
Relatório Fase 1 Miguel Agostinho (60677)

Patterns

Prototype Pattern (ProjectRolesOptionPageProvider.java)

```
@Override
protected Role createValue(Role prototype) {
    RoleSet projectRoles = getRoleManager().getProjectRoleSet();
    return projectRoles.createRole(prototype.getName());
}
```

Neste padrão estamos a criar um objeto copiando informações de um outro, para o podermos alterar sem danificar o original.

Iterator Pattern (TableRowSkinBase.java)

```
2 usages  Dmitry Barashev
private void recreateCells() {
    if (cellsMap != null) {
        Collection<Reference<R>> cells = cellsMap.values();
        Iterator<Reference<R>> cellsIter = cells.iterator();
        while (cellsIter.hasNext()) {
            Reference<R> cellRef = cellsIter.next();
            R cell = cellRef.get();
            if (cell != null) {
                cell.updateIndex(-1);
                cell.getSkin().dispose();
                cell.setSkin(null);
            }
        }
        cellsMap.clear();
    }

    ObservableList<? extends TableColumnBase*> columns = getVisibleLeafColumns();

    cellsMap = new WeakHashMap<>(columns.size());
    fullRefreshCounter = DEFAULT_FULL_REFRESH_COUNTER;
    getChildren().clear();

    for (TableColumnBase col : columns) {
        if (cellsMap.containsKey(col)) {
            continue;
        }

        // create a TableCell for this column and store it in the cellsMap
        // for future use
        createCellAndCache(col);
    }
}
```

Padrão identificado pela criação de um iterador para percorrer elementos “Reference”.

Template Method Pattern (TreeTableCells.kt)

```

Dmitry Barashev +2
override fun startEdit() {
    if (this.index == -1) {
        return
    }
    if (!isEditable) {
        onEditingCompleted()
        return
    }
    super.startEdit()
    contentDisplay = ContentDisplay.GRAPHIC_ONLY
    disclosureNode?.let {
        it.isVisible = false
    }
}

if (isEditing) {
    treeTableView.requestFocus()
    doStartEdit()
} else {
    onEditingCompleted()
}
}

```

Este padrão consiste em fazer “override” de um método da classe super chamando esse mesmo método e adicionando mais algumas funcionalidades.

Code Smells

Switch statement (VirtualFlow.java)

```

switch(event.getTextDeltaXUnits()) {
    case CHARACTERS:
        // can we get character size here?
        // for now, fall through to pixel values
    case NONE:
        double dx = event.getDeltaX();
        double dy = event.getDeltaY();

        virtualDelta = (Math.abs(dx) > Math.abs(dy) ? dx : dy);
}
}

```

O switch apresentado abaixo para além de no case “CHARACTERS” não se fazer nada podendo ser um “Dead Code”, poderia ser feito apenas com um if/else evitando assim o uso desnecessário de um switch statement.

Duplicated Code (VirtualFlow.java)

```
ScrollBar nonVirtualBar = isVertical() ? hbar : vbar;
if (needBreadthBar) {
    double nonVirtualDelta = isVertical() ? event.getDeltaX() : event.getDeltaY();
    if (nonVirtualDelta != 0.0) {
        double newValue = nonVirtualBar.getValue() - nonVirtualDelta;
        if (newValue < nonVirtualBar.getMin()) {
            nonVirtualBar.setValue(nonVirtualBar.getMin());
        } else if (newValue > nonVirtualBar.getMax()) {
            nonVirtualBar.setValue(nonVirtualBar.getMax());
        } else {
            nonVirtualBar.setValue(newValue);
        }
        event.consume();
    }
}
});
```

Neste caso podemos observar que chamamos duas vezes o método `nonVirtualBar.getMin()` e `nonVirtualBar.getMax()` o que deveríamos fazer era criar duas variáveis locais, uma que guardava o valor retornado por `nonVirtualBar.getMin()` e a outra que guarda o valor de `nonVirtualBar.getMax()` e assim evitamos chamar o mesmo método várias vezes.

Comments (ChartUIConfiguration.java)

```
public class ChartUIConfiguration {
```

```
1 dbarashev - 1
public ChartUIConfiguration(UIConfiguration projectConfig) {
    mySpanningRowTextFont = Fonts.TOP_UNIT_FONT;
    mySpanningHeaderBackgroundColor = new Color(0.93f, 0.93f, 0.93f);
    myHeaderBorderColor = new Color(0.482f, 0.482f, 0.482f);
    myWorkingLineBackgroundColor = Color.WHITE;
    myHolidayLineBackgroundColor = new Color(0.9f, 0.9f, 0.9f);
    myPublicHolidayTimeBackgroundColor = new Color(240, 220, 240);
    // myHeaderBorderColor = new Color(0f, 0f, 0f);
    myBottomUnitBorderColor = new Color(0.482f, 0.482f, 0.482f);
    myProjectConfig = projectConfig;
    myChartStylesOption = new ChartPropertiesOption();
}

2 dbarashev
ListOption<Map.Entry<String, String>> getChartStylesOption() {return myChartStylesOption;}

3 dbarashev
Font getSpanningHeaderFont() {return mySpanningRowTextFont;}

4 usage 2 dbarashev
public int getHeaderHeight() {return myHeaderHeight;}

5 usage 2 dbarashev
public void setHeaderHeight(int headerHeight) {myHeaderHeight = headerHeight;}

6 dbarashev
public int getSpanningHeaderHeight() {return myHeaderHeight / 2;}

7 dbarashev
public Color getSpanningHeaderBackgroundColor() {return mySpanningHeaderBackgroundColor;}

8 dbarashev
public Color getHeaderBorderColor() {return myHeaderBorderColor;}
}
```

Neste caso a classe `ChartUIConfiguration` tem uma ausência de comentários muito notória e os comentários nas classes são muito importantes porque se outra pessoa for programar a mesma classe precisa de saber o que os métodos ou até a classe fazem, portanto deveríamos comentar esta classe.

Relatório Fase 1 Daniel Eugénio (59797)

Padrões

- Iterador

```
112     private DefaultMutableTreeTableNode buildTree() {
113
114         DefaultMutableTreeTableNode root = new DefaultMutableTreeTableNode();
115         List<HumanResource> listResources = myResourceManager.getResources();
116         Iterator<HumanResource> itRes = listResources.iterator();
117
118         while (itRes.hasNext()) {
119             HumanResource hr = itRes.next();
120             ResourceNode rnRes = new ResourceNode(hr); // the first for the resource
121             root.add(rnRes);
122         }
123         return root;
124     }
```

- Este padrão é facilmente identificado pela criação de um objeto do tipo Iterator. É acompanhado com um ciclo while para percorrer os elementos da lista

-----> (ResourceTreeTableModel.java) <-----

- Prototype

```
141     public TaskBuilder withPrototype(Task prototype) {
142         myPrototype = prototype;
143         return this;
144     }
```

- Este pedaço de código permite criar uma cópia dum objeto Task que nele seja passado. O método em questão é usado na classe ClipboardTaskProcessor, para criar uma cópia de uma task.

-----> (TaskManager.java) <-----

- Façade

```
102 class UIFacadeImpl extends ProgressProvider implements UIFacade {
103     private final JFrame myMainFrame;
104     private final ScrollingManager myScrollingManager;
105     private final ZoomManager myZoomManager;
106     private final GanttStatusBar myStatusBar;
107     private final UIFacade myFallbackDelegate;
108     private final TaskSelectionManager myTaskSelectionManager;
109     private final List<GPOptionGroup> myOptionGroups = Lists.newArrayList();
110     private final GPOptionGroup myOptions;
111     private final LafOption myLafOption;
112     private final GPOptionGroup myLogoOptions;
113     private final DefaultFileOption myLogoOption;
114     private final NotificationManagerImpl myNotificationManager;
115     private final TaskView myTaskView = new TaskView();
116     private final DialogBuilder myDialogBuilder;
117     private final Map<String, Font> myOriginalFonts = Maps.newHashMap();
118     private final List<Runnable> myOnUpdateComponentTreeUiCallbacks = Lists.newArrayList();
119     private float myLastScale = 0;
120
121     private static Map<FontSpec.Size, String> getSizeLabels() {
122         Map<FontSpec.Size, String> result = Maps.newHashMap();
123         for (FontSpec.Size size : FontSpec.Size.values()) {
124             result.put(size, GanttLanguage.getInstance().getText("optionValue.ui.appFontSpec." + size.toString() + ".label"));
125         }
126         return result;
127     }
128 }
```

- Aqui está implementado o padrão Façade. Neste caso, o padrão é aplicado para simplificar o API ProgressProvider.

