Java Best Practices

Version 1.18-SNAPSHOT

2018-08-23

Table of Contents

Table 1. History

Date	Author	Detail
2018-08-23	bcouetil	Initial commit

1. Useful Java libraries

1.1. Mockito / PowerMockito

Usage for static classes

```
@RunWith(PowerMockRunner.class)
@PrepareForTest({ TypeUtils.class })
@PowerMockIgnore("javax.management.*")
public class OpenPojoWebTest {

    @Before
    public void before() throws Exception {
        PowerMockito.mockStatic(TypeUtils.class);
        PowerMockito.when(TypeUtils.setterDate((Date) Mockito.any(), (Date) Mockito.any()))
                .thenAnswer(invocation -> invocation.getArgumentAt(1, Date.class));
    }

}
```

1.2. OpenPojo : Auto test Pojo classes for coverage

https://github.com/OpenPojo/openpojo

OpenPojo au tests Pojo classes, especially getters and setters. Very handy for large beans / auto generated classes for whom testing is boring.

Usage

```
import com.openpojo.reflection.filters.FilterNonConcrete;
import com.openpojo.validation.Validator;
import com.openpojo.validation.ValidatorBuilder;
import com.openpojo.validation.test.impl.GetterTester;
import com.openpojo.validation.test.impl.SetterTester;

public class OpenPojoTest {

    public static void validateBeans(String javaPackage) {
        Validator validator = ValidatorBuilder.create().with(new SetterTester()).with(new GetterTester()).build();
        //exclude enums, abstracts, interfaces
        validator.validateRecursively(javaPackage, new FilterNonConcrete());
    }

    @Test ①
    public void testPojoRecursiv() {
        // recursive
        validateBeans("my.full.java.package.with.sub.packages");
    }

    @Test ②
    public void testExludingSomeClasses() {
        List<PojoClass> listOfPojoClassInDto = PojoClassFactory.getPojoClasses("my.full.java.package.with.sub.packages",
null);
        listOfPojoClassInDto.remove(PojoClassFactory.getPojoClass(SomeSpecialClassNotToTest.class));
        validator.validate(listOfPojoClassInDto);
    }

}
```

① Fully recursive example

② Excluding some classes

Maven dependency

```
<dependency>
    <groupId>com.openpojo</groupId>
    <artifactId>openpojo</artifactId>
    <version>0.8.6</version>
    <scope>test</scope>
</dependency>
```

1.3. SLF4J : Abstract logging

Maven dependencies

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.21</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>1.7.21</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.7</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.7</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.7</version>
</dependency>
```

1.4. Aspect4log : Logging functions starts/stops with inputs/outputs

See http://aspect4log.sourceforge.net

Use Aspect4Log, which logs functions start/stop with inputs/outputs using AOP.

Result log example

```
07-31_14:13:48.491 DEBUG org.a.utils.ConfigUtils        - > getParameter(test)
07-31_14:13:48.491 DEBUG org.a.utils.wmcall.WmHelper    - >  getPackageName(true)
07-31_14:13:48.492 DEBUG g.a.utils.wmcall.WmCallEclipse - >      getPackageName(true)
07-31_14:13:48.492 DEBUG g.a.utils.wmcall.WmCallEclipse - .         getPackageName(true) -> DEFAULT
07-31_14:13:48.492 DEBUG org.a.utils.wmcall.WmHelper    - .     getPackageName(true) -> DEFAULT
07-31_14:13:48.492 DEBUG org.a.utils.ConfigUtils        - >  getParameter(DEFAULT, test)
07-31_14:13:48.505 DEBUG org.a.utils.ConfigUtils        - .     getParameter(DEFAULT, test) -> (null)
07-31_14:13:48.506 DEBUG org.a.utils.ConfigUtils        - . getParameter(test) -> (null)
```

LOGGER declaration

```
import net.sf.aspect4log.Log;
import static net.sf.aspect4log.Log.Level.TRACE;

@Log ①
public class FooDao {

    public void tooLowLevelFunction(){ ②
        //[...]
    }

    @Log(enterLevel = Level.TRACE, exitLevel = Level.TRACE) ③
    public void delete(String foo) {
        //[...]
    }

    @Log(argumentsTemplate = "[...skipped...]", resultTemplate = "[...skipped...]") ④
    public void find(String bigXML) {
        //[...]
    }

    @Log(on = { @Exceptions(exceptions = { CgException.class }, level = Level.INFO) }) ⑤
    public void saveOrUpdate(String foo) {
        //[...]
    }
}
```

① @Log on a class will affect every methods not annotated

② So this method will be logged, in DEBUG by default

③ Lower the level to TRACE if some methods pollute the logs

④ You can skip only the arguments/results if they are too verbose

⑤ Some advanced functionnality are available, see the website

For runtime, have log4j & aspect4log configuration files in the classpath, examples : link:log4j2.xml & link:aspect4log.xml.

Dependencies

```
<dependencies>
    <!-- for @Log -->
    <dependency>
        <groupId>net.sf.aspect4log</groupId>
        <artifactId>aspect4log</artifactId>
        <version>1.0.7</version>
    </dependency>
    <!-- AspectJ for instrumentation -->
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjrt</artifactId>
        <version>1.8.9</version>
    </dependency>
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjtools</artifactId>
        <version>1.8.9</version>
    </dependency>
</dependencies>

