Java Best Practices

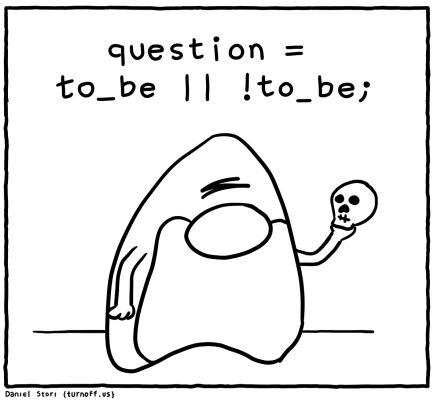
Version 1.18-SNAPSHOT

2018-11-01

Table of Contents

Table 1. History

Date	Author	Detail
2018-10-21	bcouetil	reveal.js presentation
2018-09-19	bcouetil	- Sample asciidoctor maven project published on Github - Github & LinkedIn links - Sample project tree - new images + resizing and positioning
2018-09-05	bcouetil	Minor changes
2018-08-29	bcouetil	Asciidoc HTML look & feel changes
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

SLF4J helps abstract logging from implementation. Even for libraries using log4j explicitely with the concept of bridge.

Maven dependencies

```
<!-- for SLF4J implementing log4j2 -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>1.7.25</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.11.1</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.11.1</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.11.1</version>
</dependency>
<!-- to bridge any LOG4J1 usage to SLF4J -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>log4j-over-slf4j</artifactId>
    <version>1.7.25</version>
</dependency>
```

Since slf4j does not accept multiple boot jars, you may have to exclude log4j from dependencies :

Exclude log4j JAR

```
<dependency>
    <groupId>my.bad.group</groupId>
    <artifactId>my-bad-artifact</artifactId>
    <version>1.0.0</version>
    <exclusions>
        <exclusion>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

1.4. Log4j2

Rules for a good LOG4J2 logging

- Aligned messages
- Complete but short context data
  - No need for years, even for dates if files are named with date
  - truncated package name
  - no or minimal separators

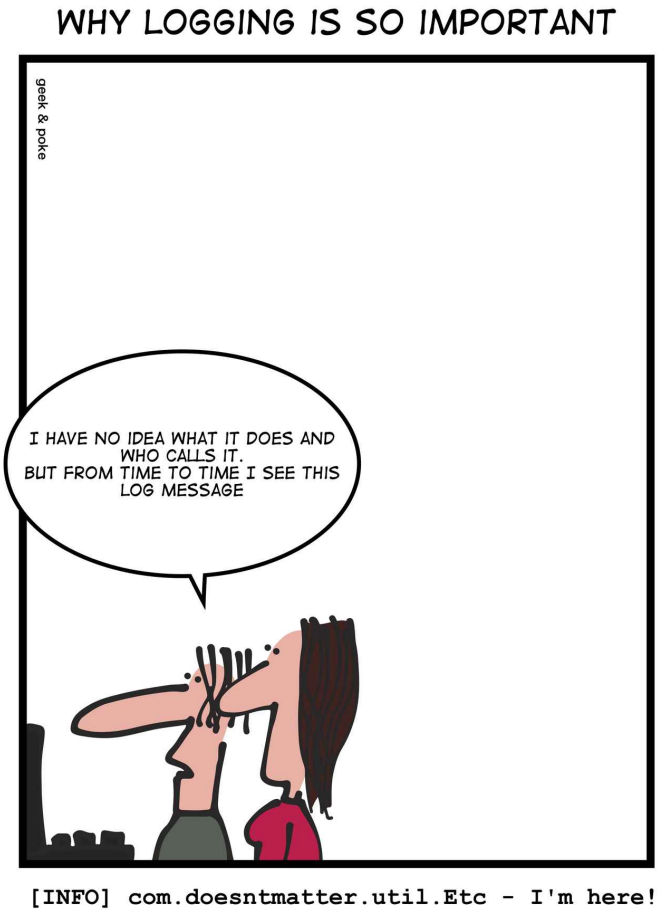
- Separate files for modules
- Separate level files
- Rolling files not to handle deletion

Logs Example