-----> (UIFacadeImpl.java) <-----

Code Smells

- Repeated Code

```
234 |getFragmentManager().createView(myGanttChartTabContent, new ImageIcon(getClass().getResource("/icons/tasks_16.gif")));
235 |getFragmentManager().toggleVisible(myGanttChartTabContent);
236 |
237 |myResourceChartTabContent = new ResourceChartTabContentPanel(getProject(), getUIFacade(), getResourcePanel(),
238 |    getResourcePanel().area);
239 |getFragmentManager().createView(myResourceChartTabContent, new ImageIcon(getClass().getResource("/icons/res_16.gif")));
240 |getFragmentManager().toggleVisible(myResourceChartTabContent);
```

- Neste trecho, é possível reparar que as linhas 234-235 e 239-240 são muito semelhantes, diferindo apenas no último argumento passado no método toggleVisible().
- Uma solução possível seria fazer um método que receberia o argumento que queremos passar no toggleVisible(). Esta simples mudança tornaria o código nesta região mais limpo e legível.

-----> (GanttProject.java) <-----

- Long Parameter List

```
147 |private void constructTopOffsets(TimeUnit timeUnit, List<Offset> topOffsets, List<Offset> bottomOffsets,
148 |    int initialEnd, int baseUnitWidth) {
```

- Neste exemplo, os argumentos topOffsets e bottomOffsets podem ser atributos numa classe (Offsets.java, por exemplo) e, nesse cenário, estes dois argumentos podiam ser 1 apenas.

-----> (OffsetBuilderImpl.java) <-----

- Divergent Change

```
85
86 Box colorLabels = Box.createHorizontalBox();
87 for (final Color c : myRecentColors) {
88     final JLabel label = new JLabel();
89     label.setBackgroundPainter(new Painter<JLabel>() {
90         @Override
91         public void paint(Graphics2D g, JLabel object, int width, int height) {
92             g.setColor(c);
93             g.fillRect(4, 4, width-8, height-8);
94         }
95     });
96     label.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
97     label.setFocusable(true);
98     label.setPreferredSize(new Dimension(20, 20));
99     label.setMaximumSize(new Dimension(20, 20));
100
101     final Border outsideFocusBorder = BorderFactory.createLineBorder(c.darker(), 2);
102     final Border outsideNoFocusBorder = BorderFactory.createEmptyBorder(2,2,2,2);
103     label.setBorder(outsideNoFocusBorder);
104     label.addFocusListener(new FocusAdapter() {
105         @Override
106         public void focusGained(FocusEvent e) {
107             label.setBorder(outsideFocusBorder);
108             myChooserImpl.setColor(c);
109             mySelectedColor = c;
110         }
111         @Override
112         public void focusLost(FocusEvent e) { label.setBorder(outsideNoFocusBorder); }
113     });
114     label.addMouseListener(new MouseAdapter() {
115         @Override
116         public void mouseClicked(MouseEvent e) { label.requestFocus(); }
117     });
118     colorLabels.add(label);
119     colorLabels.add(Box.createHorizontalStrut(5));
120 }
121 colorLabels.setBorder(BorderFactory.createEmptyBorder(0, 7, 0, 0));
```

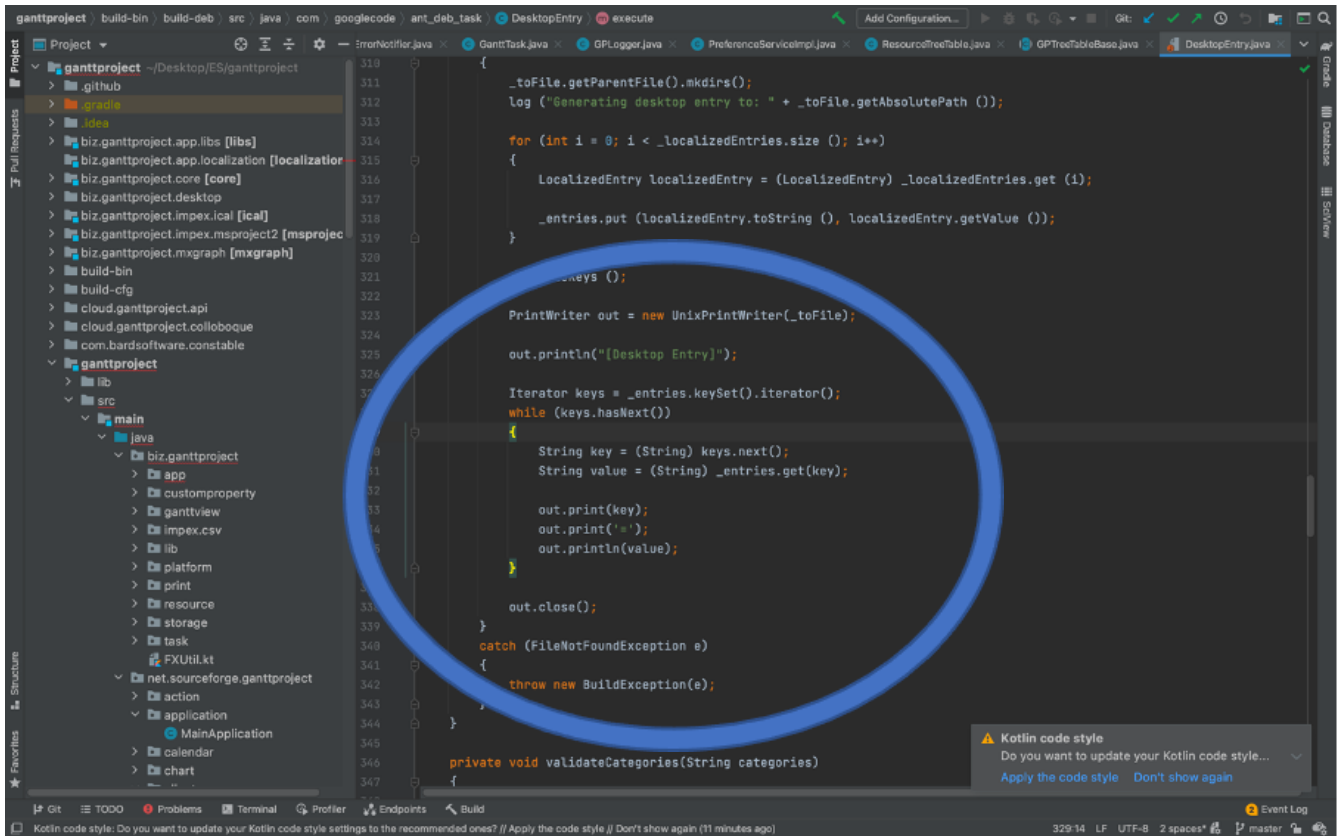
- Aqui é possível ver a completa ausência do uso de constantes, o que nos leva a ter de mexer em várias zonas do código na mesma classe. A utilização dos chamados *magic numbers* também dificulta a legibilidade do código por parte de pessoas que não estiveram envolvidas inicialmente no seu desenvolvimento.
- A solução seria utilizar constantes com nomes explicativos para podermos fatorizar várias zonas do código ao mesmo tempo.

-----> (GPColorChooser.java) <-----

Relatório Fase 1 Rafael Costa (60441)

Patterns (Padrão)

