Java 7 & 8

JWorks kickstartertraject 2016

Yannick De Turck

Java 7 (July 28th 2011)

- String in switch-statement
- Automatic resource management
- Diamond syntax
- Better Exception handling with multi-catch
- Literal enhancements
- New IO API
- Fork Join Framework
- JVM enhancements

Java 7: String in switch-statement

```
String dayPart = "evening";
switch (dayPart) {
    case MORNING:
        System.out.println("Good morning!");
        break;
    case NOON:
        System.out.println("Good afternoon!");
        break;
    case EVENING:
        System.out.println("Good evening");
        break;
    default:
        System.out.println("Good day!");
        break;
}
```

Java 7: Automatic resource management

- Try-with-resources statement
  - Resources are automatically closed

```
try (FileInputStream in2 = new FileInputStream(
    "/someplace/somewhere/file.txt")) {
    System.out.println(in2.read());
}
```

Java 7: Automatic resource management

- New `AutoCloseable` interface available to implement for your own classes

```
public class Deur implements AutoCloseable {
    @Override
    public void close() throws IOException {
        System.out.println("Deur toe");
    }
    public void open() {
        System.out.println("Deur is open");
    }
}
try(Deur deur = new Deur()){
    deur.open();
}
```

Java 7: Automatic resource management

```
// Java 6
File file = new File("leesmij.txt");
File file2 = new File("schrijfmij.txt");
InputStream in = null;
OutputStream out = null;
try {
    in = new FileInputStream(file);
    try {
        out = new FileOutputStream(file2);
    } catch (IOException e) {
    } finally {
        try {
            out.close();
        } catch (IOException e) {
        }
    }
} catch (IOException e) {
} finally {
    try {
        in.close();
    } catch (IOException e) {
    }
}
```

ORDINA

CONNECTIVATE

Java 7: Automatic resource management

```
// Java 7
try (InputStream in = new FileInputStream(file);
     OutputStream out = new FileOutputStream(outputFile)) {
} catch (IOException ex) {
    // Resources get automatically closed
   // when leaving the code block
}
```

Java 7: Diamond syntax

- Type Inference for Generic Instance Creation
- No longer required to repeat the type when instantiation

```
// Java 6
Map<String, Map<String, Integer>> map =
    new HashMap<String, Map<String, Integer>>();
List<String> strings = new ArrayList<String>();
Set<Integer> set = new HashSet<>();

// Java 7
Map<String, Map<String, Integer>> map2 = new
HashMap<>();
List<String> strings2 = new ArrayList<>();
Set<Integer> set2 = new HashSet<>();
```

Java 7: Better Exception handling with multi-catch

- No longer limited to one Exception per catch block

```
// Java 6
try {
    foo();
} catch (ClassNotFoundException ex) {
    // Handle Exception
} catch (NoSuchMethodException ex) {
    // Handle Exception
} catch (NoSuchFieldException ex) {
    // Handle Exception
}

// Java 7
try {
    foo();
} catch (ClassNotFoundException | NoSuchMethodException |
NoSuchFieldException ex) {
    // Handle Exception
}
```

Java 7: Improved checking for rethrown exceptions

- Precise rethrowing

```
public void doStuff() throws FileNotFoundException {
    try {
        throw new FileNotFoundException();
    } catch (IOException ex) {
        System.out.println("Throwing Exception...");
        throw ex;
    }
}


public class FileNotFoundException extends IOException
```

Java 7: Literal enhancements

- Prefix binary literals with `0b` or `0B`
- Use underscores in your number literals to increase readability

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Java 7: New IO API

- A whole new package: `java.nio`
- Non-blocking IO
- Buffer oriented instead of Stream oriented
- New classes to improve working with files
  - `Files`
  - `Path`
  - `FileSystem`
  - `WatchService`
  - `FileVisitor`
  - `...`

Java 7: New IO API

```java
// Reading a file in Java 6
try {
    FileInputStream fstream = new FileInputStream(
        "/some/dir/test.txt");
    DataInputStream in = new DataInputStream(fstream);
    BufferedReader br = new BufferedReader(new
InputStreamReader(in));
    String strLine;
    while ((strLine = br.readLine()) != null) {
        System.out.println(strLine);
    }
    in.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

Java 7: New IO API

