Code Smells

Francisco Vale:

Code Smell 1: Dead Code/Speculative Generality

```
42
43      protected UIFacade getUiFacade() { return myUiFacade; }
46
47      protected IGanttProject getProject() { return myProject; }
50
51      protected Preferences getPreferences() { return myPreferences; }
54
55      protected Chart getGanttChart() { return myUiFacade.getGanttChart(); }
58
59      protected Chart getResourceChart() { return myUiFacade.getResourceChart(); }
62
63  }
```

Location:

Line 55 and 59;

Synopsis: Dead Code is found when we have obsolete code that is never called throughout the entire project, which is similar to Speculative generality, but here the code is made with future intentions, even tho it is not being used at the moment.

Rationale: This class has methods that are never called in the entire code, which makes this obsolete, even if it is with future intentions, if is not something requested from the client then it is only a waste of time .

Refactor proposal: I would delete this piece of code since it is not being used and only makes the code more extensive.

Code Smell 2: Data Class

```java
/.../
package org.ganttproject.impex.htmlpdf;

import ...

/**
 * Simple base class for the rendering engines.
 *
 * @author dbarashev (Dmitry Barashev)
 */
public class AbstractEngine {
  private IGanttProject myProject;
  private UIFacade myUiFacade;
  private Preferences myPreferences;

  public void setContext(IGanttProject project, UIFacade uiFacade, Preferences preferences) {
    myProject = project;
    myUiFacade = uiFacade;
    myPreferences = preferences;
  }

  protected UIFacade getUiFacade() { return myUiFacade; }

  protected IGanttProject getProject() { return myProject; }

  protected Preferences getPreferences() { return myPreferences; }

  protected Chart getGanttChart() { return myUiFacade.getGanttChart(); }

  protected Chart getResourceChart() { return myUiFacade.getResourceChart(); }

}
```

Location:

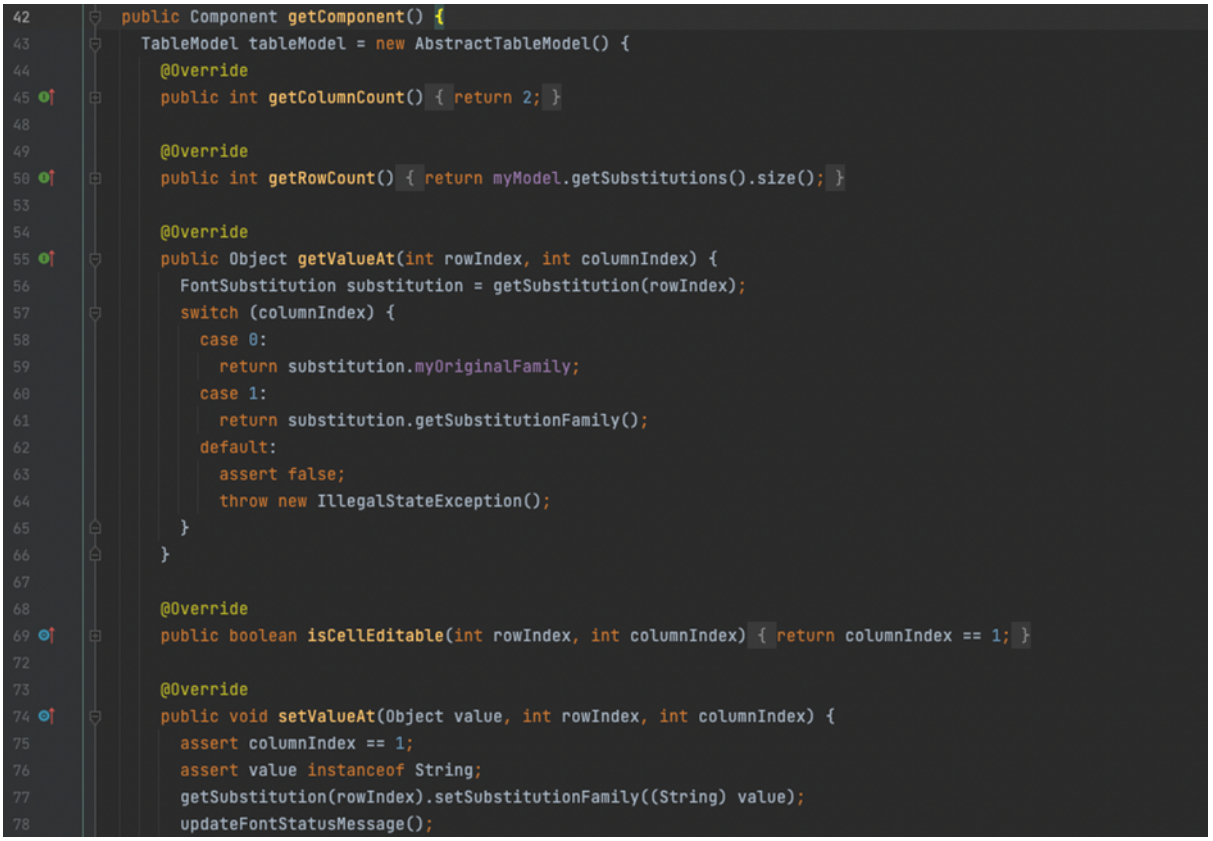
ganttproject 〉 org.ganttproject.impex.htmlpdf 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 impex 〉 htmlpdf 〉 AbstractEngine