```
17:53:33.372 DEBUG main  .wm.utils.wmcall.WmHelper:176  + setTestMode()
17:53:33.372 DEBUG main  cg.wm.utils.wmcall.WmCall:176  +   WmCall()
17:53:33.372 DEBUG main  cg.wm.utils.wmcall.WmCall:176  .   WmCall() -> (null)
17:53:33.373 DEBUG main  tils.wmcall.WmCallEclipse:176  +   WmCallEclipse()
17:53:33.373 DEBUG main  tils.wmcall.WmCallEclipse:176  .   WmCallEclipse() -> (null)
17:53:33.373 DEBUG main  .wm.utils.wmcall.WmHelper:176  . setTestMode()
17:53:33.374 DEBUG main  cg.wm.utils.ConfigUtils  :176  + healthCheck()
```

log4j2.xml example

```
Unresolved directive in BP-java.adoc - include::/builds/bcouetil/academy/cg-utils\src\test\resources\log4j2.xml[]
```



1.5. 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 by default

② ...so this method will be logged in/out, in DEBUG by default

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

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>
```

1.6. Log methods duration

1.6.1. 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 {
    //[...]
}
```

1.7. JUnit

1.7.1. JUnit 4.9+ : Real time status and duration

Sometimes on JUnit test classes, executions are very long, so it can be nice to see in real time which test is running. Here is what you could expect :

```
Running cg.msg.tracker.ui.MainFrameIT
11:23:37.814 [main] DEBUG TEST - Running aboutTest...
11:23:38.503 [main] DEBUG TEST - ...OK in 0.689S
11:25:17.561 [main] DEBUG TEST - Running updateLAFTryNextTest...
11:25:20.865 [main] DEBUG TEST - ...OK in 3.304S
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 103.443 sec - in cg.msg.tracker.ui.MainFrameIT
```

And here is how you do it.

Inner Class in your test class

```
/**
 * The Class LogTestName.
 */
public static class LogTestRule extends TestWatcher {

    Instant startingDate = Instant.now();

    @Override
    protected void failed(Throwable e, Description description) {
        LOGGER.debug("...KO [{}]", e.getMessage());
    }

    @Override
    protected void starting(Description description) {
        LOGGER.debug("Running {}...", description.getMethodName());
        startingDate = Instant.now();
    }

    @Override
    protected void succeeded(Description description) {
        LOGGER.debug("...OK in {}", Duration.between(startingDate, Instant.now()).toString().substring(2));
    }
}
```

Declare it's usage in the class with with @Rule

```
@Rule
public LogTestRule logTestRule = new LogTestRule());
```

1.7.2. JUnit 4.9+ : Retry on error

You could want to retry test on error a fixed number of time. And the retry mecanism could require some custom clean up. Here we will combine this rule and the previous one. Si you could expect this :

```
16:48:35.176 [main] DEBUG TEST - Running buttonsTest...
[WARNING] buttonsTest(cg.msg.tracker.ui.ConnectionIT): Run 1 failed [Condition with alias 'broker radio button should be
selected' didn't complete within 30 seconds because condition with lambda expression in
cg.msg.tracker.ui.utils.ParentAssertJTestCase that uses org.assertj.swing.fixture.JRadioButtonFixture was not
fulfilled.]
16:49:21.574 [main] DEBUG TEST - ...OK in 46.394S
16:49:21.665 [main] DEBUG TEST - Running connectionInfoZoneTextTest...
16:49:55.874 [main] DEBUG TEST - ...OK in 0.714S
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 85.826 sec - in cg.msg.tracker.ui.ConnectionIT
```

And here is how you do it.

Inner Class in your test class

```
/**
 * The Class RetryRule.
 */
public static class RetryRule implements TestRule {

    private final static int TRY_COUNT = 3;

    @Override
    public Statement apply(Statement base, Description description) {
        return statement(base, description);
    }

    private Statement statement(final Statement base, final Description description) {
        return new Statement() {
            @Override
            public void evaluate() throws Throwable {
                Throwable caughtThrowable = null;

