Engenharia de Software (MiEI) – 2022/2023 – 1º Semestre

Relatório Code Smells e Patterns

Gantt Project

Realizado por:

David Moreira nº 59984

Joana Maroco nº60052

João Lopes nº60055

José Romano nº59241

Sofia Monteiro nº60766

Capítulo 1 - Code Smells

- Comentários

Falta de comentários -Autor João Lopes, Review José Romano

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 -Autor Joana Maroco, Review David Moreira

Referenciando, em primeiro lugar, um problema encontrado relativamente aos comentários elaborados no código. Em múltiplas classes, é possível encontrar em comentário vestígios de código antigo (ou possível código novo) por implementar. Por vezes, aparentam ser apenas cópias do código acima.

Pelas linhas 107 e 108 de ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.chart.scene.TimelineSceneBuilder podemos encontrar, nomeadamente:

Exemplo:

```
106            finishx: sizex - 1, spanningHeaderHeight);
107        timeunitHeaderBorder.setStyle("timeline.lineSplitter");
108        //timeunitHeaderBorder.setForegroundColor(myInputApi.getTimelineBorderColor());
109        Canvas.Line bottomBorder = getTimelineContainer().createLine( startx: 0, headerHeight,  finishx: sizex - 2,
```

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

Linhas: 107 e 108, respetivamente.

Comentário a relembrar um raciocínio – Autor David Moreira, Review João Lopes

Durante a observação e análise do código do open source, reparei na existência de comentários que servem como lembretes de raciocínio, completamente inconsistente com o resto dos comentários. No caso do exemplo a baixo é possível verificar a extensa explicação do funcionamento do if e das suas consequências, enquanto o resto da classe tem uma grande falta de comentários.

Exemplo:

```
84
85      if (dependeeVector.getHProjection().reaches(dependantVector.getHProjection().getPoint())) {
86        // when dependee.end <= dependant.start && dependency.type is
87        // any
88        // or dependee.end <= dependant.end && dependency.type==FF
89        // or dependee.start >= dependant.end && dependency.type==SF
90        Point first = new Point(dependeeVector.getPoint().x, dependeeVector.getPoint().y);
91
```

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

Linha: 85-89

TODO comentado -Autor José Romano, Review Joana Maroco

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:

```
1 usage   dbarashev
303 @   private static Duration convertLag(TaskDependency dep) {
304       // TODO(dbarashev): Get rid of days
305       return Duration.getInstance(dep.getDifference(), TimeUnit.DAYS);
```

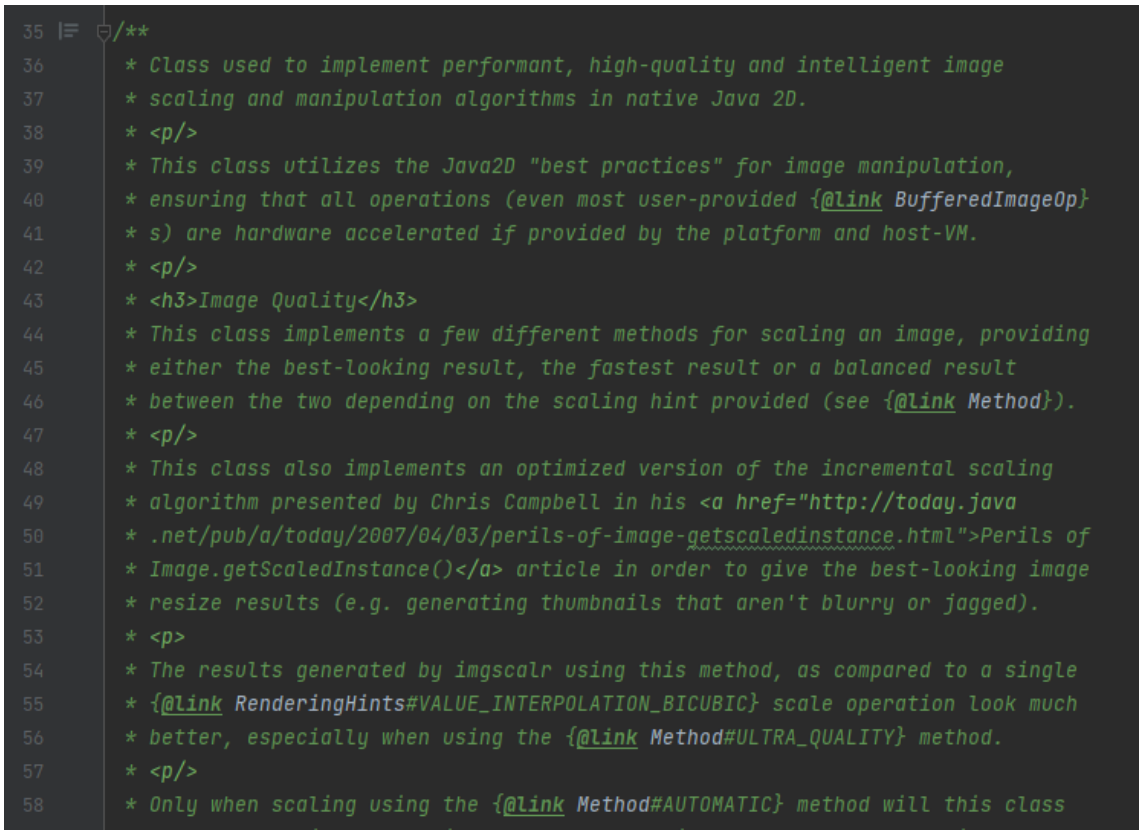
Localização:
ganttproject/biz.ganttproject.impex.msproject2/src/main/java/biz/ganttproject/impex/msproject2/ProjectFileExporter.java