```
// Reading a file in Java 7
try {
    List<String> lines = Files.readAllLines(
    FileSystems.getDefault().getPath("/some/dir/file.txt"),
StandardCharsets.UTF_8);
    for (String line : lines) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Java 7: New IO API

```java
final FileSystem fileSystem = FileSystems.getDefault();
try (final WatchService watchService = fileSystem.newWatchService()) {
    final Map<WatchKey, Path> keyMap = new HashMap<>();
    final Path path = FileSystems.getDefault().getPath("/some/dir");
    try {
        keyMap.put(path.register(watchService, StandardWatchEventKinds.ENTRY_CREATE,
StandardWatchEventKinds.ENTRY_MODIFY, StandardWatchEventKinds.ENTRY_DELETE), path);
    } catch (IOException e) {
        e.printStackTrace();
    }
    WatchKey watchKey;
    do {
        watchKey = watchService.take();
        final Path eventDir = keyMap.get(watchKey);
        for (final WatchEvent<?> event : watchKey.pollEvents()) {
            final Kind kind = event.kind();
            final Path eventPath = (Path) event.context();
            System.out.println(eventDir + ": " + event.kind() + ": " +
eventDir.resolve(eventPath));
        }
    } while (watchKey.reset());
} catch (InterruptedException | IOException ex) {
    // Oops, something went wrong
}
```

Java 7: New IO API

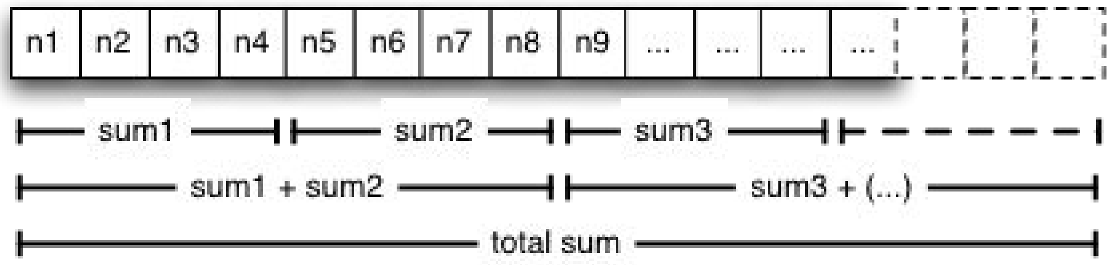
```java
public class Find extends SimpleFileVisitor<Path> {
    public static void main(final String[] args) throws IOException {
        final FileVisitor<Path> fileVisitor = new Find();
        final Path root = Paths.get("/some/dir");
        Files.walkFileTree(root, fileVisitor);
    }

    @Override
    public FileVisitResult preVisitDirectory(final Path dir, final
BasicFileAttributes attrs) {
        if (".svn".equals(dir.getFileName().toString())) {
            return FileVisitResult.SKIP_SUBTREE;
        }
        System.out.println("Directory found: " + dir);
        return FileVisitResult.CONTINUE;
    }

    @Override
    public FileVisitResult visitFile(final Path file, final BasicFileAttributes
attrs) {
        System.out.println("File found: "+ file);
        return FileVisitResult.CONTINUE;
    }
}
```

Java 7: Fork Join Framework

- Parallel programming
  - Divide a process into smaller tasks via recursion which are handled by a processor
  - Combine the processed pieces into one result
  - Divide & conquer



Java 7: Fork Join Framework

- Extend `RecursiveAction` or `RecursiveTasks`

```
if (my portion of the work is small enough) {
    do the work directly
} else {
    split my work into two pieces
    invoke the two pieces and wait for the results
}
```

- Practical example: `ForkBlur.java`

Java 7: JVM enhancements

- Support for dynamically typed languages
  - Introduction of invokedynamic
    - A new bytecode instruction on the JVM for method invocation
    - http://niklasschlimm.blogspot.be/2012/02/java-7-complete-invokedynamic-example.html
  - Performance improvements for other languages living in the JVM such as Ruby, Groovy, …
- Garbage-First Collector (or G1 collector)
  - Will eventually replace the Concurrent Mark-Sweep Collector (CMS)
  - Advantages: works with regions, more predictable

Java 8 (March 18th 2014)

- Lambda Expressions
- Extension Methods
- Functional Interfaces
- Method and Constructor References
- Streams and Bulk Data Operations for Collections
- Removal of PermGen
- New Date & Time API
- New Default API for Base64 Encoding
- Improvements for Annotations
- General Performance Improvements

