Java 7 Highlights

Presented By: Andrey Loskutov

Agenda

- **Motivation**
- Java Programming Language Changes Highlights
    - Binary Literals
    - Underscores in Numeric Literals
    - Strings in switch Statements
    - The try-with-resources Statement
    - Catching Multiple Exception Types
    - Rethrowing Exceptions with Improved Type Checking
    - Type Inference for Generic Instance Creation
- API Changes Highlights
    - IO and New IO
    - Misc
- Known issues

Motivation

Java release 7 represents (from Java 6 point of view):

- new platform vendor (Oracle)
- 5 years work on the next major Java release
- all fixes and features of ~20 minor Java 6 releases
- new G1 garbage collector
- compressed 64-bit pointers
- better support for scripting languages ("invokedynamic")
- Java bytecode specification changes
- Java language syntax changes
- lot of new API's
- various other smaller enhancements…

*Note for C/C++ programmers:
… code written for Java 1.1 (1997) is still supported on 1.7 (2012)

Agenda

- Motivation
- **Java Programming Language Changes Highlights**
  - Binary Literals
  - Underscores in Numeric Literals
  - Strings in switch Statements
  - The try-with-resources Statement
  - Catching Multiple Exception Types
  - Rethrowing Exceptions with Improved Type Checking
  - Type Inference for Generic Instance Creation
- API Changes Highlights
  - IO and New IO
  - Misc
- Known issues

Binary Literals

You can now express integral type numbers in binary number system by using only 0 and 1.

All what you need is to start the value with 0b or 0B:

```
byte b  =  0b01001001;
short s =  0B00100100;
int i   =  0b10010010;
long l  =  0B01001001;
```

The reason is usability: it is much easier to use binary numbers while manipulating bits. For example, you can easily see that each number in the series above is rotated by one bit.

Now try to see this with decimals: 73, 36, 146, 73 ☺

Underscores in Numeric Literals

- Java 7 allows to put underscores anywhere **between digits** in a numerical literal:

```
short answer = 0b1_0_1_0_1_0;
int twoMonkeys = 0xAFFE_AFFE;
long smallestLong = 0x8000_0000_0000_0000L;
long somePhoneNumber = +49_7031_4357_0L;
double e =  2.718_281_829D;
```

- Same reason as before: improving code readability

Strings in switch Statements

You can now use a String object in the expression of a switch statement.

```
public static void main(String[] args) {
    String os = System.getProperty("os.name");
    switch (os) {
    case "Linux":
        System.out.println("Cool!");
        break;
    case "Windows":
        System.out.println("Not so cool!");
        break;
    default:
        System.out.println("Obst?");
        break;
    }
}
```

•For string matching String.equals() is used, so switch on strings is case sensitive

•null strings will cause NullPointerException (surprise!)

try-with-resources

Let's remember how we write to files in Java 6:

```
FileWriter fw = null;
try {
    fw = new FileWriter(file);
    fw.write("Hello Java 6!");
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if(fw != null){
        try {
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- 15 lines of code (and 19 lines to read this file back)!
- This is mostly due the standard try/finally pattern for properly closing resources in Java 6.

try-with-resources (continued)

Welcome to the Java 7 world:

- write to file (5 lines instead of 15):

```
try (FileWriter fw =  new FileWriter(file)) {
    fw.write("Hello Java 7!");
} catch (IOException e) {
    e.printStackTrace();
}
```

- read from file (8 lines instead of 19):

```
try (BufferedReader br = new BufferedReader(new FileReader(file))){
    String line;
    while((line = br.readLine()) != null){
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

- same as above in 1 line (using new Files API):

```
Files.copy(file.toPath(), System.err);
```

Cool, isn't?

try-with-resources (how it works #1)

- In try() statement one can declare one or more resources.
- Try-with-resources ensures that each resource is closed at the end of the try{} block.
- A resource must implement java.lang.AutoCloseable
- java.io.Closeable interface extends AutoCloseable since 1.7, e.g. most streams can be used now in try() statement
- Any catch or finally block is run **after** the declared resources have been closed

try-with-resources (how it works #2)

- **Note**: in case **multiple** resources are declared in try(), they are closed in the **opposite** definition order:

```
try (
    FileReader from = new FileReader(f1);
    FileWriter to = new FileWriter(f2)) {
    int data;
    while ((data = from.read()) != -1) {
        to.write(data);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

- In example above "to" stream is closed first, "from" second.

try-with-resources (how it works #3)

- In case exceptions are thrown in **both** try() **and** try{} blocks, then those exceptions from try() are suppressed and those from try{} are thrown:

```
try (FileWriter fw = new BadWriter(file)) {
    fw.write("O-o!");
} catch (IOException e) {
    System.err.println("Thrown: " + e.getMessage());
    Throwable[] suppressed = e.getSuppressed();
    for (Throwable t : suppressed) {
        System.err.println("Suppressed: " + t.getMessage());
    }
}

// output:
Thrown: Failed to write!
Suppressed: Failed to close!

class BadWriter extends FileWriter {
    private BadWriter(File file) throws IOException {
        super(file);
    }
    @Override
    public void close() throws IOException {
        throw new IOException("Failed to close!");
    }
    @Override
    public void write(String s) throws IOException {
        throw new IOException("Failed to write!");
    }
}
```

Catching Multiple Exception Types

- Let's remember the old good way to hack reflection code in Java 6 (please don't do it at home):

```
try {
    Field value = Integer.class.getDeclaredField("value");
    value.setAccessible(true);
    Integer obj = new Integer(42);
    System.out.println(obj);
    value.set(obj, -1);
    System.out.println(obj);
} catch (SecurityException e) {
    e.printStackTrace();
} catch (NoSuchFieldException e) {
    e.printStackTrace();
} catch (IllegalArgumentException e) {
    e.printStackTrace();
} catch (IllegalAccessException e) {
    e.printStackTrace();
}
```

- 16 lines code (to do impossible things ☺)

Catching Multiple Exception Types

- With Java 7 you are can do it in 10 lines:

```
try {
    Field value = Integer.class.getDeclaredField("value");
    value.setAccessible(true);
    Integer obj = new Integer(42);
    System.out.println(obj);
    value.set(obj, -1);
    System.out.println(obj);
} catch (SecurityException | NoSuchFieldException |
       IllegalArgumentException | IllegalAccessException e) {
    e.printStackTrace();
}
```

Rethrowing Exceptions with Improved Type Checking

- With Java 6:

```
void oldWayRethrow(boolean notFound) throws IOException {
    try {
        if (notFound) {
            throw new FileNotFoundException();
        } else {
            throw new MalformedURLException();
        }
    } catch (IOException e) {
        throw e;
    }
}
```

- With Java 7 you can be more specific:

```
void java7Rethrow(boolean notFound) throws FileNotFoundException,
MalformedURLException {
    try {
        if (notFound) {
            throw new FileNotFoundException();
        } else {
            throw new MalformedURLException();
        }
    } catch (IOException e) {
        throw e; // 1.6 compiler error:
                 // "unhandled exception type IOException"
    }
}
```

Type Inference for Generic Instance Creation

ADVANTEST®

- Let's write some generics with Java 6:

```
Map<Class<Number>, Map<String, List<Number>>> map =
   new HashMap<Class<Number>, Map<String,List<Number>>>();
```

- With Java 7 you are much faster:

```
Map<Class<Number>, Map<String, List<Number>>> map =
   new HashMap<>(); // "diamond" operator!
```

- … but code below still generates two warnings ☹

```
Map<Class<Number>, Map<String, List<Number>>> map =
   new HashMap();
// 2 compiler warnings:

// "References to generic type should be parameterized",
// "Expression needs unchecked conversion"
```

Agenda

ADVANTEST®

- Motivation
- Java Programming Language Changes Highlights
  - Binary Literals
  - Underscores in Numeric Literals
  - Strings in switch Statements
  - The try-with-resources Statement
  - Catching Multiple Exception Types
  - Rethrowing Exceptions with Improved Type Checking
  - Type Inference for Generic Instance Creation
- **API Changes Highlights**
  - IO and New IO
  - Misc
- Known issues

API Changes: IO and new IO

New convenient API for dealing with paths and files:

- java.nio.file.Path (lot of useful path manipulation API)
- java.nio.file.Paths (creates Path's from strings and URL's)
- java.nio.file.Files (all things you missed in java.io.File)

You can now

- have full access to symlinks (create/resolve/check)
- manage **all** file attributes via FileAttribute
- check global file system attributes via FileSystem/FileStore
- list directories with DirectoryStream
- walk file trees with FileVisitor
- copy files to streams (Files.copy(file, System.out))
- watch file system for changes (WatchService)
- create **virtual** file systems (FileSystemProvider)
- query file content type (FileTypeDetector)

IO and new IO examples

ADVANTEST

Basic stuff