Linhas: 303-306

Excesso de comentários -Autor David Moreira, Review Joana Maroco

Apesar de na maior parte das classes não serem quase apresentados comentários, o que por si só é um code smell, existem classes, como a deste exemplo, na qual existem demasiados comentários; o que acaba também por ser um code smell. Demasiados comentários numa pequena porção de código não permitem 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/>
 * This class also implements an optimized version of the incremental scaling
 * algorithm presented by Chris Campbell in his <a href="http://today.java
 * .net/pub/a/today/2007/04/03/perils-of-image-getscaledinstance.html">Perils of
 * Image.getScaledInstance()</a> article in order to give the best-looking image
 * resize results (e.g. generating thumbnails that aren't blurry or jagged).
 * <p>
 * The results generated by imgscalr using this method, as compared to a single
 * {@link RenderingHints#VALUE_INTERPOLATION_BICUBIC} scale operation look much
 * better, especially when using the {@link Method#ULTRA_QUALITY} method.
 * <p/>
 * Only when scaling using the {@link Method#AUTOMATIC} method will this class
```

Localização:
ganttproject/ganttproject/src/main/java/org/imgscalr/Scalr.java
Linhas: 35-196

- Classes

Classe que devia ser um enumerado -Autor Sofia Monteiro, Review Joana Maroco

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:

```
2 usages
26  public static final ShapePaint TRANSPARENT = new ShapePaint( width: 4,  height: 4, new int[] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
27      0, 0, 0 });
88
    3 usages
89  public static ShapePaint[] PATTERN_LIST = { TRANSPARENT, DEFAULT, CROSS, VERT, HORZ, GRID, ROUND, NW_TRIANGLE,
90      NE_TRIANGLE, SW_TRIANGLE, SE_TRIANGLE, DIAMOND, DOTS, DOT, SLASH, BACKSLASH, THICK_VERT, THICK_HORZ, THICK_GRID,
91      THICK_SLASH, THICK_BACKSLASH };
92  }
93
```

Localização:

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

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

Classe com múltiplos métodos, definida dentro de uma interface -Autor José Romano, Review David Moreira

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:

```
4 implementations  dbarashev +2
25  public interface ColumnList {
      4 implementations  dbarashev
      dbarashev +2
60    class ColumnStub implements ColumnList.Column {
        7 usages
```

Localização:

ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.table.ColumnList.java

Linhas: 25 e 60, respetivamente.

Classe vazia -Autor Sofia Monteiro, Review José Romano

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:

```java
package biz.ganttproject.impex.msproject2;

dbarashev
public class WebStartIDClass {

}
```

Localização:
ganttproject.biz.ganttproject.impex.msproject2.src.main.java.biz.ganttproject.impex.msproject2.WebStartIDClass.java

Linhas: 1-5

Classe exaustivamente extensa – Autor David Moreira, Review Sofia Monteiro

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

Exemplo:

```java
44 usages  2 inheritors  dbarashev
public static class Polygon extends Shape {
    5 usages
```

```java
dbarashev
public static class Rectangle extends Polygon {
    2 usages
```

```java
11 usages  Dmitry Kazakov
public static class Rhombus extends Polygon {
    1 usage  Dmitry Kazakov
```

