What’s new in Java land

Michael Voelske
michael.voelske@uni-weimar.de

Bauhaus-Universitaet Weimar
Faculty of Media

August 7, 2012
Outline

Introduction

Java 7 core language changes

Selected standard library changes

Possible Java 8 features

Conclusions
Outline

Introduction
Java SE 7
Eclipse Juno

Java 7 core language changes

Selected standard library changes

Possible Java 8 features

Conclusions
Java SE 7

- Last release (Java SE 6) from December 2006
- Java SE 7 initially released in July 2011
- Many changes to language and platform
  - Core language changes ("Project Coin")
  - Additions to standard library
  - Some language features ("Project Lambda") left for Java SE 8
Outline

Introduction
Java SE 7
Eclipse Juno

Java 7 core language changes

Selected standard library changes

Possible Java 8 features

Conclusions
Eclipse Juno

- Version 4.2 (Juno) released June 27, 2012
- New features
  - Lua Development Tools
  - Code Recommenders
  - Xtend programming language
- Enhancements
  - UI Overhaul
  - JDT support for Java 7
  - Git integration
Outline

Introduction

Java 7 core language changes
  Generic type inference
    Strings in switch
    Numeric literals
    New features in exception handling

Selected standard library changes

Possible Java 8 features

Conclusions
Redundancy in generic constructor calls

Problem: type parameter must be specified twice

```java
1 // construct two lists with type parameter
2 List<String> stringlist = new ArrayList<String>();
3 List<Integer> intlist = new ArrayList<Integer>();
```
Redundancy in generic constructor calls

Problem: type parameter must be specified twice

```java
// construct two lists with type parameter
List<String> stringlist = new ArrayList<String>();
List<Integer> intlist = new ArrayList<Integer>();
```

Type parameters can be omitted

```java
// using the "raw type" constructor
intlist = new ArrayList();
```
Redundancy in generic constructor calls

Problem: type parameter must be specified twice

```java
// construct two lists with type parameter
List<String> stringlist = new ArrayList<String>();
List<Integer> intlist = new ArrayList<Integer>();
```

Type parameters can be omitted

```java
// using the "raw type" constructor
intlist = new ArrayList();
```

But we lose compile-time checks

```java
stringlist.add("foo");
// Copy constructor; legal at compile time:
intlist = new ArrayList(stringlist);
// ClassCastException at *run* time:
Integer item = intlist.get(0);
```
Enter the "diamond" operator

Diamond operator infers generic type parameters automatically

```java
List<String> stringlist = new ArrayList<>();
```
Enter the "diamond" operator

Diamond operator infers generic type parameters automatically

1
List<String> stringlist = new ArrayList <> ();

Retains compile-time checks

2
// This is a type mismatch error at *compile* time
3
List<Integer> intlist = new ArrayList <> (stringlist);
Outline

Introduction

Java 7 core language changes
  Generic type inference
  Strings in switch
  Numeric literals
  New features in exception handling

Selected standard library changes

Possible Java 8 features

Conclusions
Strings in switch statements

Alternatives based on string values in Java 1.6

```java
String hello(String string) {
    if ("Alice".equals(string)) {
        return "Hello, Alice!";
    } else if ("Bob".equals(string)) {
        return "Hello, Bob!";
    } else {
        return "Hello, Stranger!";
    }
}
```
Strings in switch statements (2)

Cleaner syntax in Java 1.7

```java
switch (string) {
    case "Alice":
        return "Hello, Alice!";
    case "Bob":
        return "Hello, Bob!";
    default:
        return "Hello, Stranger!";
}
```
Outline

Introduction

Java 7 core language changes
  Generic type inference
  Strings in switch
  Numeric literals
    New features in exception handling

Selected standard library changes

Possible Java 8 features

Conclusions
New syntax for integer literals

```java
// "0b"-prefix for base-2 integer literals
int four = 0b100;

// underscores for digit grouping
int billion = 1_000_000_000;
// anywhere *within* the digits
int million = 1_0_0_0_0___000;

// can be combined
int fortytwo = 0b0010_1010;
```
Outline