Synopsis: A Data Class is a class in which we only have getter and setter methods, and so it has no real functionality, which may indicate it may not be a good abstraction, or a necessary class.

Rationale: As we can observe the entire class as nothing but get methods.

Refactor: I would suggest trying to understand if there is something other than getters that would make sense to be placed in this class or, on the other hand, trying to see if the classes that use this gets can place this behaviours in their own class.

Code Smell 3: Long Method



```
42  public Component getComponent() {
43    TableModel tableModel = new AbstractTableModel() {
44      @Override
45      public int getColumnCount() { return 2; }
48
49      @Override
50      public int getRowCount() { return myModel.getSubstitutions().size(); }
53
54      @Override
55      public Object getValueAt(int rowIndex, int columnIndex) {
56        FontSubstitution substitution = getSubstitution(rowIndex);
57        switch (columnIndex) {
58          case 0:
59            return substitution.myOriginalFamily;
60          case 1:
61            return substitution.getSubstitutionFamily();
62          default:
63            assert false;
64            throw new IllegalStateException();
65        }
66      }
67
68      @Override
69      public boolean isCellEditable(int rowIndex, int columnIndex) { return columnIndex == 1; }
72
73      @Override
74      public void setValueAt(Object value, int rowIndex, int columnIndex) {
75        assert columnIndex == 1;
76        assert value instanceof String;
77        getSubstitution(rowIndex).setSubstitutionFamily((String) value);
78        updateFontStatusMessage();
```

Location:

From line 42 to 134;

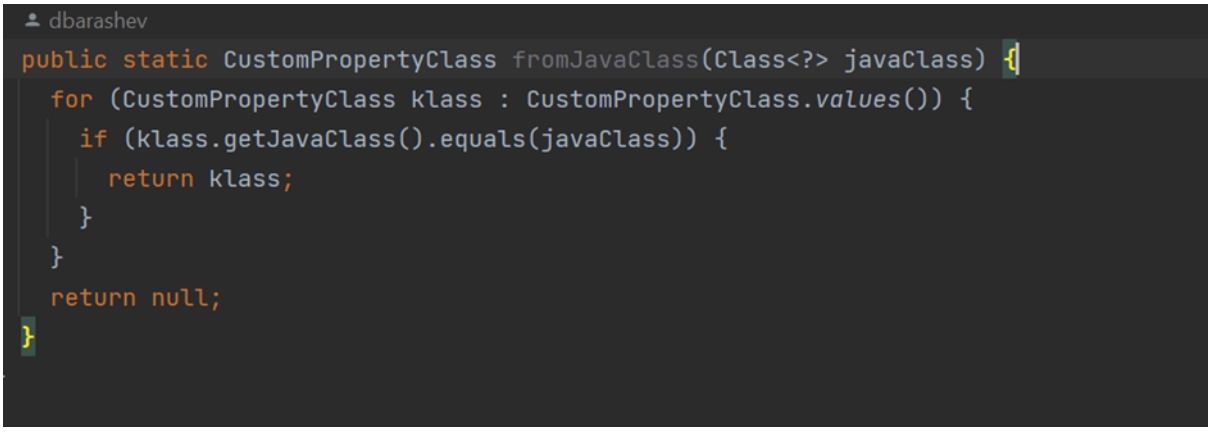
Synopsis: A Long Method, as its name suggests, is a method which is too long, and so, it may suggest that the method is too complex and does too much stuff that it probably didn't need.

Rationale: As we can see in the previous image the code is really extensive, and it continues for sixty more lines. Also it does various operations that probably could be divided throughout the code.

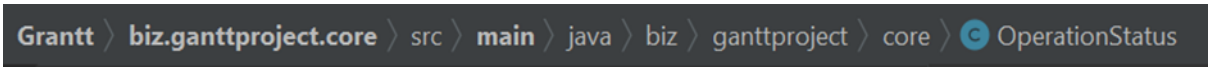
Refactor: As I said before this method deals with a lot of operations, and so I would try to divide them and try to put them in more adequate spots, rather than being all in the same method.