                // implement retry logic here
                for (int i = 0; i < TRY_COUNT; i++) {
                    try {
                        base.evaluate();
                        return;
                    } catch (Throwable t) {
                        caughtThrowable = t;
                        // [WARNING] for a colorful Jenkins build
                        System.out.println("[WARNING] " + description.getDisplayName() + ": Run " + (i + 1)
                                + " failed [" + t.getMessage() + "]");
                        LOGGER.warn(CgException.prettyPrint(t));
                        commonAfterClass();
                        commonBeforeClass();
                    }
                }
                // [ERROR] for a colorful Jenkins build
                System.out.println("[ERROR] " + description.getDisplayName() + ": Giving up after " + TRY_COUNT
                        + " failures.");
                throw caughtThrowable;
            }
        };
    }

}
```

Double rules usage

```
@Rule
public RuleChain chain = RuleChain.outerRule(new LogTestName()).around(new RetryRule());
```

1.7.3. Various patterns

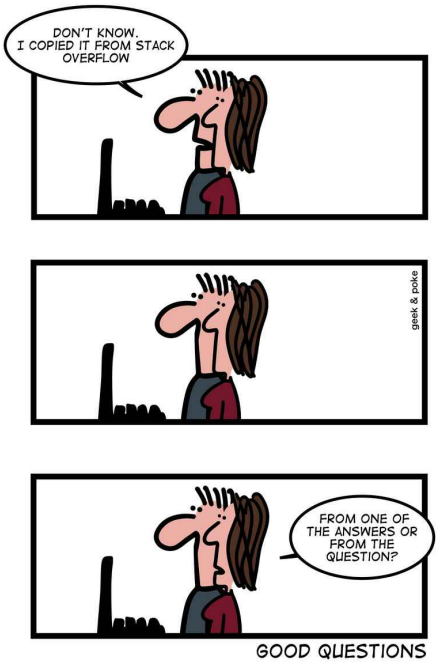
Expected exception

```
@Test(expected = IllegalArgumentException.class)
    public void testFromStringUnknown() {
        CgPackage.fromString("Unknown");
    }
```

Wait until a background task is done

```
Awaitility.await("Broker radio button should be visible").until(() -> mCEditor.getBrokerRadioButton().isShowing());
```

2. Best practices



2.1. 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

2.2. 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;
}
```

2.3. 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>());
```

2.4. 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;
    }
}
```