```java
13 usages  Dmitry Barashev +1
public static class Arrow extends Shape {
```

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

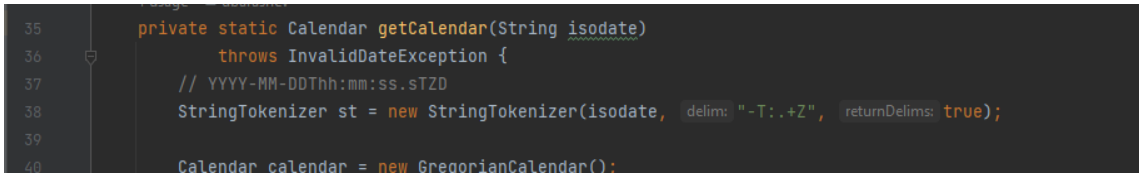
Linhas: 148, 230, 246, 257, respetivamente.

- Métodos

Método demasiado grande e complexo -Autor Joana Maroco, Review João Lopes

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:

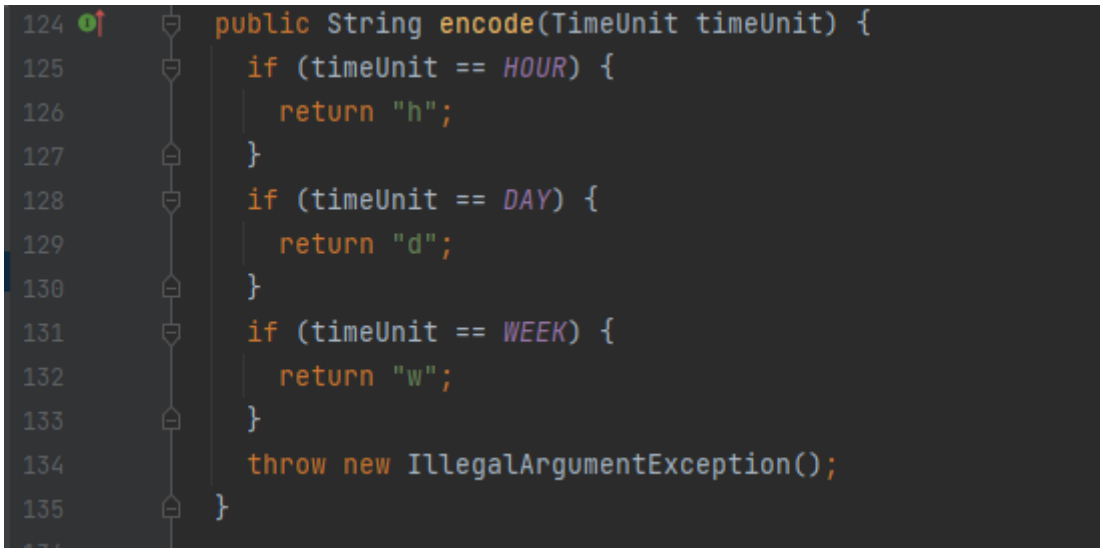


Localização:
ganttproject.biz.ganttproject.core.src.main.java.org.w3c.util.DateParser.java
Linhas: 35-157

Utilização de múltiplos if, em vez de um único switch -Autor Sofia Monteiro, Review João Lopes

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:

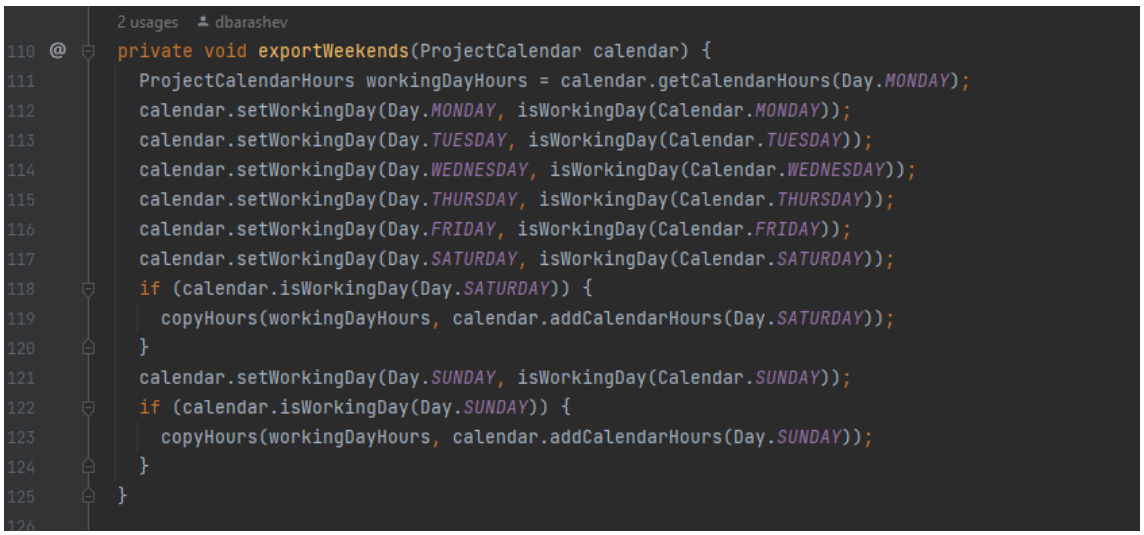


Localização:
ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.time.impl.GPTimeUnitStack.java
Linhas: 124-135

Duplicação de código- Autor José Romano, Review Sofia Monteiro

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:



```
2 usages  dbarashev
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 (Dead Code) – Autor João Lopes, Review David Moreira

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:

```
394    public void setMaxLength(int maxLength) {
395      myMaxLength = maxLength;
396    }
397
       dbarashev
398    public int getMaxLength() {
399      return myMaxLength;
400    }
```

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 – Autor João Lopes, Review Sofia Monteiro

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:

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

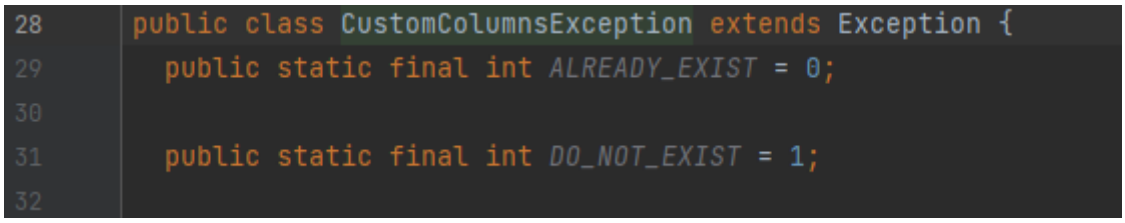
```
363        setPublicHolidays(calendar.getPublicHolidays());
364        for (int i = 1; i <= 7; i++) {
365          setWeekDayType(i, calendar.getWeekDayType(i));
```

Localização:
ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.calendar.WeekendCalendarImpl
Linha: 64 e 364, respetivamente"

Constantes não utilizadas -Autor Joana Maroco, Review José Romano

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:



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

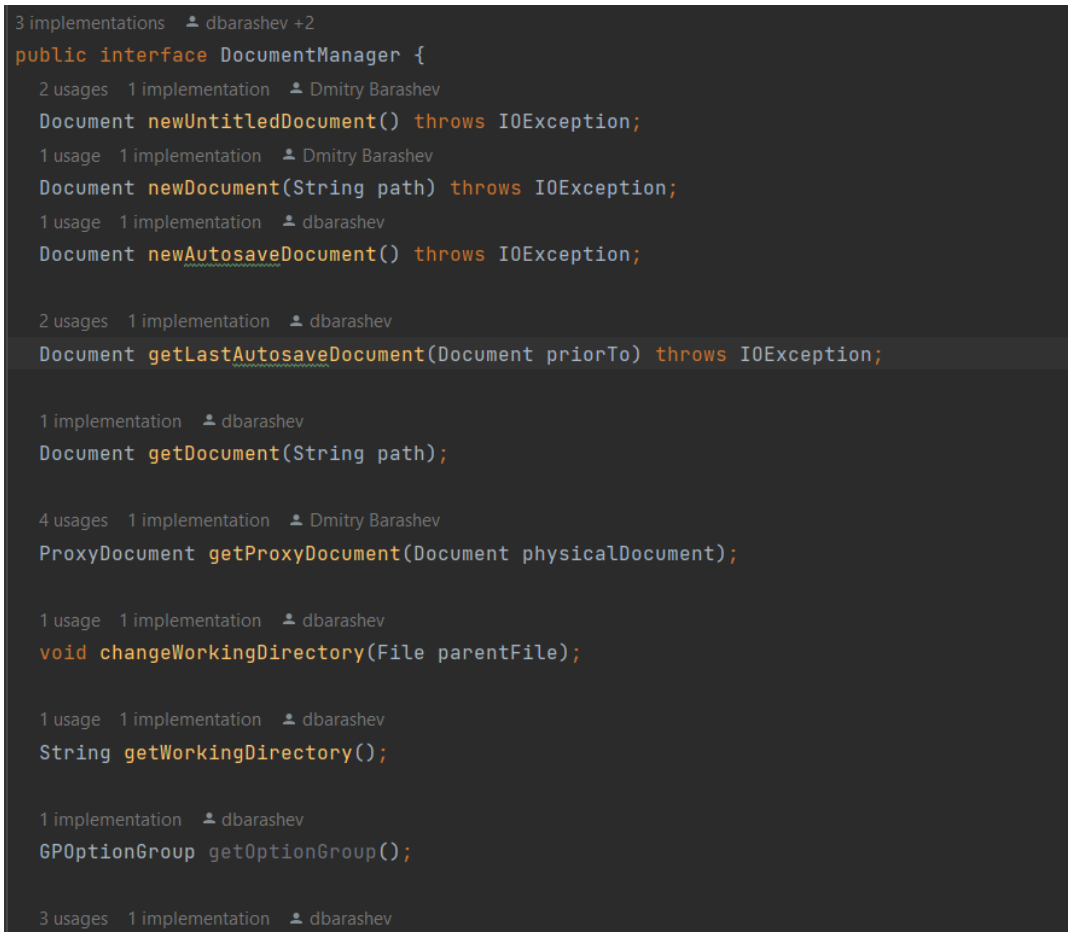
Linhas: 29-31

Capítulo 2 – GoF Design Patterns

Memento Pattern-Autor Joana Maroco, Review Sofia Monteiro

Esta interface e as respetivas implementações preservam o estado antigo, corrente e futuro do documento a editar. Isto é o foco do Memento Pattern, uma vez que ele permite que em qualquer momento seja possível ao utilizador (des)faça quaisquer operações realizadas no texto/documento a tratar.

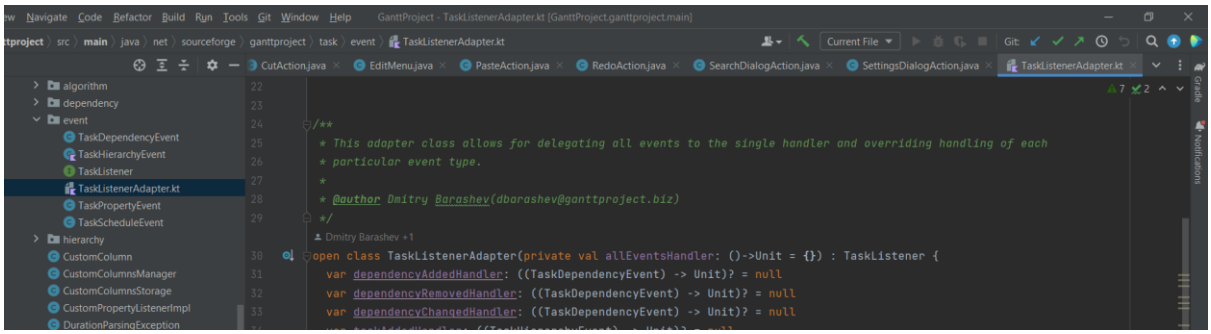
Pelas linhas 33-49 de ganttproject/src/main/java/net/sourceforge/ganttproject/document/DocumentManager.java temos, por exemplo:



The Adapter Pattern -Autor Joana Maroco, Review José Romano

Este Design Pattern foca-se em facilitar a comunicação entre dois sistemas/objetos, através de uma interface compatível para ambos. E é isto que observamos neste exemplo. A TaskListenerAdapter é uma interface de apoio ao manipulador de eventos consoante cada tipo de evento.

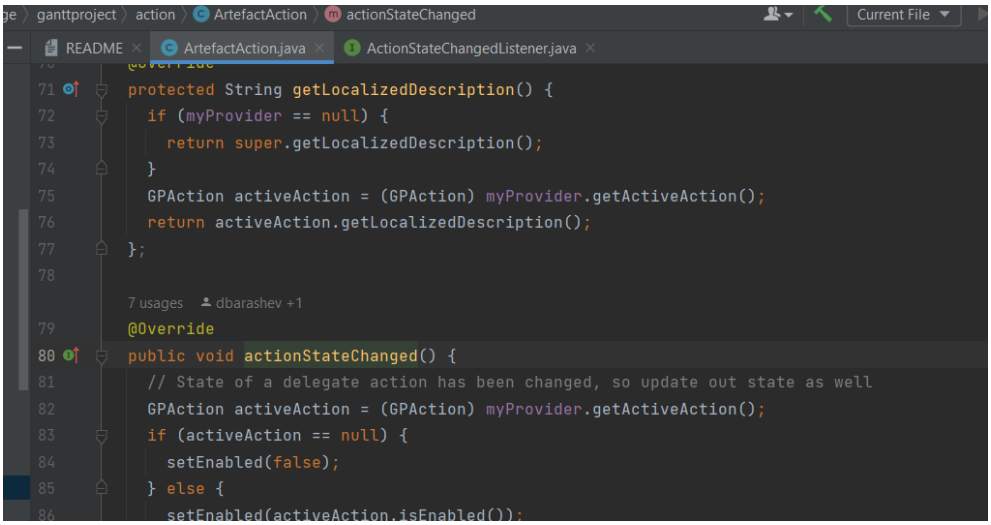
Pelas linhas 24 a 33 de ganttproject/src/main/java/net/sourceforge/ganttproject/task/event/TaskListenerAdapter.kt, temos, por exemplo:



Factory Pattern -Autor Joana Maroco, David Moreira

Como se pode ver neste exemplo, é definida a interface ActionStateChangedListener para criar todos os produtos distintos deste tipo, mas deixa a criação real do produto para a classe concreta do Produto.