Guilherme Franco:

Code Smell 1: Dead Code/Speculative Generality



Location:



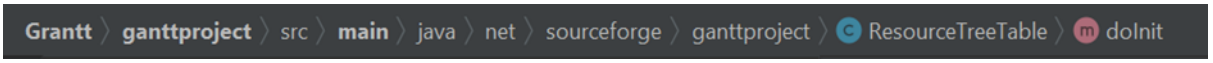
Line 66;

Rationale: This method is not used or called anywhere in the application.

Refactor proposal: I believe this was added to be used in the future but since it's not used anywhere at the moment and there is no current need for it, I would delete this piece of obsolete code.

Code Smell 2: Message Chains

Location:



Line 152 or 155

Rationale: In line 152 the code calls 4 sucessive objects just to receive the column.

In line 155 the code calls a total of 5 sucessive objects.

Refactor: I would suggest structuring the classes differently, so they don't have to call so many objects or at least creating local variables so it's more understandable.

Code Smell 3: Duplicated Code

1)

2)

```
2 usages  dbarashev
private void setEditor(ColumnImpl column) {
  if (column == null || column.getTableColumnExt() == null) {
    return;
  }
  JComboBox comboBox = new JComboBox(getRoleManager().getEnabledRoles());
  comboBox.setEditable(false);
  column.getTableColumnExt().setCellEditor(new DefaultCellEditor(comboBox));
}
});
```

Location:

Grantt 〉 ganttproject 〉 src 〉 main 〉 java 〉 net 〉 sourceforge 〉 ganttproject 〉 ResourceTreeTable

On lines 75-76 and 194-195

Rationale: Inside these two methods there is duplicated code so if we needed to alter the method of creating “JcomboBoxes” we would need to do it in two different places

Refactor: I would create a new method that creates these jumbo boxes and just call that method inside “setEditor” and “newTableColumnExt”.

Luís Abreu:

Code Smell 1: Data Class

```
dbarashev
public class TimeUnitPair {
  2 usages
  private final TimeUnit myBottomTimeUnit;

  2 usages
  private final TimeUnit myTopTimeUnit;

  2 usages
  private final TimeUnitStack myTimeUnitStack;

  /** Used scale for this TimeUnit */
  2 usages
  private final int myDefaultUnitWidth;

  10 usages  dbarashev
  public TimeUnitPair(TimeUnit topUnit, TimeUnit bottomUnit, TimeUnitStack timeUnitStack, int defaultUnitWidth) {
    myTopTimeUnit = topUnit;
    myBottomTimeUnit = bottomUnit;
    myTimeUnitStack = timeUnitStack;
    myDefaultUnitWidth = defaultUnitWidth;
  }

  2 usages  dbarashev
  public TimeUnit getTopTimeUnit() { return myTopTimeUnit; }

  2 usages  dbarashev
  public TimeUnit getBottomTimeUnit() { return myBottomTimeUnit; }

  dbarashev
  public TimeUnitStack getTimeUnitStack() { return myTimeUnitStack; }

  /** @return the scale for this TimeUnit */
  1 usage  dbarashev
  public int getDefaultUnitWidth() { return myDefaultUnitWidth; }
}
```

Location:

ganttproject 〉 biz.ganttproject.core 〉 src 〉 main 〉 java 〉 biz 〉 ganttproject 〉 core 〉 time 〉 TimeUnitPair

Lines: whole class, but more important methods in between 26-39

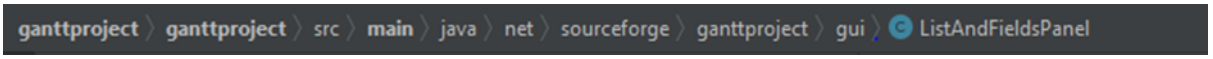
Rationale: Inside this class there is only data, no real functionality, only getter and setters methods.

Refactor: I would create a better abstraction: see if i could put something else in this class (other methods), see what classes are manipulating this data. If I failed to solve the problem, I would delete this class and try to figure out if this data would be better store in another class.

