

Effective Enterprise Application Development

Martin Ledvinka

martin.ledvinka@fel.cvut.cz

Winter Term 2017



Contents

- 1 Java 8 Features
 - Lambdas and Method References
 - Stream API
 - Optional
- 2 Continuous Integration
- 3 Static Source Code Analysis Tools
- 4 Application Monitoring and Administration
 - JMX
 - Monitoring Tools
- 5 Conclusions



Java 8 Features



Behaviour Parametrization

What do these have in common?

```
for (Report r : reports) {  
    if (!r.getAuthor().equals(me)) {  
        throw new AuthorizationException("You cannot edit other people's  
            reports.");  
    }  
}
```

```
for (Report r : reports) {  
    if (cache.contains(r)) {  
        cache.evict(r);  
    }  
}
```



Behaviour Parametrization before Java 8

Functors

```
interface Filter {...}
class AuthorizationFilter implements Filter {...}
class CacheEvictingFilter implements Filter {...}

processReports(reports, new AuthorizationFilter());
```

Anonymous Class

```
interface Filter {...}

processReports(reports, new Filter {

    process(Report r) {
        // ...
    }

});
```

Behaviour Parametrization in Java 8

```
processReports(reports, r -> {  
    if (!r.getAuthor().equals(me)) {  
        throw new AuthorizationException("You cannot edit other  
            people's reports.");  
    }  
});
```



Lambdas

A *lambda* is an anonymous function.

- Based on *Lambda calculus*,
- Quick throwaway function without a name,
- A way to express a *closure*,
 - Closure can access the lexical scope it is defined in,
 - In Java, this access is read-only.

```
Function<Integer> createAdder(int toAdd) {  
    return n -> n + toAdd;  
}
```

```
Function<Integer> addTwo = createAdder(2);  
Function<Integer> addFive = createAdder(5);  
int result = addTwo(1); // result = 3  
result = addFive(result); // result = 8
```



References to Functions

- Functions become first-class citizens in Java 8,
 - Can have references to function/methods,
- Further step in behaviour parametrization.

They allows us to:

- Store references to functions/methods,
- Pass references as arguments to other functions/methods.

```
Predicate<Integer> isEven = n -> n % 2 == 0;
```

```
void processReports(Collection<Report> reports, Consumer<Report> fn) {  
    for (Report r : reports) {  
        fn.accept(r);  
    }  
}
```



How Does it Work?

Functional Interfaces

- Interface with a single abstract method,
- Provide target types for lambdas and method references,
 - i.e. They are used as parameter/variable types.
- A number of them defined in `java.util.function`:
 - e.g. `Consumer`, `Function`, `Producer`, `Predicate`.

Technically

- Lambda captures all *effectively final* variables in its lexical scope,
- Improved *type inference*.



Method References

- Syntactic shortcut,
- References to:
 - Static method,
 - Instance method of a particular object,
 - Instance method of an arbitrary object of a particular type,
 - Constructor.

```
Arrays.sort(stringArray, String::compareToIgnoreCase);  
  
// instead of  
  
Arrays.sort(stringArray, (a, b) -> a.compareToIgnoreCase(b));
```



Syntax

Lambda

```
(arg1, arg2) -> {  
    // Do something  
    return result;  
}
```

```
() -> result
```

Method Reference

```
processReports(reports, Filter::accept);
```

```
processReports(reports, myFilter::accept);
```



Stream API

A stream is a sequence of elements supporting aggregate operations.

- Mostly used in collection processing,
- Generation of numeric data,
- Pipelines – operations on streams returning streams,
- Provide internal iteration,
- Code is:
 - Declarative,
 - Composable,
 - Parallelizable.



Stream v Collection

Collection

- Eager collection of data,
- Data structure holds all values the collection currently has,
- External iteration (for cycle, iterator).

Stream

- Elements computed on demand,
- Allows processing of possibly infinite data structures (e.g. prime numbers),
- Traversable only once,
- Internal iteration (not controlled by programmer).



Stream

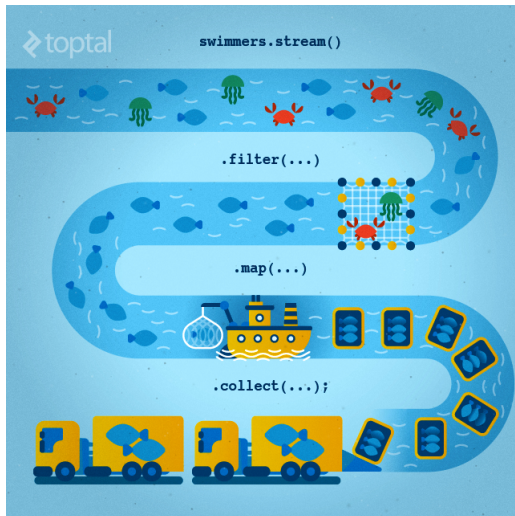


Figure : Stream processing visualization. Source: <https://www.toptal.com/java/why-you-need-to-upgrade-to-java-8-already>



Stream Operations

Intermediate operations support pipeline processing – multiple operations executed on data. E.g. `filter`, `map`, `limit`.