Java 8: Lambda Expressions

- Allows writing code in a functional style
- Passing behaviour to a method
- Prior to Java 8: Anonymous Inner Class
- Java 8: Lambda Expressions
- More readable and clear code
- Type of param may be specified but isn't obligated

```
(params) -> expression

() -> System.out.println("Hello world!");

myButton.addActionListener(
   e -> System.out.println("Clicked")
}
```

Java 8: Lambda Expressions

```
// Java 7
for (String s : aList) {
    System.out.println(s);
}

// Java 8
aList.forEach((String s) -> System.out.println(s));
// or shorter
aList.forEach(s -> System.out.println(s));
```

Java 8: Lambda Expressions

```
// Java 7
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("You clicked me!");
    }
});

// Java 8
button.addActionListener(
    e -> System.out.println("You clicked me!")
);
```

Java 8: Extension Methods

- Add non-abstract method implementations to interfaces using the ‘default’ keyword

```
interface Printer {
    default void print(String s) {
        System.out.println(s);
    }
}
```

- But what happens if default methods collide when using multiple interfaces?

```
interface Copier {
    default void print(String s) { System.out.println("Out of 
paper..."); }
}

static class MultiFunctionalPrinter implements Printer, Copier {
    // uh oh?
}
```

Java 8: Extension Methods

- Override method and pick the right implementation

```
static class MultiFunctionalPrinter implements Printer,
Copier {
    @Override
    public void print(String s) {
        Copier.super.print(s);
    }
}
```

Java 8: Functional Interfaces

- `@FunctionalInterface`
- An interface with exactly one abstract method
- A Lambda expression is applicable as implementation

```
@FunctionalInterface
interface Formula {
    double calculate(int a, int b);
}
```

- Build-in Functional Interfaces (`java.util.function`)
  - `Predicate<T>: boolean test(T t);`
  - `Function<T>: R apply(T t);`
  - `Supplier<T>: T get();`
  - `Consumer<T>: void accept(T t);`
  - `Comparator<T>: int compare(T o1, T o2);`

Java 8: Functional Interfaces

```
// Java 7
Formula myFormula = new Formula() {
    @Override
    public double calculate(int a, int b) {
        return a + b;
    }
};

// or using a Lambda
Formula myFormulaV2 = (a, b) -> a - b;
```

Java 8: Functional Interfaces

```
// Predicate example (takes an argument and returns a
boolean result)
Predicate<String> isNull = s ->  s == null;
System.out.println(isNull.test("something"));

// Function example (takes an argument and returns a result)
Function<String, Integer> calculateLength = s -> s.length();
System.out.println(calculateLength.apply("foo"));

// Consumer example (takes an argument and returns nothing,
eg it operates via side-effects)
Consumer<Integer> consumer = x -> System.out.println(x);
Arrays.asList(1,2,3).forEach(consumer);
```

ORDINA

CONNECTIVATE

Java 8: Method and Constructor References

- Pass references of methods or constructors using the `::` keyword
- Useful in combination with the Predicate class
- Bit shorter compared to lambdas

```
ContainingClass::staticMethodName            String::valueOf
ContainingObject::instanceMethodName         s::toString
ContainingType::methodName                   String::toString
ClassName::new                               String::new
```

Java 8: Method and Constructor References

```java
@FunctionalInterface
interface Converter<F, T> {
    T convert(F from);
}

Converter<String, Integer> integerConverter = s ->
Integer.parseInt(s);
Integer integer = integerConverter.convert("125");
System.out.println(integer);

// can also be written using a static method reference
Converter<String, Integer> integerConverter = Integer::parseInt;
integer = integerConverter.convert("76");
System.out.println(integer);
```

Java 8: Method and Constructor References

```java
static class Book {
    String author;
    String title;

    Book() {}
    Book(String aAuthor, String aTitle) {
        this.author = aAuthor;
        this.title = aTitle;
    }
    public String getAuthor() { return author; }
    public String getTitle() { return title; }
}
```

Java 8: Method and Constructor References

```java
@FunctionalInterface
interface BookFactory {
    Book create(String aAuthor, String aTitle);
}


BookFactory bookFactory = Book::new;
Book b = bookFactory.create("Yannick", "Yannick's book");
```