Introduction

Java 7 core language changes
  Generic type inference
  Strings in switch
  Numeric literals
  New features in exception handling

Selected standard library changes

Possible Java 8 features

Conclusions
Automatically closing resources

Problem: clean up opened resources in the presence of exceptions

```java
int firstChar(File f) throws IOException {
    FileReader fr = new FileReader(f);
    int c = fr.read() // may cause resource leak
    fr.close()
    return c;
}
```
Automatically closing resources

Problem: clean up opened resources in the presence of exceptions

```java
int firstChar(File f) throws IOException {
    FileReader fr = new FileReader(f);
    int c = fr.read() // may cause resource leak
    fr.close()
    return c;
}
```

```java
int firstChar(File f) throws IOException {
    FileReader fr = null;
    try {
        fr = new FileReader(f);
        return fr.read();
    } finally {
        if(fr != null) fr.close();
    }
}
```
Try-with-resources and AutoCloseable

New syntax in Java 1.7

```java
int firstChar(File f) throws IOException {
    try (FileReader fr = new FileReader(f)) {
        return fr.read();
    }
}
```
Try-with-resources and AutoCloseable

New syntax in Java 1.7

```java
int firstChar(File f) throws IOException {
    try (FileReader fr = new FileReader(f)) {
        return fr.read();
    }
}
```

```java
public interface AutoCloseable {
    void close() throws Exception;
}
```
Multi-catch blocks

Several exceptions can be handled in one catch block

```java
try (FileReader fr = new FileReader(file)) {
    Thread.sleep(1);
    return fr.read();
} catch (FileNotFoundException e) {
    // handle error
    System.err.println("File not found!");
} catch (IOException | InterruptedException e) {
    // ignore
    e.printStackTrace();
}
```
Outline

Introduction

Java 7 core language changes

Selected standard library changes

New file system API
  Asynchronous Channels
  Fork/Join pools

Possible Java 8 features

Conclusions
New file system API: Paths

- `java.nio.file` package supplements `java.io.File` class
- Motivation: better error handling, consistency across platforms and file systems

```java
// before 1.7:
File file = new File("data/tmp/test.txt");

// now:
Path path = Paths.get("data","tmp","test.txt");
```
New file system API: Paths

- java.nio.file package supplements java.io.File class
- Motivation: better error handling, consistency across platforms and file systems

```java
// before 1.7:
File file = new File("data/tmp/test.txt");
// now:
Path path = Paths.get("data","tmp","test.txt");
```

```java
if (!file.delete()) {
    // why didn't it work?...
}
```
New file system API: Paths

- `java.nio.file` package supplements `java.io.File` class
- Motivation: better error handling, consistency across platforms and file systems

```java
// before 1.7:
File file = new File("data/tmp/test.txt");
// now:
Path path = Paths.get("data","tmp","test.txt");

if (!file.delete()) {
    // why didn't it work?...
}

try {
    path.delete()
} catch (FileNotFoundException e) {
    // handle error
}
```
New file system API: WatchService

WatchService monitors file system changes
Outline

Introduction

Java 7 core language changes

Selected standard library changes

New file system API

Asynchronous Channels

Fork/Join pools

Possible Java 8 features

Conclusions
Asynchronous channel API

- Package `java.nio.channels`
- Non-blocking, asynchronous transfer of byte streams over network sockets or files
- `AsynchronousSocketChannel` object represents a connection; performs non-blocking I/O operations using background threads

```java
package java.nio.channels;
public abstract class AsynchronousSocketChannel
    implements AsynchronousByteChannel, NetworkChannel
{
    public abstract Future<Integer> read(ByteBuffer dst);
    public abstract Future<Integer> write(ByteBuffer src);
    /* ... */
}
```
Recap: Futures

- Asynchronous operations return Future objects that
  - Inspect state
  - Wait for completion
  - Retrieve results
  of the background thread

```java
package java.util.concurrent;
public interface Future<V> {
    /* ... */
    V get() throws InterruptedException, ExecutionException;
    boolean isDone();
    /* ... */
}
```
Outline

