Relatório de GoF e Smells

# Code Smells

## Comentários

### Falta de comentários

Ao longo do código, foi notada uma grande falta de comentários, quer seja a descrever uma classe, uma interface ou até mesmo simples métodos. Algumas classes chegam mesmo a ter unicamente o @author como comentário. Qualquer novo desenvolvedor do projeto terá bastante dificuldade em compreender o que um certo pedaço de código faz, ou deve fazer.

**Exemplo:** Classe abstrata GPCalendarBase

**Localização:** ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.calendar.GPCalendarBase

### Código antigo em comentário

Em múltiplas classes, é possível encontrar vestígios de código antigo, ou possível código novo, por implementar, em comentário. Por vezes, aparentam ser apenas cópias do código acima, com pequenas diferenças. Para resolver isto, seria necessário passar por todas as classes e apagar os respetivos comentários.

**Exemplo:**

timeunitHeaderBorder.setStyle("timeline.lineSplitter");

//timeunitHeaderBorder.setForegroundColor(myInputApi.getTimelineBorderColor());

**Localização:** ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.chart.scene.TimelineSceneBuilder

**Linhas:** 107 e 108, respetivamente.

### Comentário a relembrar um raciocínio

Foram, também, avistados comentários que servem como lembretes de raciocínio, completamente inconsistente com o resto dos comentários. No caso do exemplo a baixo, temos o raciocínio de um ‘if’ explicado e as suas consequências, enquanto que o resto da classe tem uma grande falta de comentários.

**Exemplo:**

if (dependeeVector.getHProjection().reaches(dependantVector.getHProjection().getPoint())) {

// when dependee.end <= dependant.start && dependency.type is

// any

// or dependee.end <= dependant.end && dependency.type==FF

// or dependee.start >= dependant.end && dependency.type==SF

Point first = new Point(dependeeVector.getPoint().x, dependeeVector.getPoint().y);

.

.

.

**Localização:** ganttproject.biz.ganttproject.src.main.java.biz.ganttproject.core.chart.scene.gantt.DependencySceneBuilder.java

**Linha:** 85-89

### TODO comentado

A utilização de comentários para informar os outros programadores que um método se encontra incompleto ou com problemas é também um ‘Code Smell’. A retirada deste pedaço de código ou o término deste permitirá que os outros programadores olhem para o código de maneira menos confusa, sendo também mais acessível para estes darem o seu contributo.

**Exemplo:**

private static Duration convertLag(TaskDependency dep) {

// TODO(dbarashev): Get rid of days

return Duration.getInstance(dep.getDifference(), TimeUnit.DAYS);

}

**Localização:**

ganttproject/biz.ganttproject.impex.msproject2/src/main/java/biz/ganttproject/impex/msproject2/ProjectFileExporter.java

**Linhas:** 303-306

### Excesso de código

Apesar de código que não apresenta qualquer comentário representa um Code Smells oposto também é considerado Code Smell. Demasiados comentário ao código escrito não permite que um programador consiga entender o código de forma tao eficiente.

**Exemplo :**

/\*\*

\* Class used to implement performant, high-quality and intelligent image

\* scaling and manipulation algorithms in native Java 2D.

\* <p/>

\* This class utilizes the Java2D "best practices" for image manipulation,

\* ensuring that all operations (even most user-provided {@link BufferedImageOp}

\* s) are hardware accelerated if provided by the platform and host-VM.

\* <p/>

\* <h3>Image Quality</h3>

\* This class implements a few different methods for scaling an image, providing

\* either the best-looking result, the fastest result or a balanced result

\* between the two depending on the scaling hint provided (see {@link Method}).

\* <p/>

.

.

.

\*\*/

**Localização:**

ganttproject/ganttproject/src/main/java/org/imgscalr/Scalr.java

**Linhas:** 35-196

## Classes

### Classe que devia ser um enumerado

No código, existe pelo menos uma classe que contém apenas valores constantes de um mesmo Objeto. A classe ShapeConstants consiste apenas em 21 constantes de tipo ShapePaint e uma lista com todas estas constantes, sem qualquer variável ou método associados. Este tipo de classe devia, em vez disso, ser considerado um enumerado.

**Exemplo:**

public static final ShapePaint TRANSPARENT = new ShapePaint(4, 4, new int[] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0 });

.

.

.

public static ShapePaint[] PATTERN\_LIST = { TRANSPARENT, DEFAULT, CROSS, VERT, HORZ, GRID, ROUND, NW\_TRIANGLE,

NE\_TRIANGLE, SW\_TRIANGLE, SE\_TRIANGLE, DIAMOND, DOTS, DOT, SLASH, BACKSLASH, THICK\_VERT, THICK\_HORZ, THICK\_GRID,

THICK\_SLASH, THICK\_BACKSLASH };

**Localização:**

ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.chart.render.ShapeConstants.java

**Linhas:** 26-27 e 89-91, respetivamente.

### Classe com multiplos métodos, definida dentro de uma interface

Na Iterface ColumnList podemos encontrar definidas mais uma interface e uma classe com múltiplos métodos. Isto torna o código bastante confuso e difícil de ler. Sendo assim, seria preferível separar esta classe e interface extra.

**Exemplo:**

public interface Column {...}

class ColumnStub implements ColumnList.Column {...}

**Localização:** ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.table.ColumnList.java

**Linhas:** 25 e 60, respetivamente.

### Classe vazia

Foi criada, também, uma classe totalmente vazia, que não chega a ser utilizada em lado nenhum. Esta classe pode ser considerada 'dead code', caso simplesmente não seja utilizada, ou então 'speculative generality' caso o seu autor tenha intenção de a usar, no início, mas acabou por não ser necessária.