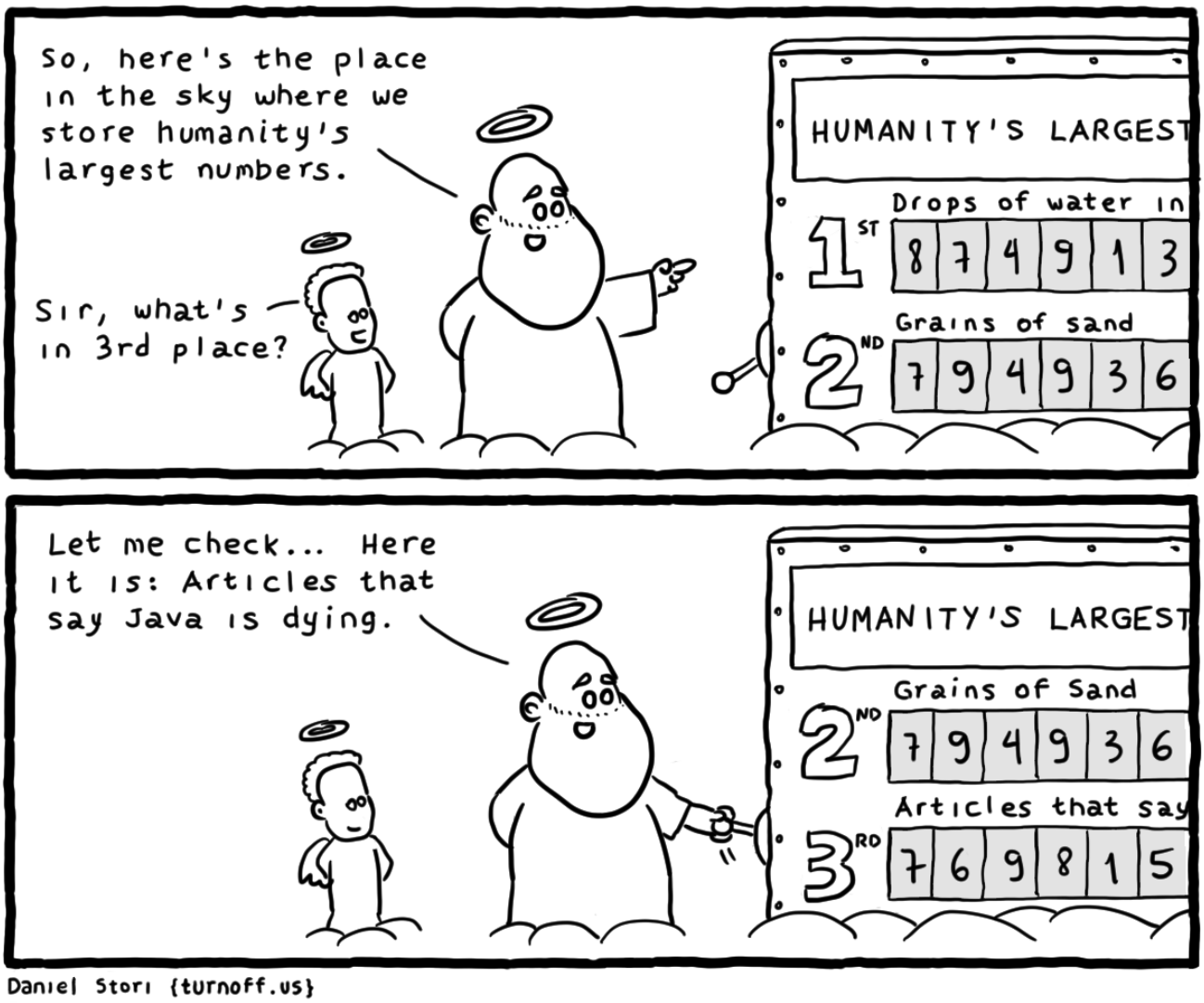
2.5. Enums and Strings

Enum Class Example

```
package cg.wm.utils;

/**
 * The Enum CgPackage.
 */
public enum CgPackage {

    /** The default. */
    DEFAULT("DEFAULT"),
    /** The cg utils. */
    CG_UTILS("CgUtils"),
    /** The cg elastic. */
    CG_ELASTIC("CgElastic");

    /** The internal string. */
    private String str;

    /**
     * Instantiates a new package.
     *
     * @param str the str
     */
    private CgPackage(String str) {
        this.str = str;
    }

    /**
     * From string.
     *
     * @param input the input
     * @return the package
     * @throws IllegalArgumentException the illegal argument exception
     */
    public static CgPackage fromString(String input) throws IllegalArgumentException {
        for (CgPackage p : CgPackage.values()) {
            if (p.str().equals(input)) {
                return p;
            }
        }
        throw new IllegalArgumentException("Unknown package=" + input);
    }

    /**
     * Custom, short-named toString().
     * Don't use defaults name() or toString(), they'll give the strict enum name
     *
     * @return the string
     */
    public String str() {
        return this.str;
    }

}
```

2.6. Single Implementation Interfaces Are Evil

See https://www.symphonious.net/2011/06/18/the-single-implementation-fallacy/


So, here's the place in the sky where we store humanity's largest numbers.
Sir, what's in 3rd place?
HUMANITY'S LARGEST
Drops of water in
1ST
8 7 4 9 1 3
Grains of sand
2ND
7 9 4 9 3 6
Let me check... Here it is: Articles that say Java is dying.
HUMANITY'S LARGEST
Grains of Sand
2ND
7 9 4 9 3 6
Articles that say
3RD
7 6 9 8 1 5
Daniel Stori {turnoff.us}

3. Appendix

3.1. Revision marks

Differences since last tag