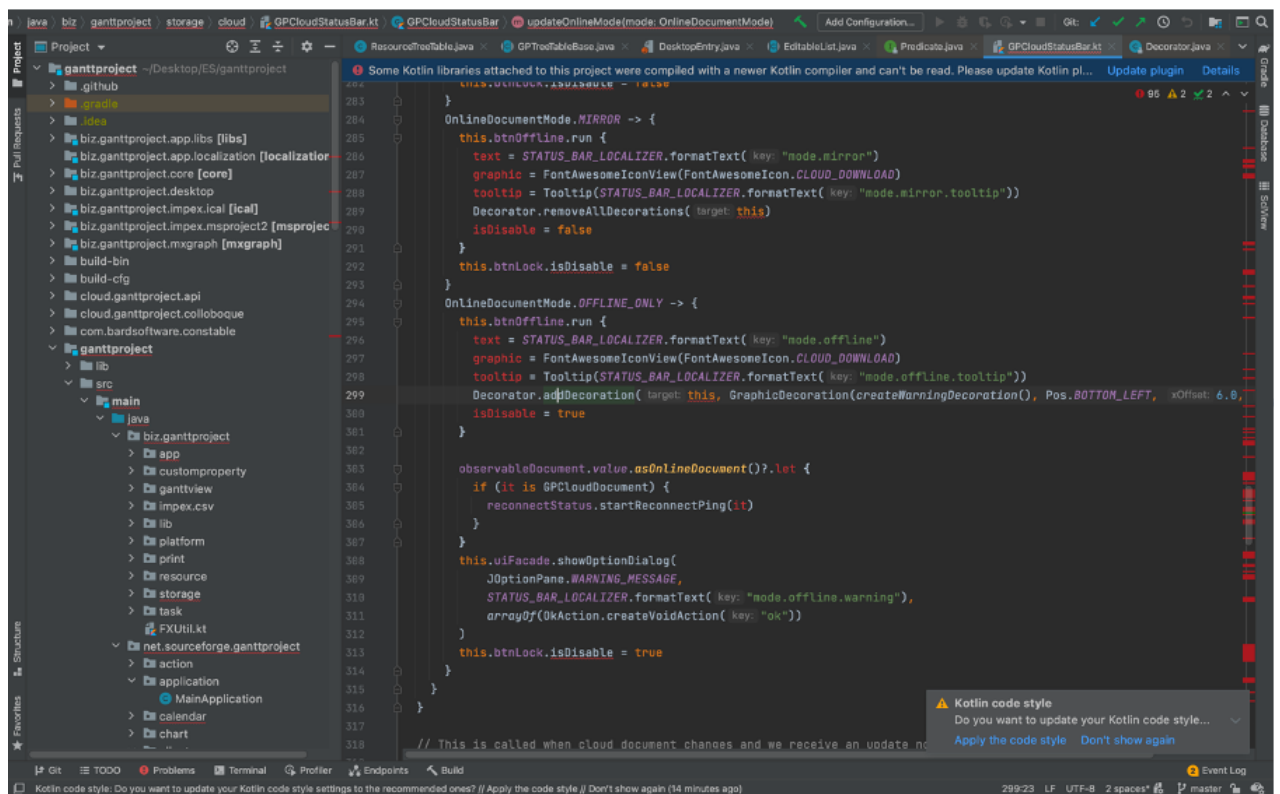
1. Iterator patterns



Este padrão é facilmente identificado pela criação de um Iterador, que vai iterar/percorrer um objeto

<DesktopEntry.java>

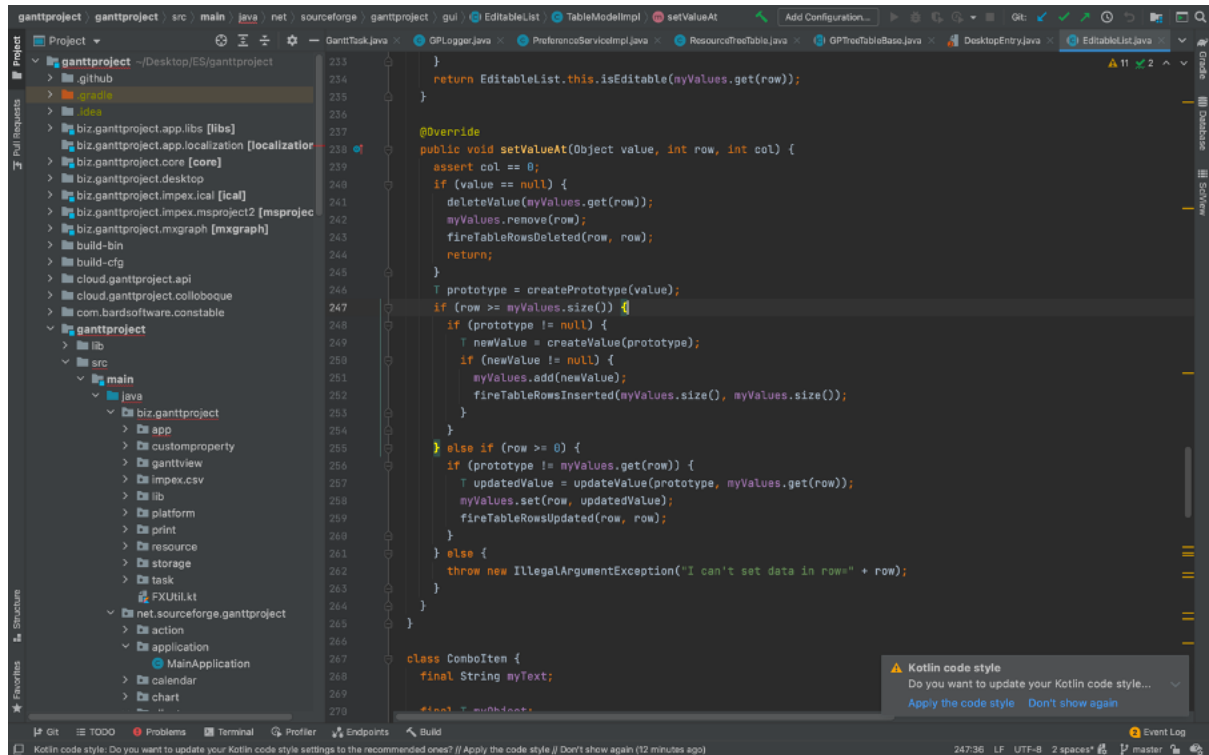
2. Decorator pattern



Este padrão permite anexar novos comportamentos, funcionalidades ou estados extra a um objeto em tempo de execução colocando esses objetos dentro de objetos especiais que contêm os comportamentos.

<GPCloudStatusBar.kt>

3. Prototype Pattern

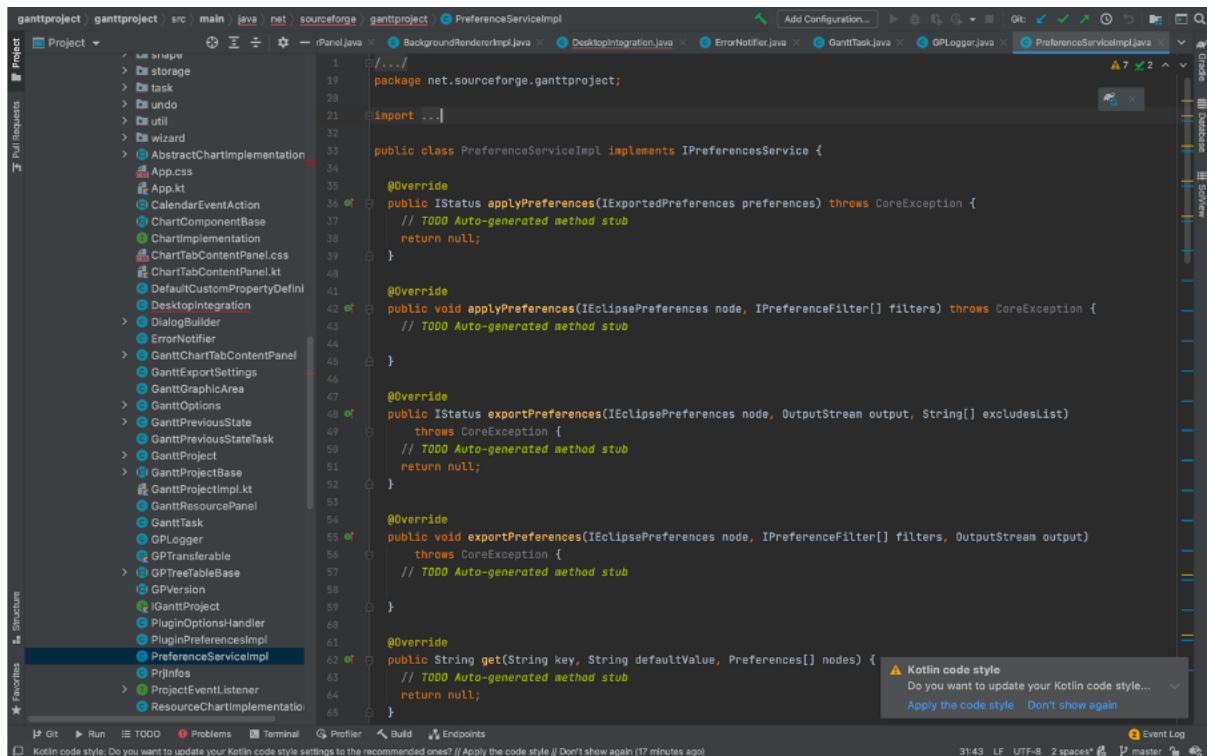


Através do prototype podemos criar uma cópia dum objeto que nele seja passado.