Apresento as linhas 80-97 de ganttproject/src/main/java/net/sourceforge/ganttproject/action/ArtefactAction.java (do produto) sendo a localização da interface: ganttproject/src/main/java/net/sourceforge/ganttproject/action/ActionStateChangedListener.java



Abstract Factory Pattern -Autor David Moreira, Review João Lopes

No exemplo apresentado, é definida a interface CalendarFactory que controla todos os tipos de unidade que aparecem no calendário relativamente ao tempo e duração. No entanto, para cada um dos Produtos que o implementam, ainda são implementadas classes específicas, isto é, conseguimos observar uma "fábrica de fábricas", como descrito nas teóricas.

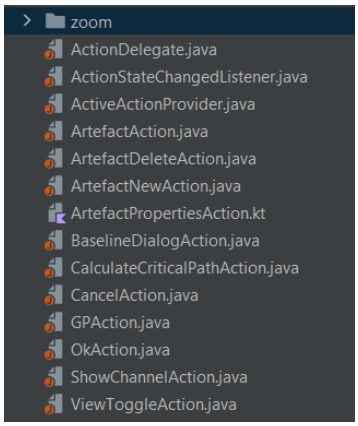
```java
public abstract class CalendarFactory {
  7 implementations  dbarashev
  public static interface LocaleApi {
    7 implementations  dbarashev
    Locale getLocale();
    7 implementations  dbarashev
    DateFormat getShortDateFormat();
  }

  5 usages
  private static LocaleApi ourLocaleApi;

  dbarashev
  public static Calendar newCalendar() { return (Calendar) Calendar.getInstance(ourLocaleApi.getLocale()).clone(); }

  dbarashev
  protected static void setLocaleApi(LocaleApi localeApi) { ourLocaleApi = localeApi; }

  dbarashev
  public static GanttCalendar createGanttCalendar(Date date) { return new GanttCalendar(date, ourLocaleApi); }

  dbarashev
  public static GanttCalendar createGanttCalendar(int year, int month, int date) {
    return new GanttCalendar(year, month, date, ourLocaleApi);
  }

  dbarashev
  public static GanttCalendar createGanttCalendar() { return new GanttCalendar(ourLocaleApi); }
}
```

Localização:
biz.ganttproject.core/src/main/java/biz/ganttproject/core/time/CalendarFactory.java

Factory Pattern- Autor José Romano, Review Sofia Monteiro

O Factory Pattern define-se como a criação de uma interface para a criação de objetos numa superclasse para assim permitir às subclasses alterarem o tipo de objetos que vão criar.

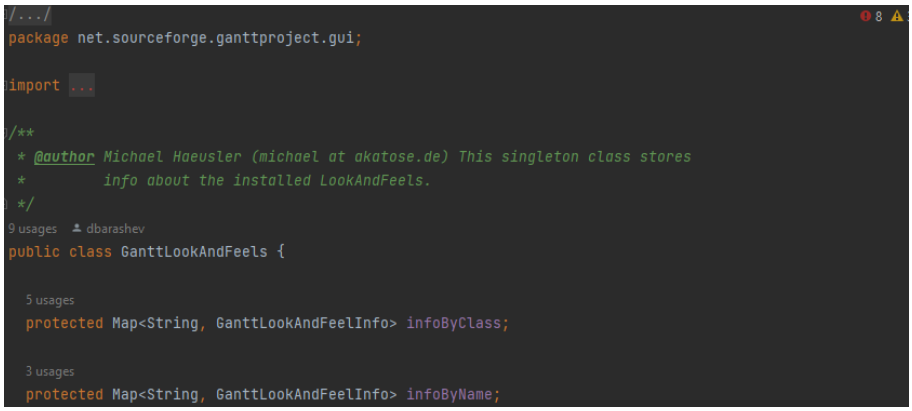
Neste caso, temos a interface ActionDelegate em que depois esta é implementada por vários objetos de tipos diferentes.



Localização: ganttproject/src/main/java/net/sourceforge/ganttproject/action

Singleton Pattern (Creational Pattern) -Autor Sofia Monteiro, Review João Lopes

Um padrão do estilo singleton tem como função principal central de forma mais restrita as variáveis globais: garante que existe apenas uma instância de classe, ou seja, existe apenas a classe singleton (classe GanttLookAndFeels). Tem também como característica ter um objeto partilhado por diferentes partes do programa.

