

Further Steps in Enterprise Application Development

Petr Aubrecht (+Martin Ledvinka)

`petr@aubrecht.net`

Winter Term 2020



Contents

- 1 Java 8 – 16 Features
 - Lambdas, Streams
 - Optional
- 2 Agile World
 - Continuous Integration & Deployment
- 3 Application Monitoring and Administration
 - JMX
 - Monitoring Tools
- 4 Database Versioning
- 5 Production



Java 8 – 16 Features



Cycles Simplification

How to do make simpler?

```
List<LocalDate> myReports = new ArrayList<>();  
for (Report r : reports) {  
    if (r.isActive()) {  
        if (r.getAuthor().equals(me)) {  
            myReports.add(r.getDueTo());  
        }  
    }  
}  
Collections.sort(myReports);
```

Nothing wrong, business as usual. Can we do it better?



Cycles Simplification

Is this better/more readable?

```
List<LocalDate> myReports = reports.stream()  
    .filter((Report r) -> r.isActive())  
    .filter((Report r) -> r.getAuthor().equals(me))  
    .map((Report r) -> r.getDueTo())  
    .sorted()  
    .collect(Collectors.toList());
```

...and we can continue...



Cycles Simplification

We can remove types...

```
List<LocalDate> myReports = reports.stream()  
    .filter(r -> r.isActive())  
    .filter(r -> r.getAuthor().equals(me))  
    .map(r -> r.getDueTo())  
    .sorted()  
    .collect(Collectors.toList());
```

...and we can continue...



Cycles Simplification

We can use method reference...

```
public static boolean isMyReport (Report r) {  
    return r.equals (me);  
}
```

```
List<LocalDate> myReports = reports.stream()  
    .filter (Report::isActive)  
    .filter (TestFunctional::isMyReport)  
    .map (Report::getDueTo)  
    .sorted()  
    .collect (Collectors.toList());
```

...Let's compare it on the next slide!



Cycles Simplification

Before

```
List<LocalDate> myReports = new ArrayList<>();  
for (Report r : reports) {  
    if (r.isActive()) {  
        if (r.getAuthor().equals(me)) {  
            myReports.add(r.getDueTo());  
        }  
    }  
}  
Collections.sort(myReports);
```

After

```
List<LocalDate> myReports = reports.stream()  
    .filter(Report::isActive)  
    .filter(TestFunctional::isMyReport)  
    .map(Report::getDueTo)  
    .sorted()  
    .collect(Collectors.toList());
```


Stream

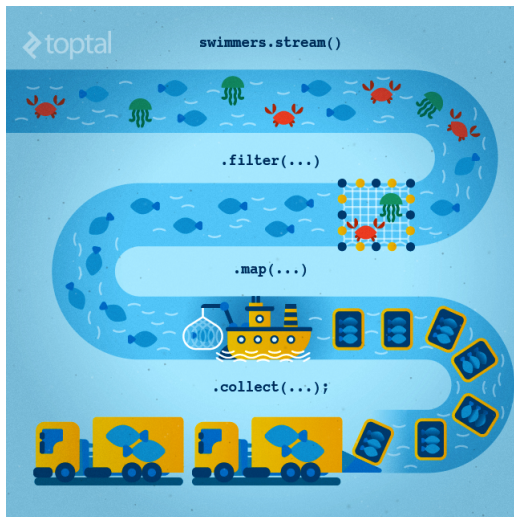


Figure: Stream processing visualization. Source: <https://www.toptal.com/java/why-you-need-to-upgrade-to-java-8-already>

Lambda for Multithreaded Application

So far it was just a syntax sugar. BUT! How easily can you write multithreaded apps?

```
List<LocalDate> myReports = reports.stream()  
    .parallel() // run on multiple threads!  
    .filter(RemoteVerification::isValid) // calls outside service  
    .collect(Collectors.toList());
```



Optional

Before

```
Report r = reports.get(0);  
Band header = r.getHeaderBand();  
if(header!=null) {  
    title = header.getTitle();  
    if(title==null) {  
        title = "Default Title";  
    }  
}
```

After

```
String title = Optional.of(reports.get(0))  
    .map(Report::getHeaderBand)  
    .map(Band::getTitle)  
    .orElse("Default Title");
```

Log4j

Before – annoying

```
if(log.isDebugEnabled()) {  
    log.debug(prepareDataForLog());  
}
```

Simple – useless overhead if not used

```
log.debug(prepareDataForLog());
```

Functional – simple and effective

```
log.debug(() -> prepareDataForLog());
```



Details

Switch Expression

```
switch (day) {  
    case MONDAY, FRIDAY, SUNDAY -> 6;  
    case TUESDAY -> 7;  
    case THURSDAY, SATURDAY -> 8;  
    case WEDNESDAY -> 9;  
    default -> throw new IllegalStateException("Invalid day: " + day);  
}
```

Records aka Lombok

```
record Point(int x, int y) {}
```

Small Details

Collectors.teeing, String.repeat or lines, text block, shebang,
if(abj instanceof String str) {}, JMH, NullPointerException: a.b.c()

Agile World



Waterfall, Model V

- What's wrong with waterfall, model V (e.g. detailed planning before programming)? Everything!
 - Detailed analysis becomes useless immediately after programming starts – many assumptions are wrong.
 - Detailed long-time planning is crazy – can you say, what you will do on September 21st 2021 in the morning? And afternoon?
 - Users tend to change their minds when they see the first version.
 - Programming takes long time and situation changes.
 - Studies have shown that in over 80 % of the investigated and failed software projects, the usage of the Waterfall methodology was one of the key factors of failure. https://www.scrum-institute.org/What_Makes_Waterfall_Fail_in_Many_Ways.php



Agile Style of Work

- Principles (see agilemanifesto.org)
 - Individuals and interactions over processes and tools
 - Working software over comprehensive documentation
 - Customer collaboration over contract negotiation
 - Responding to change over following a plan
- Pair programming
- SCRUM or Canban
- Cooperation is much more important than individual success.
- **Frequent and regular increments!** Often are shared with customers.