```
Path path = Files.createTempFile(null, ".txt");
Files.write(path, "Hello\n".getBytes());
Path link = path.getParent().resolve("link");
Files.deleteIfExists(link);
Path symlink = Files.createSymbolicLink(link, path);

out.println("Real file: " + path);
out.println("Link file: " + symlink);
out.println("Is link? " + Files.isSymbolicLink(symlink));
out.println("Link target: " + Files.readSymbolicLink(symlink));
out.println("Content: " + Files.readAllLines(path,
Charset.defaultCharset()));
out.println("Content type: " + Files.probeContentType(path));

// output
Real file: /tmp/8009678549582989860.txt
Link file: /tmp/link
Is link? true
Link target: /tmp/8009678549582989860.txt
Content: [Hello]
Content type: text/plain
```

IO and new IO examples

File attributes

```
// classic command line
Set<PosixFilePermission> permissions = fromString("rwxrwxrwx");

// object oriented way
permissions = asList(PosixFilePermission.values());

Files.createFile(onlyForMe, PosixFilePermissions.asFileAttribute(permissions));

Set<PosixFilePermission> freeAccess = Files.getPosixFilePermissions(onlyForMe);
System.out.println(freeAccess);

freeAccess.removeAll(asList(GROUP_WRITE, OTHERS_READ, OTHERS_EXECUTE));

PosixFileAttributeView attributeView = Files.getFileAttributeView(onlyForMe,
PosixFileAttributeView.class);
attributeView.setPermissions(freeAccess);

PosixFileAttributes fileAttributes = attributeView.readAttributes();
System.out.println(fileAttributes.permissions());
System.out.println("Current owner: " + attributeView.getOwner());

BasicFileAttributeView basic = Files.getFileAttributeView(onlyForMe,
BasicFileAttributeView.class);
// prints device id and inode on Linux
System.out.println(basic.readAttributes().fileKey());

// output
[OWNER_READ, OWNER_WRITE, OWNER_EXECUTE, GROUP_READ, GROUP_WRITE, GROUP_EXECUTE,
OTHERS_READ, OTHERS_EXECUTE]
[OWNER_READ, OWNER_WRITE, OWNER_EXECUTE, GROUP_READ, GROUP_EXECUTE]
Current owner: aloskuto
(dev=802,ino=26312794)
```

IO and new IO examples

DirectoryStream (old way: File.list(FilenameFilter))

```
// simple name based filter
try (DirectoryStream<Path> stream =
Files.newDirectoryStream(dir, "*.{tmp,test}")) {
    for (Path path : stream) {
        System.out.println(path);
    }
}

// filter based on file matchers and file attributes
FileSystem fs = FileSystems.getDefault();
final PathMatcher regexMatcher =
fs.getPathMatcher("regex:.*7\\.\\d+.*");
final PathMatcher globMatcher = fs.getPathMatcher("glob:/tmp/*.*");

Filter<? super Path> filter = new Filter<Path>() {
    @Override
    public boolean accept(Path path) throws IOException {
        return globMatcher.matches(path) &&
                regexMatcher.matches(path) &&
                !Files.isSymbolicLink(path);
    }
};
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir,
filter)) {
    for (Path path : stream) {
        System.out.println(path);
    }
}
```

IO and new IO examples

FileVisitor and Files.walkFileTree()

- implemented as depth-first preorder traversal
- visits a directory before visiting any of its descendants

```
FileVisitor<? super Path> visitor = new MySimpleFileVisitor();
try {
    Files.walkFileTree(path, visitor);
} catch (IOException e) {
    System.out.println("Failed to walk: " + e.getMessage());
}

class MySimpleFileVisitor implements FileVisitor<Path> {
    @Override
    public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs)
    throws IOException {
        return FileVisitResult.CONTINUE;
    }

    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)
            throws IOException {
        return FileVisitResult.CONTINUE;
    }

    @Override
    public FileVisitResult visitFileFailed(Path file, IOException exc)
            throws IOException {
        return FileVisitResult.SKIP_SUBTREE;
    }

    @Override
    public FileVisitResult postVisitDirectory(Path dir, IOException exc)
            throws IOException {
        return FileVisitResult.TERMINATE;
    }
}
```

IO and new IO examples

WatchService (has to be polled for events)