Java 8: Streams and Bulk Data Operations for Collections

- `java.util.Stream`
- A sequence of elements on which one or more operations can be performed
- Intermediate vs terminal operation
  - Intermediate: returns the stream itself in order to be able to chain operations
  - Terminal: returns a result of a certain type
- Streams are created on a source such as a `java.util.Collection`
- Can be executed sequential or parallel
- Parallel utilises Fork-Join
  - Watch out with long-running tasks! Blocks threads in the pool

Java 8: Streams and Bulk Data Operations for Collections

```
List<Integer> list = Arrays.asList(1, 3, 5, 7, 13, 17, 23);
// Filter
list.stream()
        .filter(i -> i > 10)
        .forEach(System.out::println);
// Sorted
list.stream()
        .sorted((i1, i2) -> i1.compareTo(i2) * -1)
        .forEach(System.out::println);
// Map
list.stream()
        .map(i -> i + 1)
        .forEach(System.out::println);
// Collect
String joinedList = list.stream()
        .map(i -> i.toString())
        .collect(Collectors.joining(", "));
System.out.println(joinedList);
```

Java 8: Streams and Bulk Data Operations for Collections

- Maps
  - Don’t support streams :-(
  - … But they now support various new and useful methods for executing common tasks!
    - `V putIfAbsent(K key, V value)`
    - `void forEach(BiConsumer<? super K,? super V> action)`
    - `V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)`
    - `V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)`
    - `V getOrDefault(Object key, V defaultValue)`
    - `...`

Java 8: Streams and Bulk Data Operations for Collections

```
// putIfAbsent
Map<Integer, String> map = new HashMap<>();
for (int i=0; i<10; i++) {
    map.putIfAbsent(i, "value #" + i);
}
for (int i=0; i<10; i++) {
    map.putIfAbsent(i, "otherValue #" + i);
}

// forEach
map.forEach((id, val) -> System.out.println(id + " -> " + val));

// computeIfPresent
map.computeIfPresent(3, (key, val) -> val + "(computed)");
System.out.println("Key 3 = " + map.get(3));
```

Java 8: Streams and Bulk Data Operations for Collections

- `Optional<T>`
  - May or may not contain a non-null value
  - Avoid working with null (no NPEs!)

```
Optional<String> optional = Optional.of("value");
Optional<String> emptyOptional = Optional.empty();
Optional<Integer> length = optional.map(String::length);
System.out.println("length = " + length.orElse(0));

optional.map(s -> s.substring(0, 3))
    .filter(t -> t.contains("al"))
    .ifPresent(System.out::println);
```

Java 8: Removal of PermGen

- PermGen memory space completely removed
  - `PermSize` and `MaxPermSize` JVM arguments are ignored and a warning gets displayed
- Gets replaced by Metaspace
  - `XX:MaxMetaspaceSize` flag, default is unlimited
  - System memory is the limit instead of the fixed size at startup of PermGen
  - Metaspace will dynamically resize depending on demand at runtime
- Note that this does not magically fixes your memory leaks!

Java 8: New Date & Time API

- Inspired by Joda Time
  - Human time vs computer time (aka millis since epoch)
- Offers a solution to the sometimes cumbersome way of calculating dates and time
- Interesting new classes:
  - `Clock`
  - `ZoneId`
  - `LocalDate` (date without timezone)
  - `LocalTime` (time without timezone)
  - `LocalDateTime` (datetime without timezone)
  - `DateTimeFormatter`
  - …

Java 8: New Date & Time API

```
// Doing calculations with dates
Period p = Period.ofWeeks(2);
System.out.println("Now plus 2 weeks: " + date.plus(p)); // Alternative:
date.plusWeeks(2);
LocalTime now = LocalTime.now();
LocalTime calculatedTime = now.plusMinutes(40);
System.out.println("Now plus 40 mins: " + calculatedTime.toString());

// Easily create dates
date = LocalDate.of(2014, Month.MARCH, 24);
System.out.println("Date: " + date);

LocalTime time = LocalTime.of(16, 20, 31);
System.out.println("Time: " + time);

ZonedDateTime zonedTimeBrussels = ZonedDateTime.of(2014, 4, 1, 8, 30, 0, 0,
    ZoneId.systemDefault());
ZonedDateTime zonedTime = zonedTimeBrussels.withZoneSameInstant(chihuahua);
System.out.println("Datetime in Brussels: " + zonedTimeBrussels);
System.out.println("Datetime in Chihuahua: " + zonedTime);
```