Terminal Operations close the stream. E.g. `collect`, `forEach`.

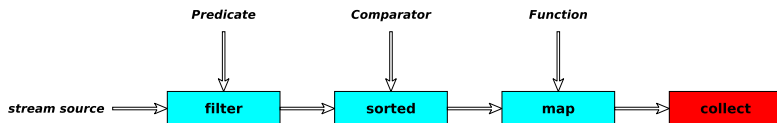


Figure : Stream operations.



Stream Examples

```
reports.parallelStream().filter(r ->
    !r.getAuthor().equals(me)).findAny();

reports.stream().filter(cache::contains).forEach(cache::evict);

List<String> todaysReports = reports.stream()
    .filter(r ->
        r.getDate().after(midnight))
    .map(Report::getName)
    .collect(Collectors.toList());

int sum = numbers.stream().reduce(0, Integer::sum);

Stream.generate(Math::random).limit(5).collect(Collectors.toList());
```



Optional

Allows to avoid null reference checks and `NullPointerException`s.

- `Optional.empty()`
- `Optional.of(T)`
- `Optional.ofNullable(T)`
- `get()`
- `ifPresent(Consumer)`
- `isPresent()`
- `map(Function)`
- `orElse(T)`
- `orElseGet(Supplier)`
- `orElseThrow(Supplier)`



Optional Examples

Null Check

```
final Report latest = findLatestRevision(fileNumber);
if (latest == null) {
    throw new NotFoundException("Report with fileNumber " + fileNumber
        + " not found.");
}
return latest;
```



```
final Optional<Report> latest = findLatestRevision(fileNumber);
return latest.orElseThrow(() -> new NotFoundException("Report with
    fileNumber " + fileNumber + " not found."));
```

```
Optional<Report> notMine = reports.parallelStream().filter(r ->
    !r.getAuthor().equals(me)).findAny();
notMine.ifPresent(r -> {
    throw new AuthorizationException();
});
```



Continuous Integration



Continuous Integration

- Term coined by Grady Booch,
- Adopted by the Extreme Programming community,
- Developers in a team integrate work at least daily,
- Integration verified by an automated build,
- Quick detection of errors, cheaper fixes, fewer integration issues.



CI Practices

- Single source code repository, use CI server,
 - Should contain all the code and configuration, so that clean clone from the repository is buildable,
- Automated build,
 - e.g. using Maven, Gradle,
 - Quick on developer machine,
 - Including automated tests,
- Build before commit/push,
- Push every day (commit → pull changes → resolve conflicts → push),
- Every push triggers build on the CI server,
- Fix broken builds immediately,
- Test in a clone of the production environment,
- Automate deployment.



CI Tools

SCM

- **Git,**
- **Subversion,**
- **RTC.**

CI Servers

- **Jenkins**
 - Open-source, free,
 - Highly configurable, lots of plugins.
- **TeamCity**
 - Free for 3 agents and 20 build configurations,
 - Developed by JetBrains,
 - More suitable for enterprises.
- **Travis CI**
 - Hosted solution,
 - Free for open-source projects (often used by Github projects).

Static Source Code Analysis Tools



Static Code Analysis

Analysis of software without actually executing the program.

- Can be formal, but usually not feasible for larger programs,
 - Used in high risk industries,
 - E.g. aviation, power plants, medicine,
- Mostly based on heuristics,
 - False positives possible,
- Detects suspicious patterns in code.

