby](#), [Python](#), [Groovy](#), [Scala](#) - they started creating the engines to run on the Java Platform

```
function max (x,y) {  
    if x.lessThan(y) then y else x  
}
```

```
MyObject function max (MyObject x, MyObject y) {  
    if x.lessThan(y) then y else x  
}
```

Dynamically Typed Languages

- Some API has to be added to Java SE supporting dynamically typed languages
- dynamic invocation (static call through interface `Dynamic`), general case;
- method handle invocation;
- exotic identifiers (“#”exotic name definition”);
- conversion rules for interface `Dynamic`;

Java Language Syntax

- Project Coin -
<http://openjdk.java.net/projects/coin/>
 - Strings in switch
 - Automatic Resource Management
 - Improved Type Inference for Generic Instance Creation
 - Simplified Varargs Method Invocation
 - An omnibus proposal for better integral literals
 - Language support for Collections

Java Language Syntax

- Left out:
 - Improved Exception Handling for Java

```
try {
    doWork(file);
} catch (final IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

- Elvis and Other Null-Safe Operators

```
String s = maybeNull?.toString() ?: "null";

Integer ival = ...; // may be null
int i = ival ?: -1; // no NPE from unboxing
```

Strings in switch

- Type Constants where String is enough
- No need to define Enum
- Fall-through feature as opposed to if-else if

```
String s = ...
switch(s) {
    case "quux":
        processQuux(s);
        // fall-through
    case "bar":
        processFooOrBar(s);
        break;
    default:
        processDefault(s);
        break;
}
```

Automatic Resource Management

- Manual closing of resources leads to frequent errors
- Even correct code can experience problems with handling exceptions
- Other languages:
 - C# has using blocks
 - C++ has destructors

Manual RM

```
static 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 RM

```
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);  
    }  
}
```

Improved Type Inference for Generic Instance Creation

- Limited type inference for class instance creation expressions

```
Map<String, List<String>> anagrams =  
    new HashMap<String, List<String>> ();
```

```
Map<String, List<String>> anagrams = new HashMap<> ();
```

Simplified Varargs Method Invocation

- Vararg method calls with a non-reifiable varargs element type:
 - Compiler warning message moved from the method call to method declaration

Before Java SE 7:

```
static <T> List<T> asList(T... elements) { ... }

static List<Callable<String>> stringFactories() {
    Callable<String> a, b, c;
    ...
    *// Warning: **"uses unchecked or unsafe operations"*
    return asList(a, b, c);
}
```

Simplified Varargs Method Invocation

- After the change:

```
*/ Warning: **"enables unsafe generic array creation"*
static <T> List<T> asList(T... elements) { ... }

static List<Callable<String>> stringFactories() {
    Callable<String> a, b, c;
    ...
    return asList(a, b, c);
}
```

- For a varargs argument of type T, the programmer can suppress the warning using `@SuppressWarnings("generic-varargs")`

An omnibus proposal for better integral literals

- Binary literals
- Underscores in numbers – for readability purposes

```
byte aByte = (byte)0b00100001;  
int anInt1 = 0b10100001010001011010000101000101;  
  
long creditCardNumber = 1234_5678_9012_3456L;  
long socialSecurityNumbers = 999_99_9999L;  
float monetaryAmount = 12_345_132.12;  
long hexBytes = 0xFF_EC_DE_5E;
```

Language support for Collections

- Introduces Collection literals

```
final Map<Integer, String> platonicSolids;

static {

    solids = new LinkedHashMap<Map<Integer, String>;

    solids.put(4, "tetrahedron");
    solids.put(6, "cube");
    solids.put(8, "octahedron");
    solids.put(12, "dodecahedron");
    solids.put(20, "icosahedron");

    platonicSolids = Collections.immutableMap(solids);

}
```

Language support for Collections

- Here's how it would look like using Map literals

```
final Map<Integer, String> platonicSolids = {  
    4 : "tetrahedron", 6 : "cube", 8 : "octahedron",  
    12 : "dodecahedron", 20 : "icosahedron"};
```

- Also allows indexing access for Lists and Maps

```
List<String> l1 = Arrays.asList(new String[] {"a", "b",  
"c"});  
Map<Integer, String> m1 = new HashMap<Integer, String>(4);  
Map<String, Integer> m2 = new HashMap<String, Integer>(4);  
  
m2[l1[2]] = m2[m1[1]] = 4;  
// m2.put(l1.get(2), m2.put(m1.get(1), 4));
```

File I/O in Java 7 (NIO.2)

- <http://java.sun.com/developer/technicalArticles/javase/nio/>
- New class “Path” due to the backwards compatibility.