Code Smell 2: Long Method

```
dbarashev
public JComponent getComponent() {
  if (myPanel == null) {
    SpringLayout topPanelLayout = new SpringLayout();
    JPanel topPanel = new JPanel(topPanelLayout);

    JComponent depsComponent = myList.getTableComponent();
    JComponent titleComponent = new JLabel(myList.getTitle());
    JComponent actionsComponent = myList.getActionsComponent();
    topPanel.add(titleComponent);
    topPanel.add(actionsComponent);

    topPanelLayout.putConstraint(SpringLayout.WEST, titleComponent, pad: 0, SpringLayout.WEST, topPanel);
    topPanelLayout.putConstraint(SpringLayout.NORTH, titleComponent, pad: 0, SpringLayout.NORTH, topPanel);

    topPanelLayout.putConstraint(SpringLayout.NORTH, actionsComponent, pad: 2, SpringLayout.SOUTH, titleComponent);
    topPanelLayout.putConstraint(SpringLayout.SOUTH, topPanel, pad: 2, SpringLayout.SOUTH, actionsComponent);
    topPanelLayout.putConstraint(SpringLayout.WEST, actionsComponent, pad: 0, SpringLayout.WEST, topPanel);

    JPanel centerPanel = new JPanel(new BorderLayout());
    centerPanel.add(depsComponent, BorderLayout.CENTER);

    myFields.setBorder(BorderFactory.createEmptyBorder( top: 0, left: 10, bottom: 0, right: 0));
    centerPanel.add(myFields, BorderLayout.EAST);

    myPanel = Box.createVerticalBox();
    myPanel.setBorder(BorderFactory.createEmptyBorder( top: 5, left: 5, bottom: 5, right: 5));
    myPanel.add(topPanel);
    myPanel.add(Box.createVerticalStrut( height: 5));
    myPanel.add(centerPanel);
  }
  return myPanel;
}
}
```

Location:



Lines: 40-72

Rationale: This method is way too big, which can lead to bad understanding of the code and can be a sign of being more complex than it needs to be.

Refactor: I would create smaller private methods that would be called inside the method *public JComponent getComponent()*.

This method will be better documentated and will provide better understanding of the code, for example one of this smaller private methods could be named *putConstraint()* and would be responsible for the lines 51-56.

Code Smell 3: Duplicated Code

```
184     /** Move Up the selected resource */
        1 usage  dbarashev
185     public boolean moveUp(HumanResource resource) {
186       myResourceManager.up(resource);
187       ResourceNode rn = getNodeForResource(resource);
188       int index = TreeUtil.getPrevSibling(root, rn);
189       if (index == -1) {
190         return false;
191       }
192       removeNodeFromParent(rn);
193       insertNodeInto(rn, root, index);
194       return true;
195     }
196
        1 usage  dbarashev
197     public boolean moveDown(HumanResource resource) {
198       myResourceManager.down(resource);
199       ResourceNode rn = getNodeForResource(resource);
200       int index = TreeUtil.getNextSibling(root, rn);
201       if (index == -1) {
202         return false;
203       }
204       removeNodeFromParent(rn);
205       insertNodeInto(rn, root, index);
206       return true;
207     }
208
```

Location:

ganttproject 〉 ganttproject 〉 src 〉 main 〉 java 〉 net 〉 sourceforge 〉 ganttproject 〉 ResourceTreeTableModel 〉 moveUp

Lines: 189-194 and 201-206

Rationale: Inside these two red boxes there is duplicated code so if we needed to alter something in the way the code deals with the fact that there is or is not a previous/next sibling we would need to do it in two different places.

Refactor: I would create a new method that has this code and call inside both methods *moveUp()* and *moveDown()*. By doing this the program would perform the same tasks but with less code.

Pedro Gouveia:

1- Switch statements

```
switch (type) {
case PertChartAbstraction.Type.NORMAL:
  color = NORMAL_COLOR;
  break;
case PertChartAbstraction.Type.SUPER:
  color = SUPER_COLOR;
  break;
case PertChartAbstraction.Type.MILESTONE:
  color = MILESTONE_COLOR;
  break;
default:
  color = NORMAL_COLOR;
}
```

```
static class Type {
  2 usages
  public static final int NORMAL = 0;

  2 usages
  public static final int SUPER = 1;

  2 usages
  public static final int MILESTONE = 2;
}
```

Location:

ganttproject 〉 org.ganttproject.chart.pert 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 chart 〉 pert 〉 ActivityOnNodePertChart 〉 GraphicalNode 〉 paintMe

Rationale: here the type of graph is determined by an integer and then there are a bunch of switch statements regarding the behavior of such graph depending on the type of graph it is. This should be resolved by using polymorphism to specify the type of graph.
Also, if there is the need to add functionalities to different types of graphs we would need to add a switch statement for which, and if we want to add a new graph we would need to change every scattered switch statement.

Refractor: Should be created an abstract class from which 3 different subclasses inherit.