```
Path path = Paths.get(System.getProperty("java.io.tmpdir"));
WatchService watchService = path.getFileSystem().newWatchService();
WatchKey watchKey = path.register(watchService,
    OVERFLOW, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);
Path tempFile = Files.createTempFile(path, "", ".tmp");
Path path2 = Paths.get(tempFile + "_moved");
Files.move(tempFile, path2);
Files.write(path2, "Hello".getBytes());
Files.deleteIfExists(path2);
printEvents(watchKey);
watchKey.cancel();

void printEvents(WatchKey watchKey) {
    List<WatchEvent<?>> events = watchKey.pollEvents();
    for (WatchEvent<?> event : events) {
        System.out.println("-> " + event.count() + " event(s):");
        Object context = event.context();
        if(context instanceof Path){
            Path path = (Path) context;
            System.out.print("\tPath: " + path);
        }
        System.out.println("\tKind: " + event.kind());
    }
}

// output:
-> 1 event(s):
    Path: 7788862439291078942.tmp    Kind: ENTRY_CREATE
-> 1 event(s):
    Path: 7788862439291078942.tmp    Kind: ENTRY_DELETE
-> 1 event(s):
    Path: 7788862439291078942.tmp_moved        Kind: ENTRY_CREATE
-> 2 event(s):
    Path: 7788862439291078942.tmp_moved        Kind: ENTRY_MODIFY
-> 1 event(s):
    Path: 7788862439291078942.tmp_moved        Kind: ENTRY_DELETE
```

IO and new IO examples

java.nio.file.spi.FileSystemProvider

```
List<FileSystemProvider> providers = FileSystemProvider.installedProviders();
for (FileSystemProvider fsProvider : providers) {
    System.out.println("sheme: '" + fsProvider.getScheme() + "', provider: " +
fsProvider.getClass());
}

Path tmpFile = Files.createTempFile("", ".tmp");
Files.write(tmpFile, "Hello".getBytes());
Path jarFile = Files.createTempFile("", ".jar");

try(JarOutputStream outputStream = new JarOutputStream(Files.newOutputStream(jarFile))){
    outputStream.putNextEntry(new ZipEntry(tmpFile.getFileName().toString()));
    Files.copy(tmpFile, outputStream);
}

try(FileSystem fileSystem = createVirtualFS(jarFile)){
    Iterable<Path> directories = fileSystem.getRootDirectories();
    for (Path dir : directories) {
        System.out.println("Reading dir: " + dir.toUri());
        DirectoryStream<Path> stream = Files.newDirectoryStream(dir);
        for (Path file : stream) {
            System.out.println("Reading file: " + file.toUri());
            System.out.println("\tfrom " + file.getFileSystem().provider().getClass());
            System.out.print("Content: ");
            Files.copy(file, System.out);
        }
    }
}
FileSystem createVirtualFS(Path jarFile) throws IOException {
    return FileSystems.newFileSystem(jarFile, FileSystemProviderAPI.class.getClassLoader());
}
// output:
sheme: 'file', provider: class sun.nio.fs.LinuxFileSystemProvider
sheme: 'jar', provider: class com.sun.nio.zipfs.ZipFileSystemProvider
Reading dir: jar:file:///tmp/6394383027288508157.jar!/
Reading file: jar:file:///tmp/6394383027288508157.jar!/3853639990503364043.tmp
    from class com.sun.nio.zipfs.ZipFileSystemProvider
Content: Hello
```

Misc

java.util.concurrent.ForkJoinPool

java.util.concurrent.ForkJoinTask<V>

•simple to use fork/join framework

•uses worker pool and workers do "work stealing"

```
List<Integer> findAll(List<String> list, String key) {
    ForkJoinPool pool = new ForkJoinPool();
    FindTask task = new FindTask(list, key, 0);
    return pool.invoke(task);
}
```

java.util.concurrent.Phaser

•reusable synchronization barrier

•more flexible than CyclicBarrier or CountDownLatch

Misc

java.util.Objects

```
String [] array1 = {"a", "b", null};
String [] array2 = {"a", "b", null};

System.out.println(array1.equals(array2));

System.out.println(Objects.equals(array1, array2));
System.out.println(Objects.deepEquals(array1, array2));
System.out.println(Objects.hash(array1, array2));

array1[2] = Objects.requireNonNull(System.getProperty("undefined"),
   "No NULL please!");

// output

false
false
true
1763727812
Exception in thread "main" java.lang.NullPointerException: No NULL
please!
   at java.util.Objects.requireNonNull(Objects.java:226)
   at ObjectsAPI.main(ObjectsAPI.java:25)
```

Known problems

- Javac and ecj compiler have sometimes different opinion about valid Java code ☺

Links

- Official Java 7 Documentation

  http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html
- Tutorial new IO

  http://docs.oracle.com/javase/tutorial/essential/io/file.html
- Java History

  http://en.wikipedia.org/wiki/Java_version_history
- All examples and this presentation

git clone https://github.com/iloveeclipse/java7examples.git