Java 8: New Default API for Base64 Encoding

- More extensive API than the 1.6+ Base64 API (`sun.misc.BASE64Encoder`)
- 3 encoders and decoders
  - Basic (For regular encoding)
  - URL (Encoded String needs to be used in file or url)
  - MIME (MIME friendly encoding)

Java 8: New Default API for Base64 Encoding

```java
// Java 1.6 API from JAXB
String encodedString =
    DatatypeConverter.printBase64Binary("secret".getBytes("UTF-8"));

// Java 8
String encodedString = new
    String(Base64.getEncoder().encode("secret".getBytes("UTF-8")),
        "UTF-8");
String decodedString = new
    String(Base64.getDecoder().decode(encodedString.getBytes()),
        "UTF-8");

// Basic vs URL
String basicEncoded = Base64.getEncoder().encodeToString(
    "watch?v=oavMtUWDBTM".getBytes("utf-8"));

String urlEncoded = Base64.getUrlEncoder().encodeToString(
    "watch?v=oavMtUWDBTM".getBytes("utf-8"));
```

Java 8: Improvements for Annotations

- Annotations in Java 8 are repeatable
- `@Repeatable`

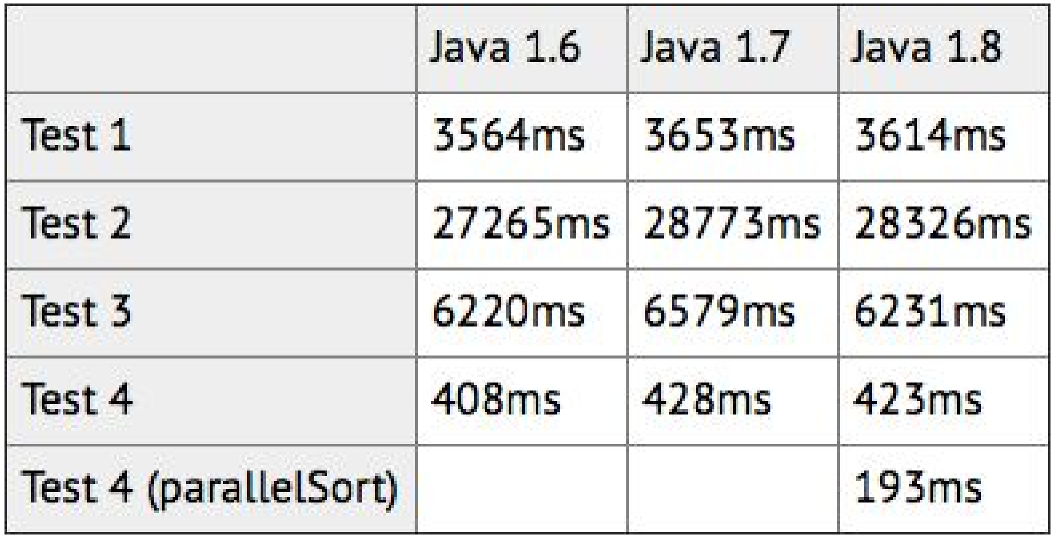
```java
@interface Cars {
    Car[] value();
}
@Repeatable(Cars.class)
@interface Car {
    String value();
}
@Car("Opel Corsa")
class Person {}


@Cars({@Car("Porsche Boxter"), @Car("BMW 3")})
class Family2 {}

@Car("Volkswagen Sharan")
@Car("Mini Cooper")
class Family1 {}
```

Java 8: General Performance Improvements

- Performs a bit faster compared to Java 7
- Great performance improvement when making use of parallelism
- Example with `Arrays.sort`



	Java 1.6	Java 1.7	Java 1.8
Test 1	3564ms	3653ms	3614ms
Test 2	27265ms	28773ms	28326ms
Test 3	6220ms	6579ms	6231ms
Test 4	408ms	428ms	423ms
Test 4 (parallelSort)			193ms

Sources

- Examples and exercises
  - https://github.com/yannickdeturck/workshop-java-7-8
- Java 8 Cheatsheet: http://www.java8.org

ANY
QUESTIONS
?