2- Law of demeter

```
74      } else if (myTaskManager.getTaskHierarchy().getNestedTasks(task).length == 0) {
```

Location:

ganttproject 〉 org.ganttproject.chart.pert 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 chart 〉 pert 〉 PertChartAbstraction 〉 getTaskGraphNode

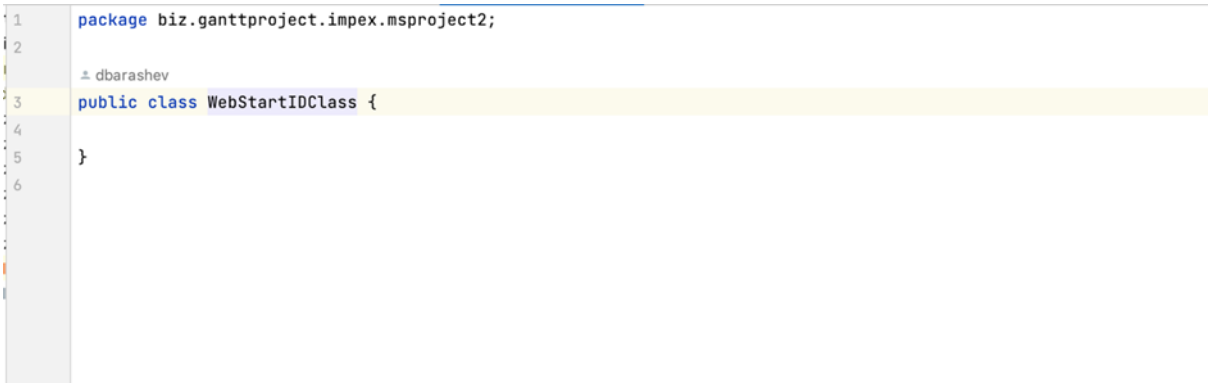
Rationale: Here *PerChartAbstraction* class in its method *getTaskGraphNode( )* is calling a method named *getNestedTasks()*, and this method is not:

- Encapsulated within the same object – the method *getNestedTasks()*, is from an object of the type *TaskContainmentHierarchyFacade*, so it obviously isn't from the same class as *getTaskGraphNode(),* or in other words is not from *PerChartAbstraction* class.

- Encapsulated within an object that is in the parameters of M - the object from which this method is from is received from the method *getTaskHierarchy()* of the myTaskManager object, and this object is not from the parameters of *getNestedTasks()*, it is just returned from a function.
- Encapsulated within an object that is instantiated inside the M - the object from which this method is from is received from the method *getTaskHierarchy()* of the *myTaskManager* object and this object is not instantiated inside *getNestedTasks()*, it is just returned from a function.
- Encapsulated within an object that is referenced in an instance variable of the class in M - the object from which this method is from is received from the method *getTaskHierarchy()* of the *myTaskManager* object and not from an instance variable of the class *PerChartAbstraction*.

Refractor: There should be a method in task manager that returned the nested tasks.

3-Dead Code



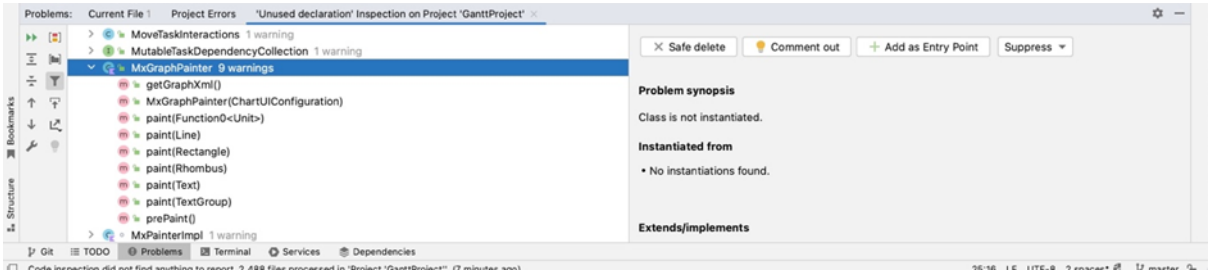
Location:

Rationale: This class doesn't have anything and is not implemented anywhere or used. It was probably created to implement something that stopped making sense before it was implemented or is a placeholder for something to do in the future.

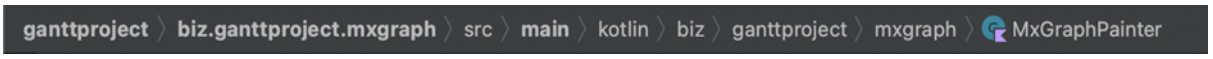
Refractor: This class should not exist before it is needed.