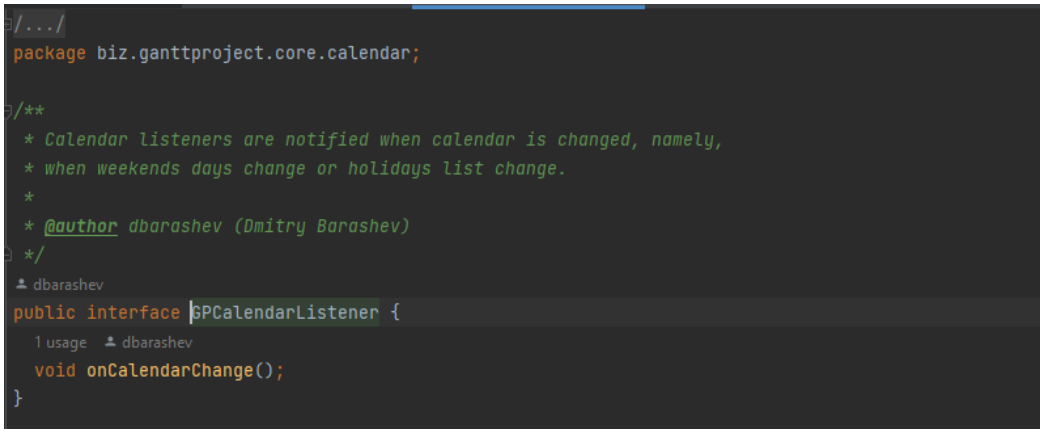


Localização:
ganttproject/src/main/java/net/sourceforge/ganttproject/gui/GanttLookAndFeels.java

Observer Pattern(Behavioral Pattern) -Autor Sofia Monteiro, Review Joana Maroco

Este padrão serve essencialmente para em vez de fazermos a chamada de um método diversas vezes para várias classes, por exemplo update(), fazemos apenas uma vez no observer/listener e este comunica com os restantes caso seja necessário.

Neste caso, temos a classe GPCalendarListener.java que é exatamente um listener de algum update que tenha de haver no calendário e, se houver, envia isso para as classes correspondentes.



Localização:
ganttproject/biz.ganttproject.core/src/main/java/biz/ganttproject/core/calendar/GPCalendarListener.java

Observer Pattern(Behavioral Pattern) – Autor David Moreira, Review Sofia Monteiro

Usamos este tipo de padrão quando se pretende atualizar algum parâmetro que irá alterar por consequência muitos outros parâmetros. Para tal usamos Interfaces Listeners que automaticamente atualizam todos os outros parâmetros a atualizar quando recebe o update do parâmetro original. Neste exemplo podemos observar o ScrollingListenner que recebe o update quando se dá scroll a algum número de dias.

State Pattern(Behavioural Pattern) -Autor Sofia Monteiro, Review José Romano

O padrão State tem como objetivo alterar o comportamento de um objeto quando o seu estado se altera. É usado de maneira a simplificar o código, limpando o número de excessivo de condições. No método parseDuration, pertencente à classe GPTimeUnitStack, verifica-se a obediência a este padrão, dado que a condição Character.isDigit(nextChar) muda de comportamento, dependendo do estado (state) em que se encontra.

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

Linhas: 161-254

Command Design Pattern -Autor David Moreira, Review Joana Maroco

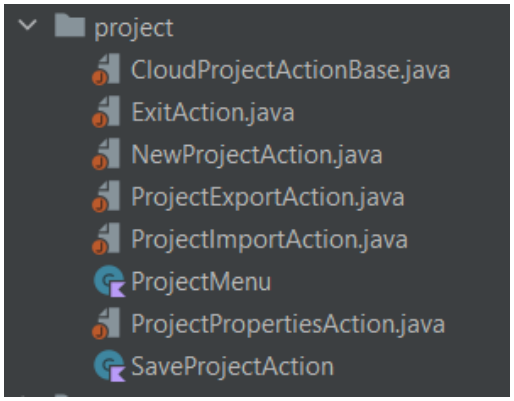
Como se pode observar neste exemplo, existem classes que representam ações específicas (no caso, "Copy", "Cut", "Edit", "Paste", "Redo", "RefreshView", "Search Dialog", "SettingsDialog" e "Undo"). Todas estas classes guardam linhas de código para serem executadas mais tarde as vezes que forem necessárias e vão ser enviadas pelo Invoker para o Receiver para este corre o comando ou a ação.

```java
public class CopyAction extends GPAction {
  3 usages
  private final GPViewManager myViewmanager;

  2 usages  dbarashev
  public CopyAction(GPViewManager viewManager) {
    super( name: "copy");
    myViewmanager = viewManager;
  }

  dbarashev +1
  @Override
  public void actionPerformed(ActionEvent e) {
    if (calledFromAppleScreenMenu(e)) {
      return;
    }
    myViewmanager.getSelectedArtefacts().startCopyClipboardTransaction();
  }
```

Localização: ganttproject/src/main/java/net/sourceforge/ganttproject/action/edit

Command Pattern -Autor José Romano, Review David Moreira

Observando o exemplo, podemos verificar que há classes que expressam ações concretas, tais como "ExitAction", "NewProjectAction", "ProjectExportAction", entre outras.