**Exemplo:**

package biz.ganttproject.impex.msproject2;

public class WebStartIDClass {

}

**Localização:** ganttproject.biz.ganttproject.impex.msproject2.src.main.java.biz.ganttproject.impex.msproject2.WebStartDClass.java

**Linhas:** 1-5

### Classe muito extensa

Ao longo do código, encontramos várias classes anormalmente extensas. Um exemplo destas é a classe Canvas, com mais de 708 linhas de código. A maior parte destas é devida ao facto de estar a declarar várias classes, como a Rhombus, Arrow, Line. Classes estas que podiam ser declaradas individualmente, diminuído, assim, o peso sobre a classe Canvas.

**Exemplo:**

public static class Polygon extends Shape {...}

public static class Rectangle extends Polygon {...}

public static class Rhombus extends Polygon {...}

public static class Arrow extends Shape {...}

.

.

.

**Localização:** ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.chart.canvas.Canvas.java

**Linhas:** 148, 230, 246, 257, respetivamente.

## Métodos

### Método demasiado grande e complexo

Na classe DateParser existe um método com um total de 122 linhas, o que o torna confuso e difícil de ler. Este método pode ser facilmente decomposto em múltiplos métodos auxiliares de nomes fáceis de ler.

**Exemplo:**

private static Calendar getCalendar(String isodate)

throws InvalidDateException {

.

.

.

}

**Localização:** ganttproject.biz.ganttproject.core.src.main.java.org.w3c.util.DateParser.java

**Linhas:** 35-157

### Utilização de múltiplos if, em vez de um único switch

Na classe GPTimeUnitStack, o método encode(TimeUnit timeUnit) utiliza vários if, uns atrás dos outros. Estes if podiam ser facilmente substituídos por um único switch, cujo default envia uma exceção.

**Exemplo:**

public String encode(TimeUnit timeUnit) {

if (timeUnit == HOUR) {

return "h";

}

if (timeUnit == DAY) {

return "d";

}

if (timeUnit == WEEK) {

return "w";

}

throw new IllegalArgumentException();

}

**Localização:** ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.time.impl.GPTimeUnitStack.java

**Linhas:** 124-135

### Duplicação de código

A duplicação de código de maneira a preencher todos os dias de um calendário retrata uma má prática de código (code smell) muito comum, a repetição de código. Com o objetivo de eliminar a repetição de código, é possível recorrer a um ciclo {for each} de maneira a percorrer todos os valores pertencentes ao Enum {Day}, chamando desta forma a função {calendar.setWorkingDay(…)} uma vez apenas.

**Exemplo:**

private void exportWeekends(ProjectCalendar calendar) {

ProjectCalendarHours workingDayHours = calendar.getCalendarHours(Day.MONDAY);

calendar.setWorkingDay(Day.MONDAY, isWorkingDay(Calendar.MONDAY));

calendar.setWorkingDay(Day.TUESDAY, isWorkingDay(Calendar.TUESDAY));

calendar.setWorkingDay(Day.WEDNESDAY, isWorkingDay(Calendar.WEDNESDAY));

calendar.setWorkingDay(Day.THURSDAY, isWorkingDay(Calendar.THURSDAY));

calendar.setWorkingDay(Day.FRIDAY, isWorkingDay(Calendar.FRIDAY));

calendar.setWorkingDay(Day.SATURDAY, isWorkingDay(Calendar.SATURDAY));

if (calendar.isWorkingDay(Day.SATURDAY)) {

copyHours(workingDayHours, calendar.addCalendarHours(Day.SATURDAY));

}

calendar.setWorkingDay(Day.SUNDAY, isWorkingDay(Calendar.SUNDAY));

if (calendar.isWorkingDay(Day.SUNDAY)) {

copyHours(workingDayHours, calendar.addCalendarHours(Day.SUNDAY));

}

**Localização:** ganttproject/biz.ganttproject.impex.msproject2/src/main/java/biz/ganttproject/impex/msproject2/ProjectFileExporter.java

**Linhas:** 110-124

### Métodos não utilizados

Um outro Code Smell que detetamos foi o facto de existirem métodos não utilizados, ou seja, código morto. Estes podem confundir o programador e estendem o código desnecessariamente. Uma solução possível é retirá-los e apenas adicioná-los quando forem realmente necessários.

**Exemplo:**

public void setMaxLength(int maxLength) {

myMaxLength = maxLength;

}

public int getMaxLength() {

return myMaxLength;

}

**Localização:** ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.chart.canvas.Canvas.java

**Linhas:** 394-400

## Variáveis/Constantes

### Valores que deviam ser constantes

Especialmente na classe WeekendCaledarImpl, foi notado o uso repetitivo de certos valores, cujo significado é igual. Neste caso, o 7, que representa o número de dias da semana. Em vez de criar uma constante, apenas é repetido o número ao longo de toda a classe, cada vez que é necessário. Isto dificulta muitas vezes a compreensão do código e obrigada o desenvolvedor a encontrar todas as vezes que o número foi mencionado, caso tenha intenção de o alterar.

**Exemplo:**

private final DayType[] myTypes = new DayType[7];

for (int i = 1; i <= 7; i++) {…}

**Localização:** ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.calendar.WeekendCalendarImpl

**Linha:** 64 e 364, respetivamente"

### Constantes não utilizadas

Foram também encontradas constantes que nunca são utilizadas no código. Estas, naturalmente, também representam um Code Smell, uma vez que podem confundir um novo desenvolvedor. Sendo assim, a melhor solução é retirá-las, até serem realmente necessárias.

**Exemplo:**

public static final int ALREADY\_EXIST = 0;

public static final int DO\_NOT\_EXIST = 1;

**Localização:** ganttproject.ganttproject.src.main.java.biz.ganttproject.customproperty.CustomColumnsException.java

**Linhas:** 29-31