Java objects on steroids

by Romain Rochegude

Introduction

- Many ways to see "objects": POJO, bean
- Many concepts: encapsulation, inheritance, polymorphism, immutability
- Who Is an Object?
- Seven Virtues of a Good Object

1. Write simple immutable object

```
public final class User {

    private final int id;
    private final String login;
    private final String avatarUrl;

    public User(
            final int pId,
            final String pLogin,
            final String pAvatarUrl) {
        id = pId;
        login = pLogin;
        avatarUrl = pAvatarUrl;
    }

}
```

2. Improve pojo methods testing with pojo-tester

http://www.pojo.pl/

Gradle installation:

```
repositories {
    jcenter()
}

dependencies {
    testCompile 'pl.pojo:pojo-tester:${latest-version}'
}
```

Write unit test:

```
import pl.pojo.tester.api.assertion.Method;
import static pl.pojo.tester.api.assertion.Assertions.assertPojoMethodsFor;

public class UserTest {
    @Test
    public void Should_Pass_All_User_Tests() {
        assertPojoMethodsFor(
                User.class
        ).testing(
                Method.EQUALS,
                Method.HASH_CODE,
                // Method.SETTER,
                // Method.GETTER,
                // Method.TO_STRING,
                Method.CONSTRUCTOR
        ).areWellImplemented();
    }
}
```

Why?

- No getters/setters testing?
  - Because getters/setters are evil
- No `toString`?
  - Because it expects a specific format that is not extensible

The result is:

```
Class fr.guddy.joos.domain.User has bad 'hashCode' method implementation.
The hashCode method should return same hash code for equal objects.
Current implementation returns different values.
Object:
fr.guddy.joos.domain.User@7946e1f4
and
fr.guddy.joos.domain.User@5cc7c2a6
have two different hash codes:
2034688500
and
1556595366
```

3. Improve pojo methods writing with pojomatic

http://www.pojomatic.org/

Why pojomatic instead of Commons Lang, Guava or Lombok?

- Because pojomatic is only focused on the `equals(Object)`, `hashCode()` and `toString()` methods
- Because Commons Lang are verbose
- Because Guava has many other features
- Because Lombok needs an extra plugin and

Gradle installation:

```
compile 'org.pojomatic:pojomatic:2.0.1'
```

Configure object:

```
import org.pojomatic.Pojomatic;
import org.pojomatic.annotations.Property;

public final class User {
    @Property
    private final int id;
    // ...

    @Override
    public boolean equals(final Object pObj) {
        return Pojomatic.equals(this, pObj);
    }

    @Override
    public int hashCode() {
        return Pojomatic.hashCode(this);
    }
}
```

4. Improve immutable writing with auto-value

https://github.com/google/auto/tree/master/value

Gradle installation:

```
compile 'com.google.auto.value:auto-value:1.2'
annotationProcessor 'com.google.auto.value:auto-value:1.2'
```

Configure object:

```
import com.google.auto.value.AutoValue;

@AutoValue
public abstract class Repo {
    public abstract int id();
    public abstract String name();
    public abstract String description();
    public abstract String url();

    public static Repo create(
            final int pId,
            final String pName,
            final String pDescription,
            final String pUrl) {

        return new AutoValue_Repo(pId, pName, pDescription, pUrl);
    }
}
```

5. Improve object testing with AssertJ Assertions Generator

http://joel-costigliola.github.io/assertj/assertj-assertions-generator.html

Inspired by Single Statement Unit Tests, for the following benefits:

- Reusability
- Brevity
- Readability
- Immutability
- Fluent test result

Gradle installation:

AssertJ dependency:

```
testCompile 'org.assertj:assertj-core:3.8.0'
```

assertjGen plugin installation:

```
buildscript {
    repositories {
        maven { url "https://plugins.gradle.org/m2/" }
    }
    dependencies {
        classpath "gradle.plugin.com.github.opengl-8080:
          assertjGen-gradle-plugin:1.1.3"
    }
}
```

assertjGen plugin configuration:

```
assertjGen {
    classOrPackageNames = ['fr.guddy.joos.domain']
    outputDir = 'src/test/java'
    cleanOnlyFiles = true
    assertjGenerator = 'org.assertj:assertj-assertions-generator:2.0.0'
}
```

Run the `assertjGen` Gradle task to generate assertion classes

Write unit test:

```
import static fr.guddy.joos.domain.UserAssert.assertThat;

public class UserTest {
    @Test
    public void Should_Have_Matching_Id() {
        assertThat(
                new User(12,
                        "Romain",
                        "https://...")
        ).hasId(13);
    }
}
```

The result is:

```
java.lang.AssertionError:
Expecting id of:
  <User{id: {12}, login: {Romain}, avatarUrl: {https://...}}>
to be:
  <13>
but was:
  <12>
```

Conclusion

- Focus on the objects
- Less boilerplate code
- On the way to DDD
- On the way to hexagonal architecture

Bibliography

- http://www.yegor256.com/2014/11/20/seven-virtues-of-good-object.html
- http://www.yegor256.com/2016/07/14/who-is-object.html
- http://www.yegor256.com/2014/09/16/getters-and-setters-are-evil.html
- http://www.yegor256.com/2017/05/17/single-statement-unit-tests.html
- http://thierry-leriche-dessirier.developpez.com/tutoriels/java/simplifier-code-guava-lombok/

Logo credits

Business graphic by freepik from Flaticon is licensed under CC BY 3.0.
Made with Logo Maker