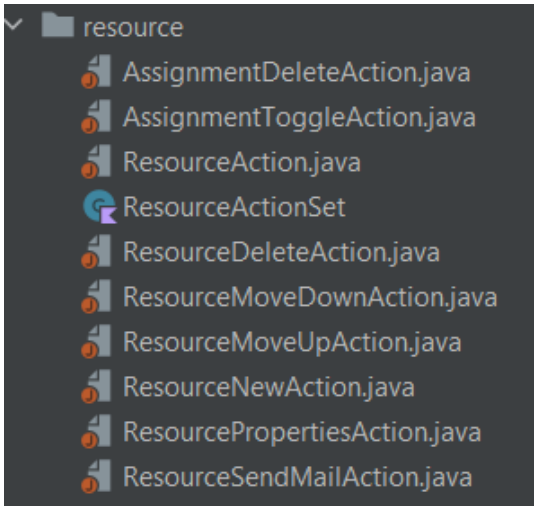


Localização: ganttproject/src/main/java/net/sourceforge/ganttproject/action/project

Command Pattern (Behavioural Pattern) -Autor João Lopes, Review David Moreira

O command Pattern é essencialmente existirem objetos que se caracterizem por comandos, mas que são definidos como classes. Neste caso temos, por exemplo, "AssignmentDeleteAction", "AssignmentToggleAction", "ResourceAction", "ResourceDeleteAction" entre outras classes.

Exemplo:



Localização:
ganttproject/src/main/java/net/sourceforge/ganttproject/action/resource

Iterator Pattern – Autor José Romano, Review João Lopes

É usado um iterador para correr todos os documentos existentes na nossa aplicação.

```
/** @return an Iterator over the entries of the list of Documents MRU */
dbarashev
public Iterator<String> iterator() { return documents.iterator(); }
```

Localização:
ganttproject/src/main/java/net/sourceforge/ganttproject/document/DocumentsMRU.java
Linhas: 95-97

Facade Pattern (Structural Pattern) – Autor João Lopes, Review José Romano

Este padrão simplifica a interação com um *framework* complexo de diversas Tasks/Tarefas. No caso, é uma interface para aceder a todas elas de forma hierárquica.

Exemplo:

```java
public interface TaskContainmentHierarchyFacade {
  2 implementations  dbarashev
  Task[] getNestedTasks(Task container);

  3 usages  2 implementations  dbarashev
  Task[] getDeepNestedTasks(Task container);

  2 implementations  dbarashev
  boolean hasNestedTasks(Task container);

  2 implementations  dbarashev
  Task getRootTask();

  2 implementations  dbarashev
  Task getContainer(Task nestedTask);

  2 implementations  Kambius
  void sort(Comparator<Task> comparator);
```

Localização:

ganttproject/src/main/java/net/sourceforge/ganttproject/task/TaskContainmentHierarchyFacade.java

Linhas apresentadas: 32-44

Composite Pattern (Structural Pattern) -Autor João Lopes, Review Joana Maroco

Usado para implementação de estruturas do tipo tree (classe TreeTableViewSkin). Tem como características a partilha de uma interface (TableViewSkinBase), algo que permite a criação de folhas (leaf) mais simples e retira a preocupação do cliente necessitar de ter em conta as classes dos objetos com os quais trabalha.

Exemplo:

```
* @see TableView
* @see TreeTableView
* TableViewSkin
* @see TreeTableViewSkin
*/
```

```
/**
 * Keeps track of how many leaf columns are currently visible in this table.
 */
2 usages  Dmitry Barashev
private void updateVisibleColumnCount() {
  visibleColCount = getVisibleLeafColumns().size();

  updatePlaceholderRegionVisibility();
  requestRebuildCells();
}
```

Localização:

ganttproject/ganttproject/src/main/java/biz/ganttproject/lib/fx/treetable/TableViewSkinBase.java

Github Repositório

Repositório do Projeto:
https://github.com/Joao1531/ganttproject

Repositório dos Smells e Patterns:
https://github.com/Joao1531/Phase1_Presentations_and_Reviews