Continuous Integration

- After every commit, build is verified – including unit tests
- At least once a day, the whole product is deployed – including functional tests
- UI tests are done frequently (can take hours)
- ... all automated.
- Quick detection of errors, cheaper fixes, fewer integration issues.



CI Tools

Version Control

- **Git**, others: CSV, Subversion, Bazaar, Mercurial, Bitkeeper, RTC...

CI Servers

- **Jenkins**
 - Open-source, easy to setup,
 - Highly configurable, lots of plugins.
- **TeamCity**
 - Free for 3 agents and 20 build configurations,
 - Developed by JetBrains,
 - More suitable for enterprises – beast.
- **Gitlab CI**
- Today's servers concentrate on whole process including deployment to cloud.

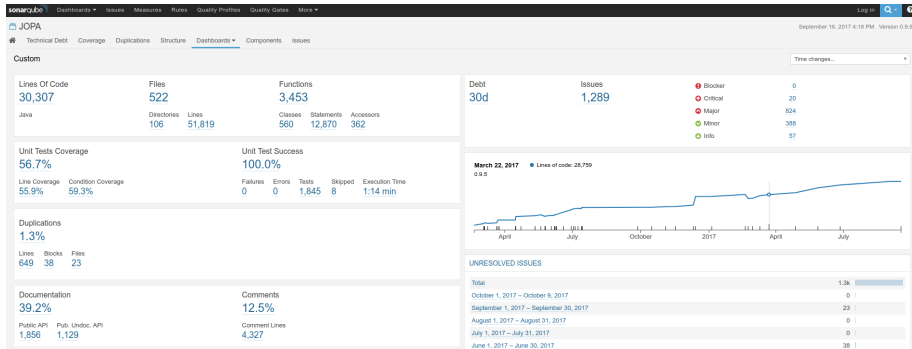


Static Code Analysis

- Analyze code structure or flows, don't run it.
- Full-featured IDEs contain some sort of SCA.
- **Checkstyle** – checks just formatting.
- **FindBugs** – simple and pretty fast check, can find adding to String inside cycle, impossible equals, bad null handling. . .
- **Sonarqube** – server-side analysis, long, discovers data flow from database to servlet (e.g. finds XSS)



Sonarqube



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

Do not override the Object.finalize() method.

Critical Open Not assigned Not planned 20min debt

2 years ago L516

cert, unpredictable

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

Either log or rethrow this exception.

Critical Open Martin Ledvinka Not planned 10min debt

10 months ago L87

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 pane displays a tree of MBeans, with 'ch.qos.logback.classic.jmx.JMXConfigurator' selected under the 'default' package. The right pane shows the details 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 – LIVE DEMO

- 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, works also on IBM Java on mainframes
- 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.
 - **CA Wily** – very famous and very detailed monitoring of JavaEE



JavaMelody

- Very simple to implement (few lines in web.xml, few lines in pom.xml)
- Navigate to `/monitoring` and enjoy!
- LIVE DEMO (at least PDF)
 - Records data for last year, older data is summarized, old data removed
 - memory, sessions, threads, CPU, disk space, network bandwidth, SQL traffic, JMX, timing statistics of requests...
 - actions: invalidate sessions, perform garbage collection, memory dump



Prometheus + Grafana

- Ready software, frequently used in production
- LIVE DEMO
- Prometheus collects and saves metrics
 - just provide data in the right format
- Grafana makes nice dashboards, graphs
 - make your own dashboard drag&drop
 - download some ready dashboard via "Import via grafana.com"



Database Versioning



Database Versioning

- JPA provides a possibility to create missing tables
- ... useless when table is changed
- Libraries: **Liquibase** and **Flyway**
- A list of changes is recorded, keeps current database version
- Application keeps steps to upgrade from one version to the next
- The most reliable way
- Alternatives: direct upgrades from older version (leads to multiple ways – hard testing), creating SQL scripts (customers tend to make mistakes during deployment, problematic error handling)



Production



Production Environments

- As usual – supported servers inside client's network (Payara, Glassfish, TomEE, WildFly, WebSphere)
- Hosted – our servers in server houses
- Currently investigating – Clouds, Docker
 - Problem with acceptance in banks
 - Cloud requires multitenancy application, e.g. there is a big risk of information leak, very rare
 - Docker seem a good choice, pack of all required software, needs just CPU, memory, disk space, TCP/IP ports.



What We Actually Use

- Versioning: git, gitlab
- CI: Jenkins, investigating Gitlab CI
- Code analysis: Findbugs
- IDE: NetBeans :-), Idea (In fact, this doesn't matter.)
- Servers: Payara, TomEE, less Glassfish, WebSphere, WildFly
- Databases: PostgreSQL, MSSQL, Oracle
- Monitoring: JavaMelody
- OS: our systems – Linux, clients often Windows, recently Docker



The End

Thank You

Petr Aubrecht
petr@aubrecht.net



Resources

- R. Urma, M. Fusco and A. Mycroft: Java 8 in Action,
- <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>,
- <https://github.com/javamelody/javamelody/wiki>.