<EditableList.java>

Code Smell

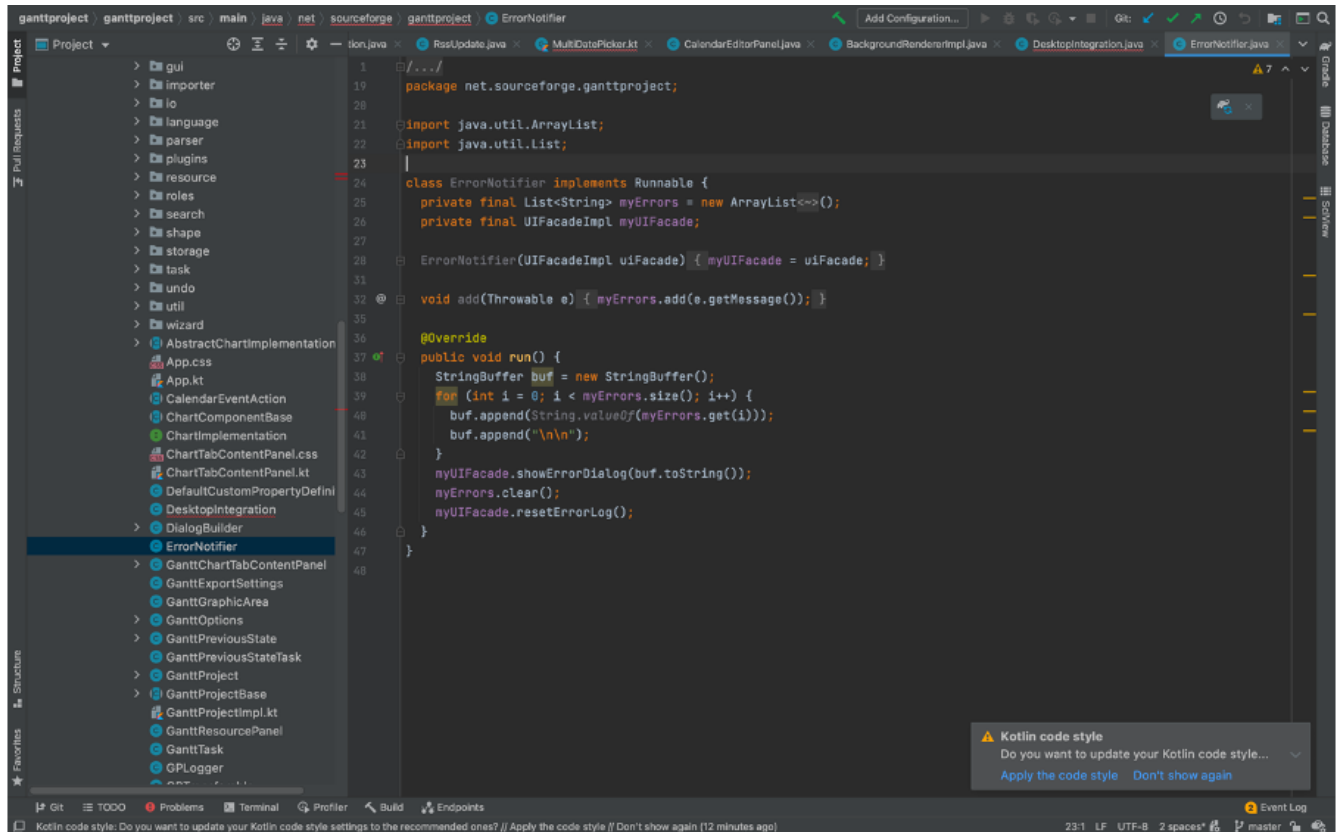
1. Speculative Generality



É constituído por código que é genérico ou abstrato e o mais importante, não é realmente necessário hoje. Esse código está lá para apoiar o comportamento futuro, que pode ou não, ser necessário no futuro. Solução será apenas criar os métodos e parâmetros à medida que é necessário.

<PreferenceServiceImpl.java>

2. Not Comment

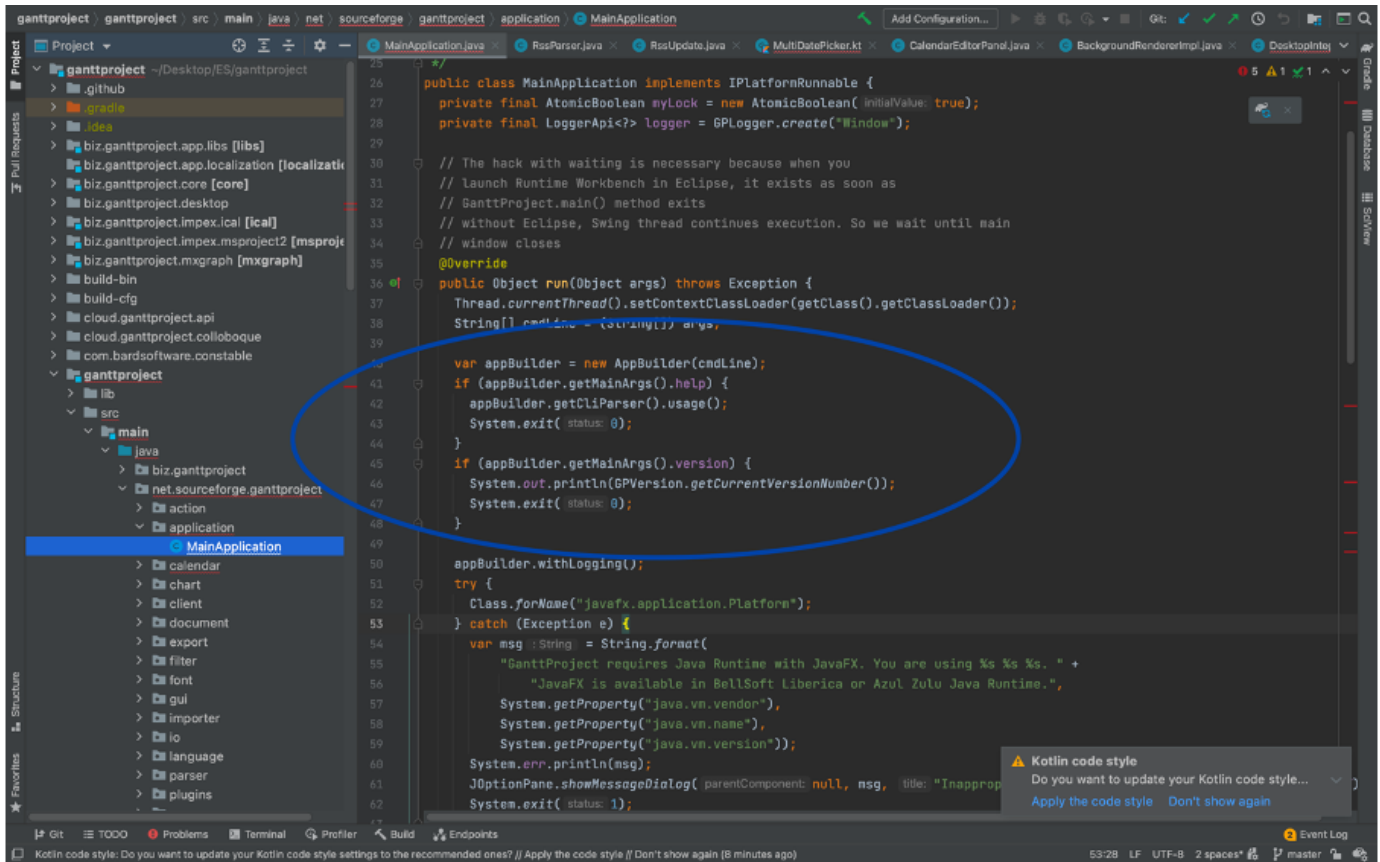


A classe apresenta-se sem comentários, o que é algo essencial para a interpretação da mesma.

Os comentários são essenciais uma vez que ao passar o código a outra pessoa, ou a trabalhar em equipa vai facilitar a interpretação do código pelos mesmos

<ErrorNotifier.java>

3. Duplicated Code



Dentro de cada "if" é chamada uma função para chegar a um valor da mesma. Neste caso, para evitarmos estarmos sempre a chamar a mesma função, podemos fazer a chamada antes do "if" e guardar `appBuilder.getMainArgs()` em uma variável, com isso não necessitamos de estar a chamar 2 vezes o `getMainArgs()` da classe `AppBuilder`

<MainApplication.java>

Relatório Fase 1 Guilherme Abrantes (60971)

Patterns (Padrão)

1.Singleton Pattern

```
70
71 private static GPCalendarProvider ourInstance;
72
73 static GPCalendar readCalendar(File resource) {
74     WeekendCalendarImpl calendar = new WeekendCalendarImpl();
75
76     HolidayTagHandler holidayHandler = new HolidayTagHandler(calendar);
77     CalendarTagHandler calendarHandler = new CalendarTagHandler(calendar, holidayHandler);
78     XmlParser parser = new XmlParser(
79         ImmutableList.<TagHandler>of(calendarHandler, holidayHandler),
80         ImmutableList.<ParsingListener>of());
81     try {
82         parser.parse(new BufferedInputStream(new FileInputStream(resource)));
83         return calendar;
84     } catch (IOException e) {
85         GPLogger.logToLogger("Failed to parse file "+resource.getAbsolutePath());
86         GPLogger.logToLogger(e);
87         return null;
88     }
89 }
90
91 private static List<GPCalendar> readCalendars() {
92     return HolidayCalendarKt.loadCalendars();
93 }
94
95 public static synchronized GPCalendarProvider getInstance() {
96     if (ourInstance == null) {
97         List<GPCalendar> calendars = readCalendars();
98         Collections.sort(calendars, new Comparator<GPCalendar>() {
99             public int compare(GPCalendar o1, GPCalendar o2) {
100                 return o1.getName().compareTo(o2.getName());
101             }
102         });
103         ourInstance = new GPCalendarProvider(calendars);
104     }
105     return ourInstance;
106 }
```

Este padrão é identificável pela criação de uma instância única `ourInstance` e pelo método `public static synchronized` que fornece acesso global a esta instância.