Tiago Francisco:

Code smell 1: Dead code:



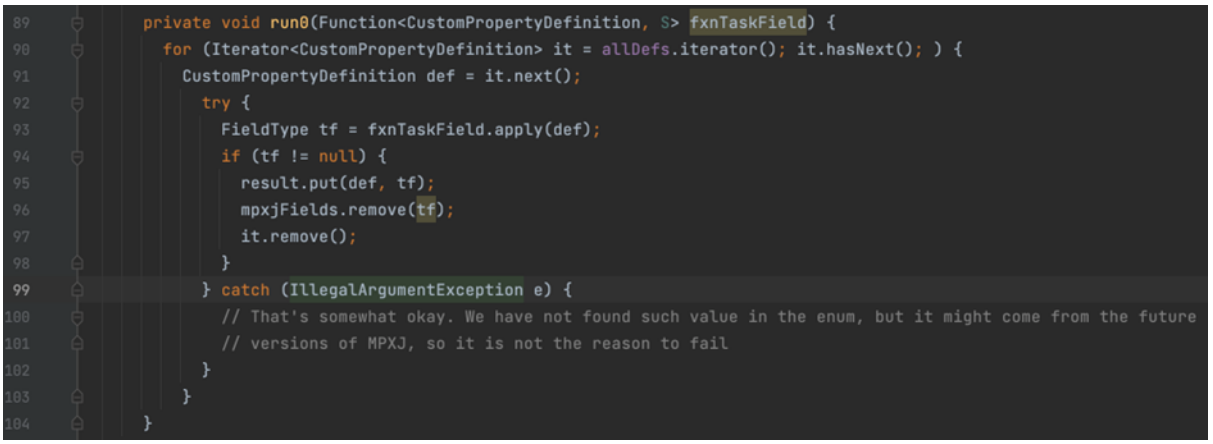
Location:



Rationale: This class is not used or instantiated anywhere in the application.

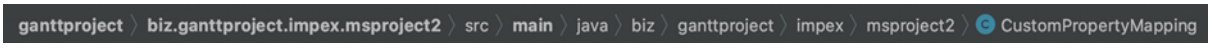
Refactor proposal: It might be a class that is not used anymore, so it should be deleted.

Code smell 2: Speculative Generality:



```
89    private void run0(Function<CustomPropertyDefinition, S> fxnTaskField) {
90      for (Iterator<CustomPropertyDefinition> it = allDefs.iterator(); it.hasNext(); ) {
91        CustomPropertyDefinition def = it.next();
92          try {
93            FieldType tf = fxnTaskField.apply(def);
94            if (tf != null) {
95              result.put(def, tf);
96              mpxjFields.remove(tf);
97              it.remove();
98            }
99          } catch (IllegalArgumentException e) {
100           // That's somewhat okay. We have not found such value in the enum, but it might come from the future
101           // versions of MPXJ, so it is not the reason to fail
102         }
103       }
104   }
```

Location:



Line 99:

Rationale: This catch clause was written thinking in something that might happen in the future but doesn't happen at this moment.

Refactor proposal: It should be deleted to make the code clearer.

Code Smell 3: Law of Demeter

```
72        float scaleFactor = fontSpec.getSize().getFactor() * getDpi();
```

Location:

Rationale: Here *PerChart* class in its method *updateFonts( )* is calling a method named *getFactor()*, and this method is not:

Encapsulated within the same object – the method *getFactor()*, is from an object of the type *Size*, so it obviously isn't from the same class as *updateFonts(),* or in other words is not from *PerChart* class.

Encapsulated within an object that is in the parameters of *updateFonts()* - the object from which this method is from is received from the method *getSize()* of the *fontSpec* object, and this object is not from the parameters of *updatedFonts()*, it is just returned from a function.

Encapsulated within an object that is instantiated inside *updateFonts()*- the object from which this method is from is received from the method *getSize()* of the *fontSpec* object and this object is not instantiated inside *updateFonts()*, it is just returned from a function.

Encapsulated within an object that is referenced in an instance variable of the class in *updateFonts()* - the object from which this method is from is received from the method *getSize()* of the *fontSpec* object and not from an instance variable of the class *PerChart*.