The `java.io.File` class provides the `toPath` method, which converts an old style `File` instance to a `java.nio.file.Path` instance, as follows:

```
Path input = file.toPath();
```

- <http://java.sun.com/docs/books/tutorial/essential/io/legacy.html>

Motivation

- Many methods didn't throw Exceptions when they failed.
- The `rename` method didn't work consistently across platforms.
- There was no real support for symbolic links.

New Exceptions

directory: data/test.dat

```
File dataFile = new File("data");  
System.out.println(this.dataFile.delete());  
  
Path path = this.dataFile.toPath();  
path.delete();
```

output:
false

```
java.nio.file.DirectoryNotEmptyException: data  
    at  
sun.nio.fs.WindowsPath.implDelete(WindowsPath.java:977)  
    at sun.nio.fs.AbstractPath.delete(AbstractPath.java:65)  
    at test.nio.DataSource.loadData(DataSource.java:22)  
    at test.nio.Test.main(Test.java:32)
```

File attributes

- More support for metadata was desired, such as file permissions, file owner, and other security attributes.
- Accessing file metadata was inefficient.
- **The File methods:** `isDirectory`, `isFile`, `setExecutable`, `setReadable`, `setReadOnly`, `lastModified`, `setLastModified`, `length`, `setWritable`. **Replaced by** `java.nio.file.attributes` package, which reads the attributes in a bulk operation.

File attributes example

```
Path file = ...;
BasicFileAttributes attr =
Attributes.readBasicFileAttributes(file);

if (attr.creationTime() != null) {
    System.out.println("creationTime: " +
attr.creationTime());
}
```

Directory listing

- Many `File` operations didn't scale.
- `File.list` and `listFiles` are replaced with `Path.newDirectoryStream`
- This approach scales well to very large directories.

Directory listing

```
Path dir = ...;
DirectoryStream<Path> stream = null;
try {
    stream = dir.newDirectoryStream();
    for (Path file: stream) {
        System.out.println(file.getName());
    }
} catch (IOException x) {
    //IOException can never be thrown by the iteration.
    //In this snippet, it can only be thrown by
newDirectoryStream.
    System.err.println(x);
} finally {
    if (stream != null) stream.close();
}
```

Filtering using globbing:

```
stream = dir.newDirectoryStream("*. {java.class,jar}");
```

Recursive directory walk

- It was not possible to write reliable code that could recursively walk a file tree and respond appropriately if there were circular symbolic links

```
Files.walkFileTree(Path start,  
                  FileVisitor<? super Path> visitor)
```

```
Files.walkFileTree(Path start,  
                  Set<FileVisitOption> options, int maxDepth,  
                  FileVisitor<? super Path> visitor)
```

Watch Service

- Now, it's possible to listen to the File System changes.
- Create a “watcher”. Register a directory with the “watcher” and specify the event type – this returns a WatchKey instance.
- You can listen for events in an endless loop.

Swing API Additions

- Small additions to the Swing API including the JXLayer component decorator and JXDatePicker
- JXDatePicker combines a button, an editable field and a JXMonthView component.

```
final JXDatePicker datePicker = new
JXDatePicker(System.currentTimeMillis());
datePicker.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        label.setText(datePicker.getDate().toString());
    }
});
```

JXLayer decorator

- Don't like verbose `AWTEventListeners`?
- Got tired of using `frame's glassPane`?
- Going to write another custom `RepaintManager`, but don't want to break compatibility with one from `SwingX`?

- With `JXLayer` you can easily decorate your compound components and catch all `Mouse`, `Keyboard` and `FocusEvent` for all its subcomponents.

JXLayer example

To enable auto-scrolling:

```
JScrollPane sp = JScrollPane(createMyTable());  
JXLayer<JScrollPane> l = new JXLayer<JScrollPane>(sp,  
    new MouseScrollableUI());  
add(l);
```

Recursively lockable component:

```
LockableUI lockableUI = new LockableUI();  
// wrap the panel with JXLayer and the lockableUI  
JXLayer<JComponent> l = new JXLayer<JComponent>(panel,  
    lockableUI);  
// lock the layer  
lockableUI.setLocked(true);  
  
// add the layer as any other component  
frame.add(l);
```

Networking

- Sockets Direct Protocol (SDP) for Solaris OS (supported from Solaris OS 10)
- Transmits data over InfiniBand (IB) switched fabric communications link
- Remote Direct Memory Access (RDMA)
- No code adjustments, just configuration file

Networking

- Packages that support networking over SDP:
 - **java.net** package
 - Socket
 - ServerSocket
 - **java.nio.channels** package:
 - SocketChannel
 - ServerSocketChannel
 - AsynchronousSocketChannel
 - AsynchronousServerSocketChannel