-----> (GPCalenderProvider.java) <-----

2.Iterator Pattern

```
66● private Dimension calculateDimension() {
67     int width = 0;
68     int assignmentsCount = 0;
69     final BufferedImage testImage = new BufferedImage(10, 10, BufferedImage.TYPE_INT_RGB);
70     final Graphics2D g = (Graphics2D) testImage.getGraphics();
71     final int tabSize = 5;
72     final List<HumanResource> users = myResourceManager.getResources();
73     for (Iterator<HumanResource> user = users.iterator(); user.hasNext();) {
74         HumanResource hr = user.next();
75         int nameWidth = TextLengthCalculatorImpl.getTextLength(g, hr.getName());
76         if (nameWidth > width) {
77             width = nameWidth;
78         }
79         ResourceAssignment[] assignments = hr.getAssignments();
80         if (assignments != null) {
81             for (int i = 0; i < assignments.length; i++) {
82                 if (isAssignmentVisible(assignments[i])) {
83                     int taskWidth = tabSize + TextLengthCalculatorImpl.getTextLength(g, assignments[i].getTask().getName());
84                     if (taskWidth > width) {
85                         width = taskWidth;
86                     }
87                     assignmentsCount++;
88                 }
89             }
90         }
91     }
92     width += 20;
93     int height = (assignmentsCount + users.size()) * getRowHeight() + 90;
94     return new Dimension(width, height);
95 }
96
```

Este padrão é facilmente identificado pela criação de um objeto do tipo Iterator. É acompanhado com um ciclo for para percorrer os elementos da lista.

-----> (ResourceTreeImageGenerator.java) <-----

3.Memento Pattern

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure: ganttproject > JRE System Library [jdk-14] > gui > action > edit. The main editor shows the code for RedoAction.java. The code implements the Memento Pattern for redoing actions. It includes a constructor, a private constructor, and three overridden methods: actionPerformed, undoableEditHappened, and undoReset. The actionPerformed method calls myUndoManager.redo() when the action is performed. The undoableEditHappened method updates the redo state and tooltip. The undoReset method resets the redo state and tooltip.

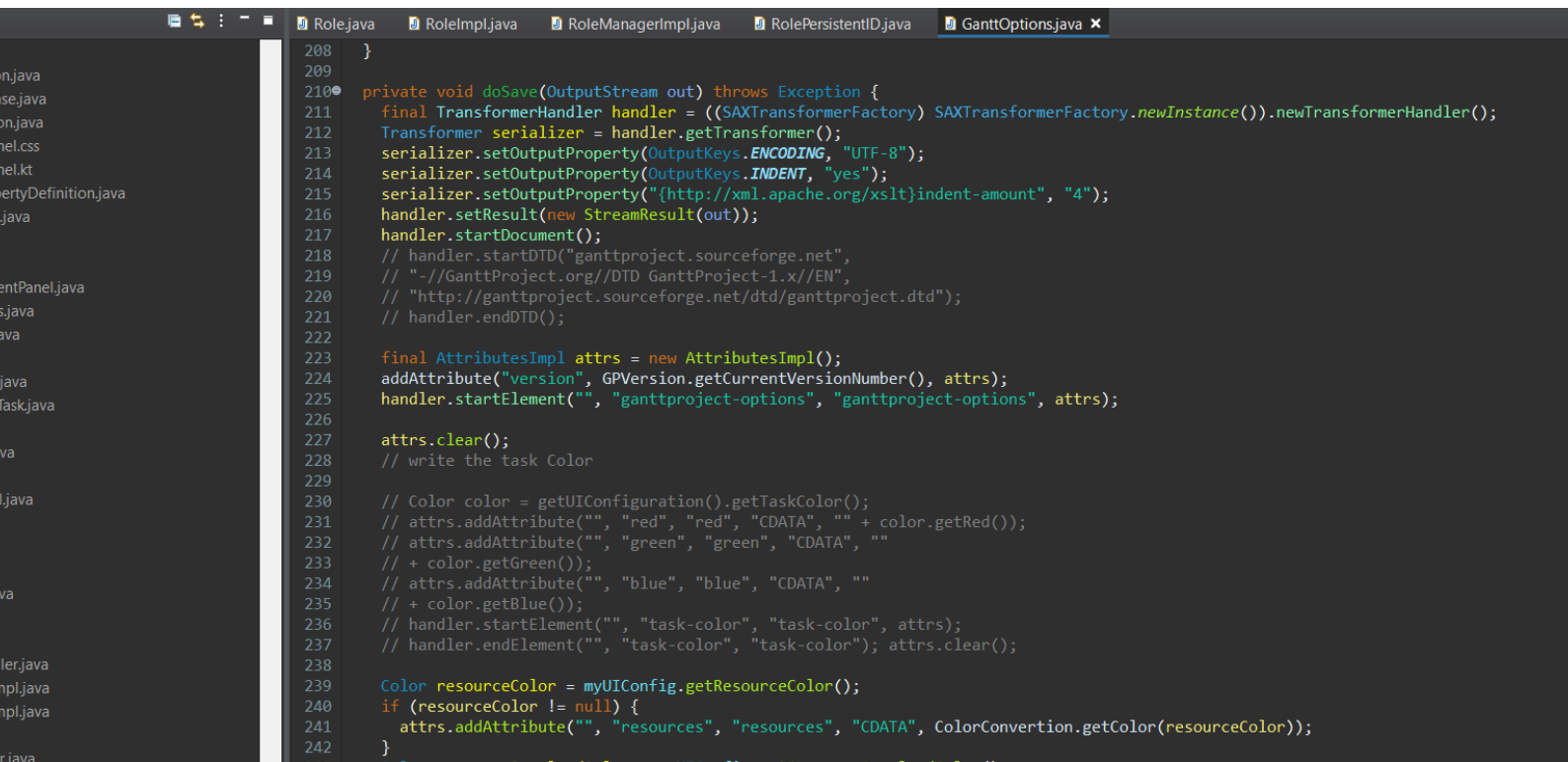
```
30 * @author bard
31 */
32 public class RedoAction extends GPAction implements GPUndoListener {
33     private final GPUndoManager myUndoManager;
34
35     public RedoAction(GPUndoManager undoManager) {
36         this(undoManager, IconSize.MENU);
37     }
38
39     private RedoAction(GPUndoManager undoManager, IconSize size) {
40         super("redo", size);
41         myUndoManager = undoManager;
42         myUndoManager.addUndoableEditListener(this);
43         setEnabled(myUndoManager.canRedo());
44     }
45
46     @Override
47     public void actionPerformed(ActionEvent e) {
48         if (calledFromAppleScreenMenu(e)) {
49             return;
50         }
51
52         myUndoManager.redo();
53     }
54
55     @Override
56     public void undoableEditHappened(UndoableEditEvent e) {
57         setEnabled(myUndoManager.canRedo());
58         updateTooltip();
59     }
60
61     @Override
62     public void undoOrRedoHappened() {
63         setEnabled(myUndoManager.canRedo());
64         updateTooltip();
65     }
66
67     @Override
68     public void undoReset() {
69         undoOrRedoHappened();
70     }
71
72     @Override
```

Aqui podemos observar uma classe que permite que a ação corrente volte atrás para a ação antiga a partir do método `undoReset()`.