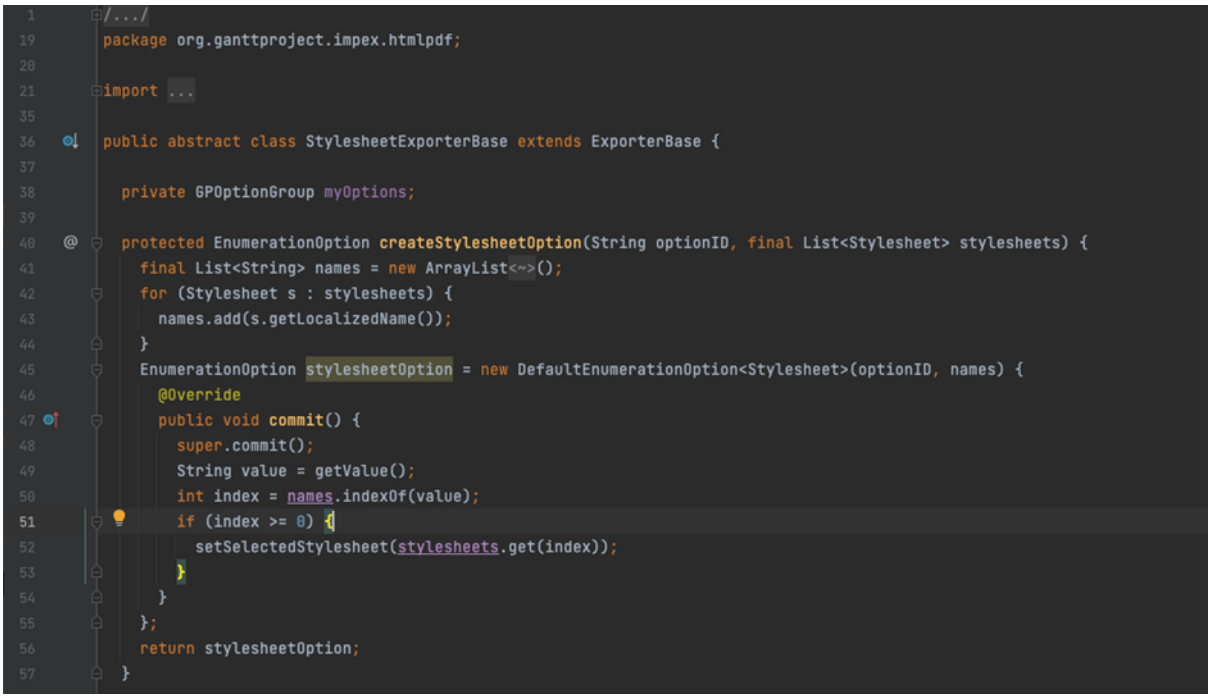
Refactor proposal: I would create a method in the FontSpec class that returns the factor since the object is received from the constructor parameters in that class.

Design Patterns

Francisco Vale:

Design Pattern 1: Template Method Pattern

SuperClass:



```
/.../
package org.ganttproject.impex.htmlpdf;

import ...

public abstract class StylesheetExporterBase extends ExporterBase {

  private GPOptionGroup myOptions;

  protected EnumerationOption createStylesheetOption(String optionID, final List<Stylesheet> stylesheets) {
    final List<String> names = new ArrayList<~>();
    for (Stylesheet s : stylesheets) {
      names.add(s.getLocalizedName());
    }
    EnumerationOption stylesheetOption = new DefaultEnumerationOption<Stylesheet>(optionID, names) {
      @Override
      public void commit() {
        super.commit();
        String value = getValue();
        int index = names.indexOf(value);
        if (index >= 0) {
          setSelectedStylesheet(stylesheets.get(index));
        }
      }
    };
    return stylesheetOption;
  }
```

SubClasses:

```java
/.../
package org.ganttproject.impex.htmlpdf;

import ...

public class ExporterToHTML extends StylesheetExporterBase {
  static final String GANTT_CHART_FILE_EXTENSION = "png";
  static final String RESOURCE_CHART_FILE_EXTENSION = "res.png";
  private static final String PNG_FORMAT_NAME = "png";
  private HTMLStylesheet mySelectedStylesheet;
  private final HtmlSerializer mySerializer = new HtmlSerializer( engine: this);
```

```java
/.../
package org.ganttproject.impex.htmlpdf;

import ...


public class ExporterToPDF extends StylesheetExporterBase {

  private final ITextEngine myITextEngine = new ITextEngine( exporter: this);
  private Stylesheet mySelectedStylesheet;

  @Override
  protected ExporterJob[] createJobs(File outputFile, List<File> resultFiles) {
    super.setCommandLineStylesheet();
```

Locations:

Synopsis: Defines an algorithm's steps generally, deferring the implementation of some steps to subclasses. In other words, it is concerned with the assignment of responsibilities and it makes sense to be used when we have two separated classes with really similar functionalities.

Rationale: We have StylesheetExporterBase as the SuperClass and ExporterToHTML and ExporterToPDF being the SubClasses. Both of the SubClasses implement the methods that are not implemented in the Abstract Class and since they are similar Classes it just made sense to have a SuperClass that centralises some of the similar functionalities.