<plugins>
    <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>aspectj-maven-plugin</artifactId>
        <version>1.7</version>
        <executions>
            <execution>
                <goals>
                    <goal>compile</goal>
                </goals>
            </execution>
        </executions>
        <configuration>
            <showWeaveInfo>false</showWeaveInfo>
            <Xlint>adviceDidNotMatch=ignore,noGuardForLazyTjp=ignore</Xlint>
            <aspectLibraries>
                <aspectLibrary>
                    <groupId>net.sf.aspect4log</groupId>
                    <artifactId>aspect4log</artifactId>
                </aspectLibrary>
            </aspectLibraries>
        </configuration>
        <dependencies>
            <dependency>
                <groupId>org.aspectj</groupId>
                <artifactId>aspectjtools</artifactId>
                <version>1.8.9</version>
            </dependency>
        </dependencies>
    </plugin>
</plugins>
```

1.5. Log methods duration

1.5.1. using JCabi @Loggable

See https://aspects.jcabi.com/annotation-loggable.html

With AOP, get selected methods duration :

```
2016-10-11 14:22:52.716 [main] INFO  PERFORMANCES - #setTestMode(...): in 30,51ms
2016-10-11 14:22:52.857 [main] INFO  PERFORMANCES - #setTestMode(...): in 1,20ms
```

Loggable example

```
@Loggable(skipArgs = true, skipResult = true, name = "PERFORMANCES")
public static void topLevelJarFunction(IData pipeline) throws ServiceException {
    //[...]
}
```

2. Best practices

2.1. Java

2.1.1. Java packages & classes naming

- Best package organization is by fonctionnality first, and then technically when many classes of the same type
- Always put classes in subpackage of the project
  - If a java project is **bar-a-b**, all packages are **mycorp.bar.a.b.***
- Don't use different packages for a few classes, regroup them (if below or equal 3 classes by package)
- Don't put in the class name what is already in the package name, except for too generic file name

Some naming conventions

http://stackoverflow.com/questions/3226282/are-there-best-practices-for-java-package-organisation
http://www.javapractices.com/topic/TopicAction.do?Id=205

Some widely used examples

http://commons.apache.org/proper/commons-lang/javadocs/api-2.6/overview-tree.html
https://commons.apache.org/proper/commons-lang/apidocs/overview-tree.html

2.1.2. Java 7 try with closable objects

Before Java 7, you had to close() streams and other closable objects in a try/catch/finally. Now Java handles everything if you use the right pattern :

try-with-resource

```
try (
    ZipOutputStream zos = new ZipOutputStream(new FileOutputStream(dstDirectory + "/" + fileName + ".zip"));
    FileInputStream in = new FileInputStream(foundFile.getAbsolutePath())
    ) {
    ZipEntry ze = new ZipEntry(fileName);
    zos.putNextEntry(ze);

    int len;
    while ((len = in.read(buffer)) > 0) {
        zos.write(buffer, 0, len);
    }

    if (delete)
        foundFile.delete();
} catch (IOException e) {
    LOGGER.error("Unable to zip or delete the file=" + srcDirectory + "/" + fileName + ", dest=" + dstDirectory, e);
    throw e;
}
```

2.1.3. Static Java Maps

When a **Map** is static (and then accessed by multiple threads), declare it Map and instantiate it **ConcurrentHashMap** :

Thread-safe Map

```
Map<a,b> myMap == new ConcurrentHashMap<>();
```

Idem for a **Set** but this is a bit tricky :

Thread-safe Set

```
Set<String>
mySet = Collections.newSetFromMap(new ConcurrentHashMap<String,Boolean>());
```

2.1.4. Init on demand

For objects used by static functions, try to initialize them only once and do it in thread safe mode.

Init on demand pattern

```
public class Something {
    private Something() {}

    private static class LazyHolder {
        private static final Something INSTANCE = new Something();
    }

    public static Something getInstance() {
        return LazyHolder.INSTANCE;
    }
}
```

2.1.5. Enum and String

A String from an Enum must be used with a custom **toString()**, never with **getName()** or default

toString().

Enum.toString() pattern

```
// Natures d echange
public enum EsbNatureType {
     DIFFUSION_FICHIER("DiffusionFichier"), DIFFUSION_MESSAGES("DiffusionMessages");

     private String name = null;

     EsbNatureType(String nameString) {
            this.name = nameString;
     }

     @Override
     public String toString() {
            return this.name;
     }

};
```

If you don't do this way, we loose the flexibility to rename either the Enum or the String.

2.1.6. MyEnum.toEnum(String)

Comment déclarer l'Enum :

toEnum pattern

```
public enum ServiceOption {
    COMPLEMENTS,
    RESTRICTIONS,
    RISQUES,
    SNGI_EM_DECEDE,
    SNGI_EM_NON_IDENT,
    SNGI_ID_OBLIGATOIRE,
    DCR,
    DCR_STATUT,
    DCR_DELAI,
    LISTE_PSORTANTS,
    ID_TIERS,
    NOM_FLUX_SORTIE,
    ABO_ACTIF,
    DENOM_METIER,
    REF_ABO,
    UNKNOWN;

    public static ServiceOption toEnum(String optionName) {
        switch (optionName) {
        case "priseEnCptLstRisque":
            return RISQUES;
        case "priseEnCptLstCompl":
            return COMPLEMENTS;
        default:
            return UNKNOWN;
        }
    }
}
```

Puis ton Builder tu fais un **ServiceOption.toEnum(tonOption)** et le tour est joué.

Never write files outside of target/

3. Appendix

3.1. Revision marks

Differences since last tag