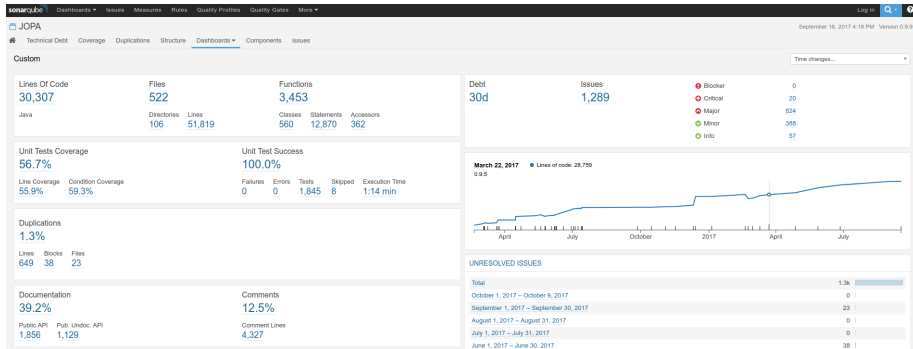


SCA Tools

- **IDE**
 - Most IDEs contain some sort of SCA feature.
- **Checkstyle**
 - Can be integrated into Maven build.
- **FindBugs**
 - Older tool. Plugins exist for all major IDEs and CI servers.
- **Sonarqube**
 - Multiplatform code analysis tool.
- **Upsource**
 - Code review, SCA, team collaboration.



Sonarqube



JOPA JOPA - implementation src/main/java/cz/cvut/kbss/jopa/model/EntityManagerImpl.java

Do not override the Object.finalize() method. 2 years ago L516

● Critical ○ Open ○ Not assigned ○ Not planned 20min debt cert, unpredictable

JOPA JOPA - implementation src/main/java/cz/cvut/kbss/jopa/model/metamodel/ManagedClassProcessor.java

Either log or rethrow this exception. 10 months ago L87

● Critical ○ Open x Martin Ledvinka ○ Not planned 10min debt error-handling, security



Application Monitoring and Administration



Java Management Extensions (JMX)

- Allow management of resources in an application,
- Standard part of the Java platform,
- Resources represented by *Managed Beans (MBeans)*, registered in an *MBean* server,
- Accessible via JMX connectors.

Managed Beans

- Operations (MBean methods), through which the application can be managed,
- Attributes (getters/setters) for information/configuration.



Application Management via JMX

- Connect to application with *JConsole*,
- Locate the desired MBean,
 - Invoke managed operations,
 - View/configure attributes,
- MBean server set up in Spring – `@EnableMBeanExport`.

The screenshot shows the JConsole interface with the 'MBeans' tab selected. The left-hand tree view shows the hierarchy of MBeans, with 'ch.qos.logback.classic.jmx.JMXConfigurator' selected. The right-hand pane displays the configuration for this MBean.

MBeanInfo	
Name	Value
Info:	
ObjectName	ch.qos.logback.classic.Name=default,Type=ch.qos.logback.classic.jmx.JMXConfigurator
ClassName	ch.qos.logback.classic.jmx.JMXConfigurator
Description	Information on the management interface of the MBean
Constructor-0:	
Name	ch.qos.logback.classic.jmx.JMXConfigurator
Description	Public constructor of the MBean
Parameter-0-0:	
Name	p1
Description	
Type	ch.qos.logback.classic.LoggerContext
Parameter-0-1:	
Name	p2
Description	
Type	javax.management.MBeanServer
Parameter-0-2:	
Name	p3
Description	
Type	javax.management.ObjectName

Descriptor	
Name	Value
Info:	
immutableInfo	true
interfaceClassName	ch.qos.logback.classic.jmx.JMXConfiguratorMBean
mbean	false



JConsole

- GUI-based Java monitoring tool,
- JMX compliant,
- Allows connection to local or remote (if configured) processes,
- Part of the JDK.



VisualVM

- GUI-based Java monitoring tool,
- Allows collection and saving of monitoring data,
 - Thread dump, heap dump,
- Profiling, sampling,
 - CPU, memory,
 - Local applications only,
 - Profiling has major impact on application performance,
- Support for plugins,
- Analysis of stored thread or heap dumps.



More Tools

JDK

- **jmap** – memory-related statistics about a VM, obsolete,
 - **jcmd** – send diagnostic commands to JVM, internally used by the GUI tools,
 - **jstat** – monitors JVM statistics, lots of options.
-
- **Eclipse MAT** – advanced memory analyzer,
 - **Java Mission Control** and **Java Flight Recorder** – commercial JVM monitoring tools by Oracle,
 - **StageMonitor**, **MoSKito** etc. – open source alternatives.



Conclusions



The End

Thank You



Resources

- R. Urma, M. Fusco and A. Mycroft: *Java 8 in Action*,
- <http://www.oracle.com/technetwork/articles/java/architect-lambdas-part1-2080972.html>,
- <http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html>,
- <https://martinfowler.com/articles/continuousIntegration.html>,
- <http://docs.oracle.com/javase/tutorial/jmx/mbeans/index.html>,
- <http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html>,
- <https://visualvm.github.io/documentation.html>.