-----> (RedoAction.java) <-----

Code Smell

1. comments that take on a “reminder” nature



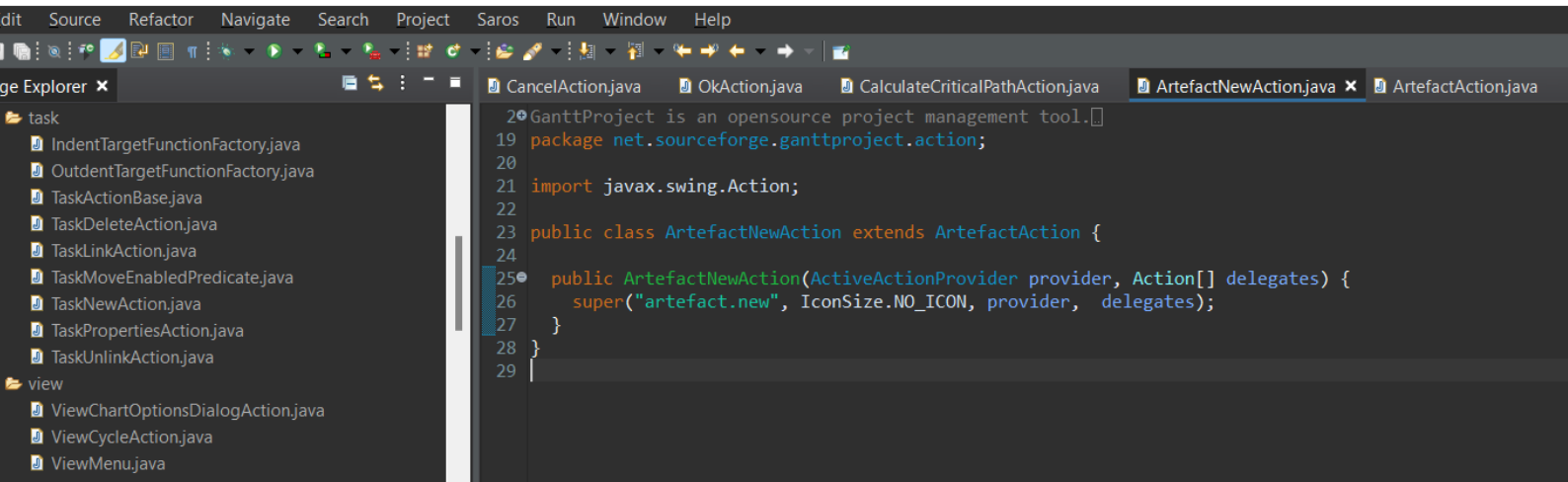
```
208 }
209
210 private void doSave(OutputStream out) throws Exception {
211     final TransformerHandler handler = ((SAXTransformerFactory) SAXTransformerFactory.newInstance()).newTransformerHandler();
212     Transformer serializer = handler.getTransformer();
213     serializer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
214     serializer.setOutputProperty(OutputKeys.INDENT, "yes");
215     serializer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");
216     handler.setResult(new StreamResult(out));
217     handler.startDocument();
218     // handler.startDTD("ganttproject.sourceforge.net",
219     // "-//GanttProject.org//DTD GanttProject-1.x//EN",
220     // "http://ganttproject.sourceforge.net/dtd/ganttproject.dtd");
221     // handler.endDTD();
222
223     final AttributesImpl attrs = new AttributesImpl();
224     addAttribute("version", GPVersion.getCurrentVersionNumber(), attrs);
225     handler.startElement("", "ganttproject-options", "ganttproject-options", attrs);
226
227     attrs.clear();
228     // write the task Color
229
230     // Color color = getUIConfiguration().getTaskColor();
231     // attrs.addAttribute("", "red", "red", "CDATA", "" + color.getRed());
232     // attrs.addAttribute("", "green", "green", "CDATA", ""
233     // + color.getGreen());
234     // attrs.addAttribute("", "blue", "blue", "CDATA", ""
235     // + color.getBlue());
236     // handler.startElement("", "task-color", "task-color", attrs);
237     // handler.endElement("", "task-color", "task-color"); attrs.clear();
238
239     Color resourceColor = myUIConfig.getResourceColor();
240     if (resourceColor != null) {
241         attrs.addAttribute("", "resources", "resources", "CDATA", ColorConversion.getColor(resourceColor));
242     }
243 }
```

Aqui vemos um comentário que evidencia que algo precisa de ser feito no futuro o que aplica alterações em outros métodos para resolver devíamos ter implementado logo esta funcionalidade para não resultar em mais problemas.

-----> (GanttOptions.java) <-----

2.Data Class

ens - ganttproject/action/ArtefactNewAction.java - Eclipse IDE

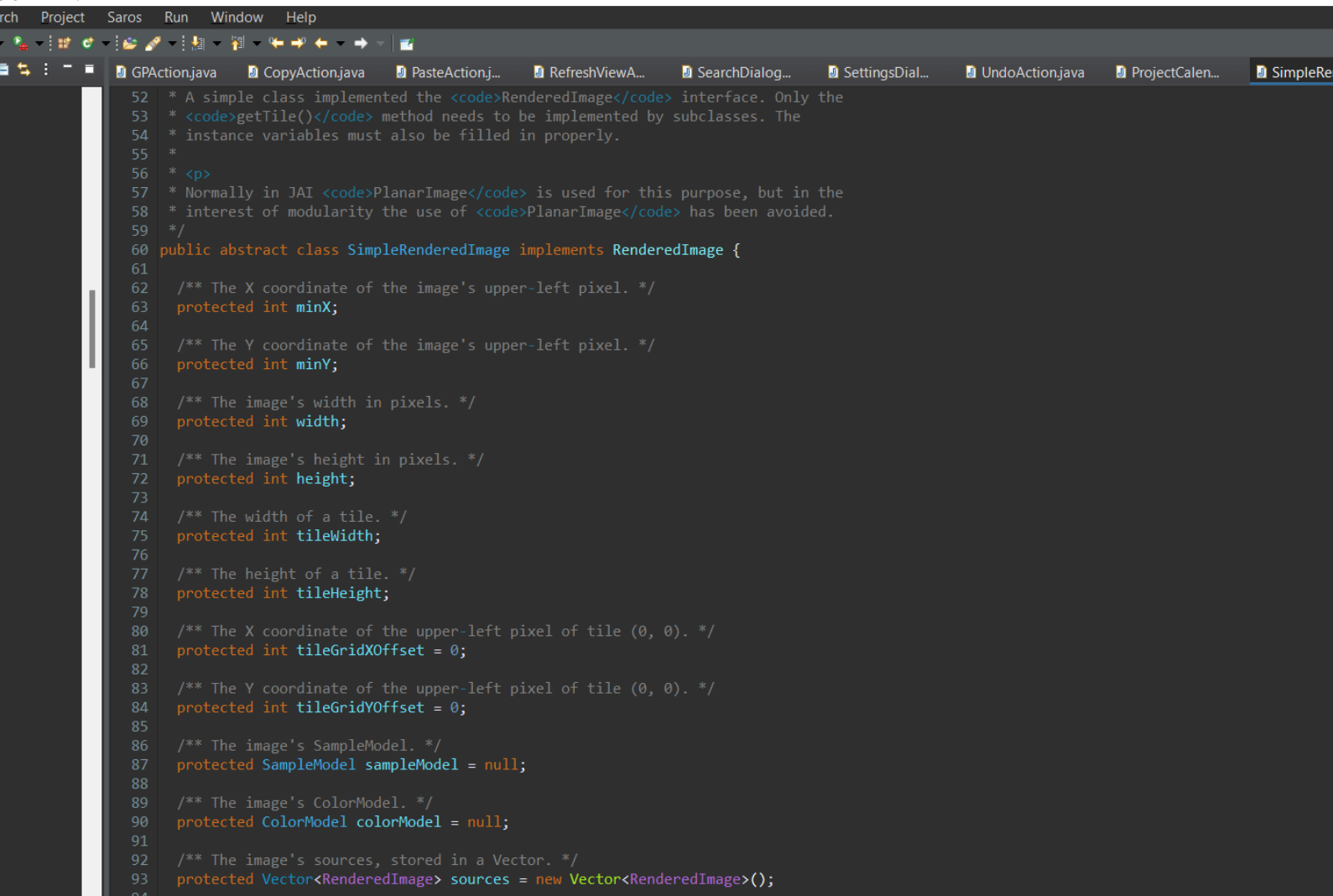


```
20 GanttProject is an opensource project management tool.
19 package net.sourceforge.ganttproject.action;
20
21 import javax.swing.Action;
22
23 public class ArtefactNewAction extends ArtefactAction {
24
25     public ArtefactNewAction(ActiveActionProvider provider, Action[] delegates) {
26         super("artefact.new", IconSize.NO_ICON, provider, delegates);
27     }
28 }
29
```

Aqui vemos o Code smell Dataclass sendo esta classe bastante desnecessária pois apenas tem o construtor, esta classe seria facilmente substituída por um simples método na classe ArtefactAction