Introduction

Java 7 core language changes

Selected standard library changes
  New file system API
  Asynchronous Channels
  Fork/Join pools

Possible Java 8 features

Conclusions
Fork/Join pools

New parallelization framework, especially well suited to recursive divide-and-conquer algorithms

```java
package java.util.concurrent;
public class ForkJoinPool extends AbstractExecutorService {
    public ForkJoinPool(int parallelism);
    public <T> T invoke(ForkJoinTask<T> task);
    /* ... */
}
```
Fork/Join pools

New parallelization framework, especially well suited to recursive divide-and-conquer algorithms

```java
package java.util.concurrent;

public class ForkJoinPool extends AbstractExecutorService {

    public ForkJoinPool(int parallelism);

    public <T> T invoke(ForkJoinTask<T> task);
    /* ... */
}

public abstract class RecursiveTask<V> extends ForkJoinTask<V> {

    protected abstract V compute();
    /* ... */
}
```
Outline

Introduction

Java 7 core language changes

Selected standard library changes

Possible Java 8 features
  Lambdas

Conclusions
Many modern programming languages have syntax for function literals

Example: lambda keyword in Python

```python
list1 = [1,2,3,4,5]
list2 = map(lambda x: x * x, list1)
# list2 now contains [1,4,9,16,25]

# equivalent named function:
def square(x):
    return x * x
list2 = map(square, list1)
```
Use case for lambda functions in Java 8

Can’t pass function, but often "functional interfaces"

```java
public class File {
    /* ... */
    public File[] listFiles(FileFilter filter);
}
```

```java
public interface FileFilter {
    boolean accept(File pathname);
}
```
Use case for lambda functions in Java 8

Can’t pass function, but often "functional interfaces"

```java
public class File {
    /* ... */
    public File[] listFiles(FileFilter filter);
}
```

```java
public interface FileFilter {
    boolean accept(File pathname);
}
```

Often implemented as anonymous inline class

```java
File[] files = myDir.listFiles(new FileFilter(){
    @Override
    public boolean accept(File pathname) {
        return pathname.canExecute();
    }
});
```
Main use for lambda expressions will be implementing functional interfaces

```java
// statement syntax
FileFilter exists = (File f) -> f.exists();
File[] files = myDir.listFiles(exists);

// block syntax
files = myDir.listFiles((File f) -> {
  return f.canWrite();
});
```
Many standard library interfaces would fit this "functional" pattern

```
1 // interface:
2 // int Comparator<String>.compare(String s1, String s2)
3 Comparator<String> cmp =
4   (s1, s2) -> s1.compareToIgnoreCase(s2);

6 // interface:
7 // void Runnable.run()
8 new Thread(() -> {
9   performLongComputation();
10   System.out.println("Done!");
11 }) .start()
```
Conclusions

- Several new language features
  - Diamond operator
  - Switch statement with strings
  - Numeric literals
  - Exceptions
- Standard library additions
  - Paths/Files systems API
  - Asynchronous channels
  - Fork/Join pools
- Project Lambda will be released with version 8
  - Expected in Summer 2013
  - Will include bulk parallel collections APIs (Filter/Map/Reduce)
- Eclipse Juno
Further Reading

**OpenJDK**
Project Coin
http://openjdk.java.net/projects/coin/

**Oracle Corporation**
Java Programming Language Enhancements
http://docs.oracle.com/javase/7/docs/technotes/guides/language/enhancements.html#javase7

**Madhusudhan Konda**
A look at Java 7’s new features
O’Reilly Radar, September 2, 2011

**Alan Bateman et al**
JSR 203: More New I/O APIs for the Java Platform ("NIO.2")
Further Reading II

Catherine Hope and Oliver Deakin
An NIO.2 primer, Part 1: The asynchronous channel APIs

Brian Goetz et al.
JSR 335: Lambda Expressions for the Java Programming Language
http://www.jcp.org/en/jsr/detail?id=335

Brian Goetz
State of the Lambda
http://cr.openjdk.java.net/~briangoetz/lambda/lambda-state-4.html