-----> (ArtefactNewAction.java) <-----

3.Data Clumps



```
52 * A simple class implemented the RenderedImage interface. Only the
53 * getTile() method needs to be implemented by subclasses. The
54 * instance variables must also be filled in properly.
55 *
56 * <p>
57 * Normally in JAI PlanarImage is used for this purpose, but in the
58 * interest of modularity the use of PlanarImage has been avoided.
59 */
60 public abstract class SimpleRenderedImage implements RenderedImage {
61
62     /** The X coordinate of the image's upper-left pixel. */
63     protected int minX;
64
65     /** The Y coordinate of the image's upper-left pixel. */
66     protected int minY;
67
68     /** The image's width in pixels. */
69     protected int width;
70
71     /** The image's height in pixels. */
72     protected int height;
73
74     /** The width of a tile. */
75     protected int tileWidth;
76
77     /** The height of a tile. */
78     protected int tileHeight;
79
80     /** The X coordinate of the upper-left pixel of tile (0, 0). */
81     protected int tileGridXOffset = 0;
82
83     /** The Y coordinate of the upper-left pixel of tile (0, 0). */
84     protected int tileGridYOffset = 0;
85
86     /** The image's SampleModel. */
87     protected SampleModel sampleModel = null;
88
89     /** The image's ColorModel. */
90     protected ColorModel colorModel = null;
91
92     /** The image's sources, stored in a Vector. */
93     protected Vector<RenderedImage> sources = new Vector<RenderedImage>();
94 }
```

Podemos observar nesta classe bastantes inteiros que representam coordenadas e dimensões, é possível então ter uma classe com estas constantes e fazendo nesta classe alguns dos cálculos tornando tudo mais claro e fácil de perceber.

-----> (SimpleRenderedImage.java) <-----

Relatório Fase 1 Francisco Silveira (60816)

- Padrões:

1.Template Method Pattern:

(ShortDateFormatOption.java & DefaultStringOption.java)

```
@Override
public void loadPersistentValue(String value) {
    super.loadPersistentValue(value);
    GanttLanguage.getInstance().setShortDateFormat(myDateFormat);
}
```

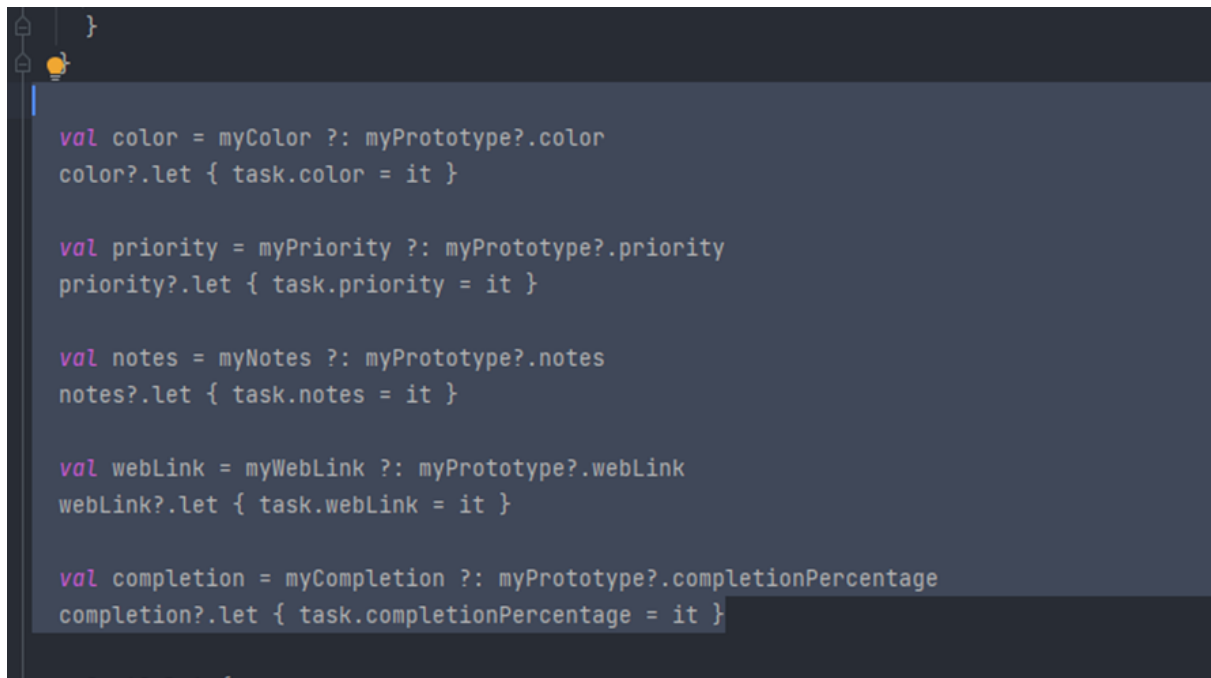
```
@Override
public void loadPersistentValue(String value) { setValue(value); }
}
```

Neste padrão podemos observar que existe um “override” do método “loadPersistentValue”.

Isto permite modificar ou fazer algumas alterações ao método da superclasse pois as classes que estendem a super classe podem por exemplo ter uma análise e processamento de dados idêntica mas lidar com os vários formatos de dados de maneira diferente.

2. Prototype pattern:

(taskManagerImpl.kt)



```
}  
  
val color = myColor ?: myPrototype?.color  
color?.let { task.color = it }  
  
val priority = myPriority ?: myPrototype?.priority  
priority?.let { task.priority = it }  
  
val notes = myNotes ?: myPrototype?.notes  
notes?.let { task.notes = it }  
  
val webLink = myWebLink ?: myPrototype?.webLink  
webLink?.let { task.webLink = it }  
  
val completion = myCompletion ?: myPrototype?.completionPercentage  
completion?.let { task.completionPercentage = it }
```

Aqui podemos ver que é utilizado um protótipo de um objeto já existente para ser comparado com outros valores, ou seja, utilizamos a cópia de um objeto para podermos trabalhar nele à vontade.

Isto permite que não seja possível alterar o objeto inicial criando uma maior segurança no nosso código e dar mais liberdade ao programador pois este pode fazer os testes e alterações que ache necessário para o seu programa no protótipo.

3.Iterator pattern:

(CustumColumnsStorage.java)

```
public void addCustomColumnsListener(CustomPropertyListener listener) { myListeners.add(l

private void fireCustomColumnsChange(CustomPropertyEvent event) {
    Iterator<CustomPropertyListener> it = myListeners.iterator();
    while (it.hasNext()) {
        CustomPropertyListener listener = it.next();
        listener.customPropertyChange(event);
    }
}
```

Neste pedaço de código conseguimos facilmente identificar a criação de um iterador. Estes são utilizados para percorrer uma lista ou outro tipo de estrutura de dados com maior eficiência.

- Code Smells:

1.Switch statement:

(GanttCSVExport.java)

Neste programa podemos ver que existem case's dentro do switch sem qualquer tipo de código, ou seja, não servem para nada. Neste caso, deveríamos remover esses cases ou em alguns casos específicos criar um case "Default" que trataria de todos os cases que não é suposto executar código.

```
end
case END_DATE:
    writer.print(task.getDisplayEnd());
    break;
case DURATION:
    writer.print(task.getDuration().getLength());
    break;
case COMPLETION:
    writer.print(task.getCompletionPercentage());
    break;
case OUTLINE_NUMBER:
    List<Integer> outlinePath = task.getManager().getTaskHierarchy().getOutlinePath(task);
    writer.print(Joiner.on('.').join(outlinePath));
    break;
case COORDINATOR:
    ResourceAssignment coordinator = Iterables.tryFind(Arrays.asList(task.getAssignments()), COORDINATOR_PREDICATE).orElse(null);
    writer.print(coordinator == null ? "" : coordinator.getResource().getName());
    break;
case PREDECESSORS:
    writer.print(TaskProperties.formatPredecessors(task, ";", true));
    break;
case RESOURCES:
    writer.print(getAssignments(task));
    writer.print(buildAssignmentSpec(task));
    break;
case COST:
    writer.print(task.getCost().getValue());
    break;
case COLOR:
    if (!Objects.equal(task.getColor(), task.getManager().getTaskDefaultColorOption().getValue())) {
        writer.print(ColorConvention.getColor(task.getColor()));
    } else {
        writer.print("");
    }
    break;
case INFO:
case PRIORITY:
case TYPE:
    break;
}
```

⚠ Kotlin code style
Do you want to use
Apply the code style

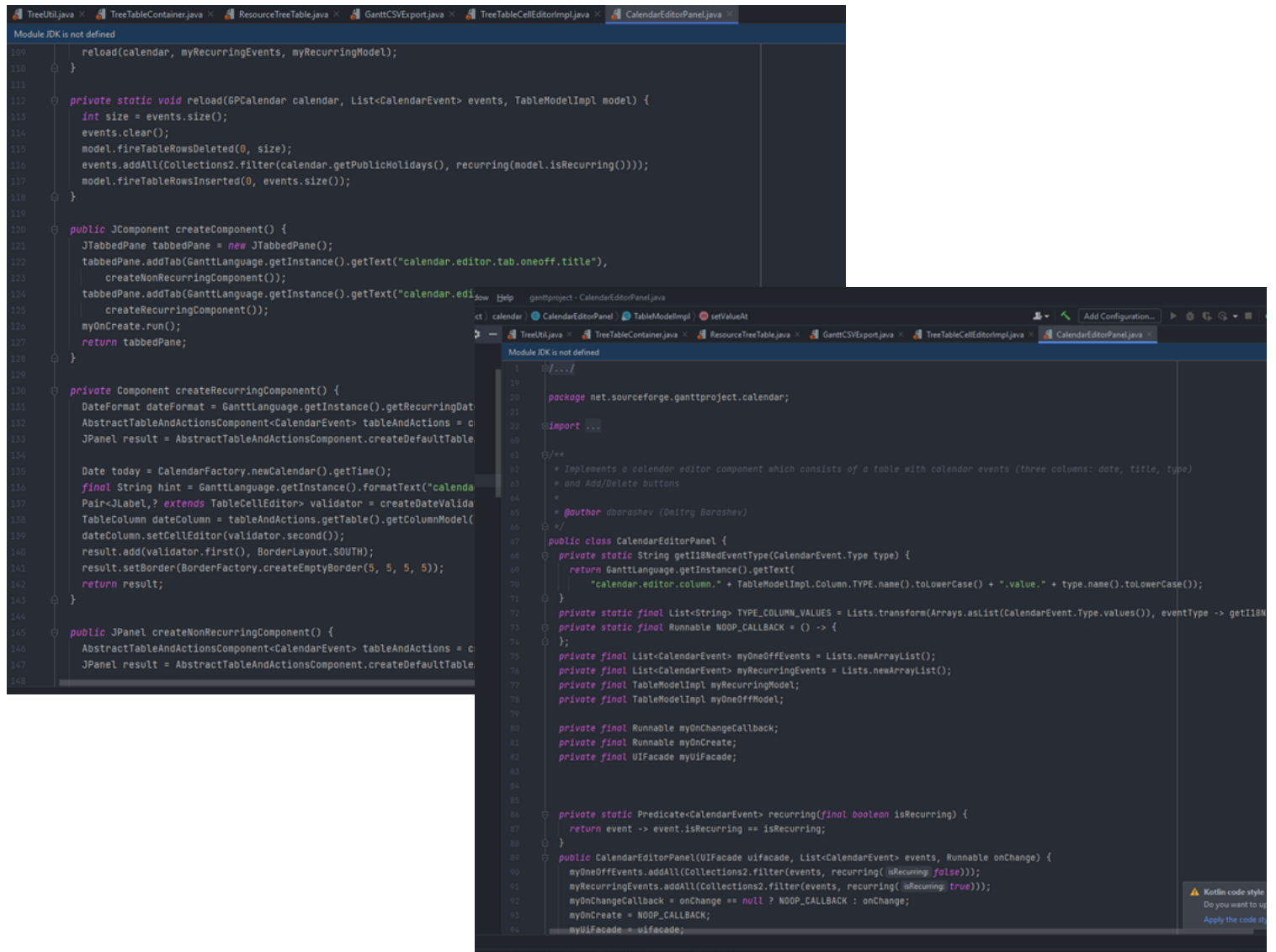
2.

(TaskDisplayColumnsTagHandler.java)

```
@Override
protected boolean onStartElement(Attributes attrs) {
    if (!isEnabled) {
        return false;
    }
    loadTaskDisplay(attrs);
    return true;
}
```

Como podemos ver neste pedaço de código no primeiro “if” tratamos primeiro da negação de um método booleano. Deveríamos testar a condição “isEnabled” dentro do “if” e executar “loadTaskDisplay()” e retornar “true”. De seguida depois do “if” retornávamos “false”.

3.No Comments (CalendarEditorPanel):



```
109 reload(calendar, myRecurringEvents, myRecurringModel);
110 }
111
112 private static void reload(GPCalendar calendar, List<CalendarEvent> events, TableModelImpl model) {
113     int size = events.size();
114     events.clear();
115     model.fireTableRowsDeleted(0, size);
116     events.addAll(Collections2.filter(calendar.getPublicHolidays(), recurring(model.isRecurring())));
117     model.fireTableRowsInserted(0, events.size());
118 }
119
120 public JComponent createComponent() {
121     JTabbedPane tabbedPane = new JTabbedPane();
122     tabbedPane.addTab(GanttLanguage.getInstance().getText("calendar.editor.tab.oneoff.title"),
123         createNonRecurringComponent());
124     tabbedPane.addTab(GanttLanguage.getInstance().getText("calendar.editor.tab.recurring.title"),
125         createRecurringComponent());
126     myOnCreate.run();
127     return tabbedPane;
128 }
129
130 private Component createRecurringComponent() {
131     DateFormat dateFormat = GanttLanguage.getInstance().getRecurringDateFormat();
132     AbstractTableAndActionsComponent<CalendarEvent> tableAndActions = createTableAndActionsComponent(dateFormat);
133     JPanel result = AbstractTableAndActionsComponent.createDefaultTableAndActionsComponent(tableAndActions);
134
135     Date today = CalendarFactory.newCalendar().getTime();
136     final String hint = GanttLanguage.getInstance().formatText("calendar.editor.recurring.hint", today);
137     Pair<JLabel,? extends TableCellEditor> validator = createDateValidator(today, hint);
138     TableColumn dateColumn = tableAndActions.getTable().getColumnModel().getColumn(0);
139     dateColumn.setCellEditor(validator.second());
140     result.add(validator.first(), BorderLayout.SOUTH);
141     result.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
142     return result;
143 }
144
145 public JPanel createNonRecurringComponent() {
146     AbstractTableAndActionsComponent<CalendarEvent> tableAndActions = createTableAndActionsComponent();
147     JPanel result = AbstractTableAndActionsComponent.createDefaultTableAndActionsComponent(tableAndActions);
148 }
```

```
1 //...
2
3 package net.sourceforge.ganttproject.calendar;
4
5 import java.util.*;
6
7 /**
8  * Implements a calendar editor component which consists of a table with calendar events (three columns: date, title, type)
9  * and Add/Delete buttons
10  *
11  * @author dbarashev (Dmitry Barashev)
12  */
13 public class CalendarEditorPanel {
14     private static String getI18NedEventType(CalendarEvent.Type type) {
15         return GanttLanguage.getInstance().getText(
16             "calendar.editor.column." + TableModelImpl.Column.TYPE.name().toLowerCase() + ".value." + type.name().toLowerCase());
17     }
18
19     private static final List<String> TYPE_COLUMN_VALUES = Lists.transform(Arrays.asList(CalendarEvent.Type.values()), eventType -> getI18NedEventType(eventType));
20     private static final Runnable NOOP_CALLBACK = () -> {};
21
22     private final List<CalendarEvent> myOneOffEvents = Lists.newArrayList();
23     private final List<CalendarEvent> myRecurringEvents = Lists.newArrayList();
24     private final TableModelImpl myRecurringModel;
25     private final TableModelImpl myOneOffModel;
26
27     private final Runnable myOnChangeCallback;
28     private final Runnable myOnCreate;
29     private final UIFacade myUIFacade;
30
31     private static Predicate<CalendarEvent> recurring(final boolean isRecurring) {
32         return event -> event.isRecurring == isRecurring;
33     }
34
35     public CalendarEditorPanel(UIFacade uifacade, List<CalendarEvent> events, Runnable onChange) {
36         myOneOffEvents.addAll(Collections2.filter(events, recurring(false)));
37         myRecurringEvents.addAll(Collections2.filter(events, recurring(true)));
38         myOnChangeCallback = onChange == null ? NOOP_CALLBACK : onChange;
39         myOnCreate = NOOP_CALLBACK;
40         myUIFacade = uifacade;
41 }
```

Esta Classe esta sem comentários, o que é essencial para a interpretação da mesma. Sem comentários, alguém ou alguma equipa que fosse trabalhar no nosso código iria ter bastante dificuldade em entender o programa. O que dificultaria o trabalho da equipa e faria com que demorasse muito mais tempo.