Design Pattern 2: Chain of Responsibility Design Pattern

ITextEngine:

```
94
95      private Component createFontPanel() {
96        return new FontSubstitutionPanel(mySubstitutionModel).getComponent();
97      }
98
```

FontSubstitutionPanel:

```
119
120     myFamiliesComboBox = new JComboBox(myModel.getAvailableSubstitutionFamilies().toArray(new String[0]));
121     table.getColumnModel().getColumn( columnIndex: 0).setCellRenderer(new CellRendererImpl());
```

(Inside getComponent method in line 42)

FontSubstitutionModel:

```
109     public List<String> getAvailableSubstitutionFamilies() {
110       return myFontCache.getRegisteredFamilies();
111     }
```

TTFontCache:

```
79
80      public List<String> getRegisteredFamilies() {
81        return new ArrayList<>(myMap_Family_RegularFont.keySet());
82      }
```

Location:

ganttproject 〉 org.ganttproject.impex.htmlpdf 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 impex 〉 htmlpdf 〉 itext 〉 ITextEngine

ganttproject 〉 org.ganttproject.impex.htmlpdf 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 impex 〉 htmlpdf 〉 itext 〉 FontSubstitutionPanel

ganttproject 〉 org.ganttproject.impex.htmlpdf 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 impex 〉 htmlpdf 〉 itext 〉 FontSubstitutionModel

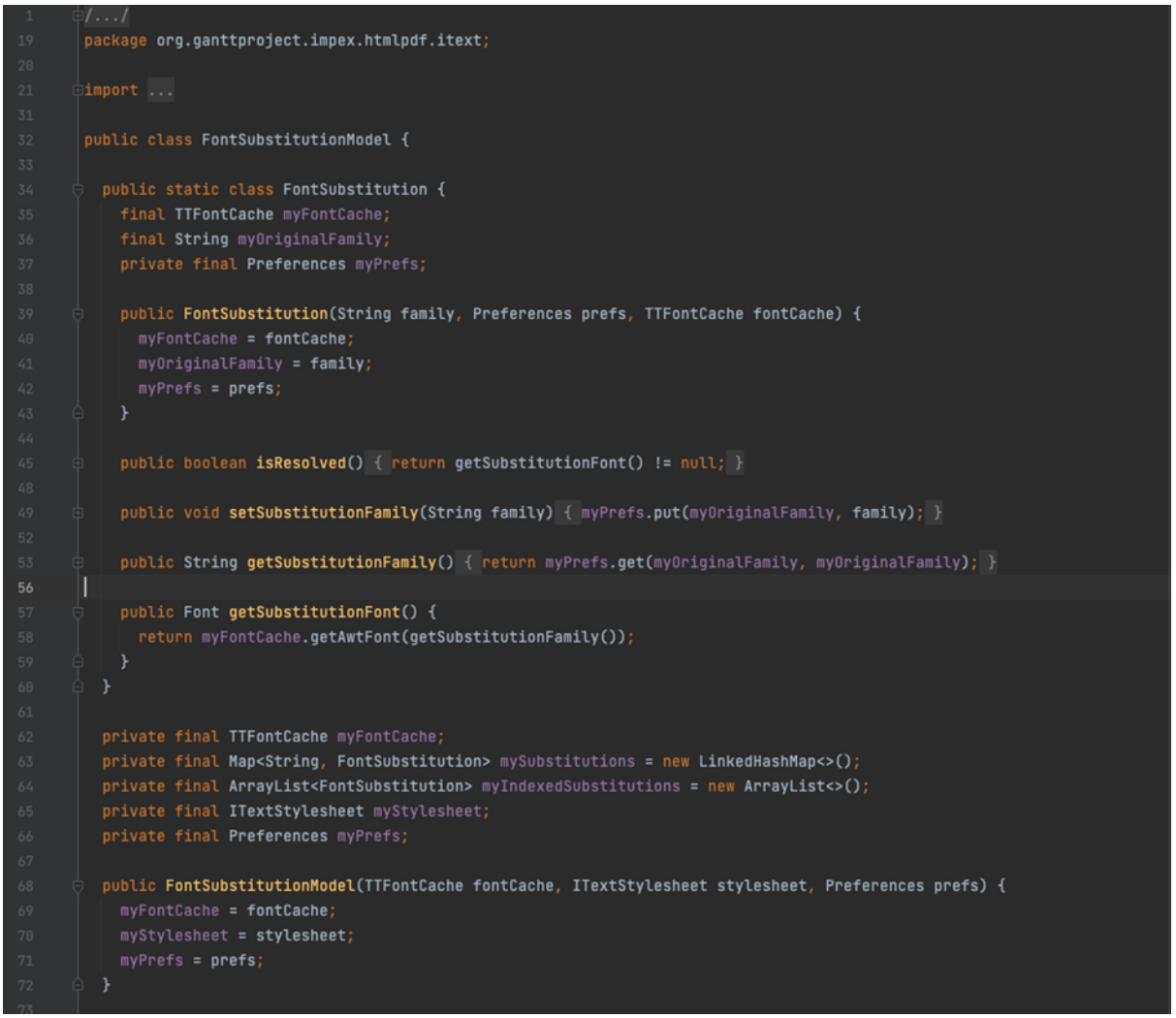
ganttproject 〉 org.ganttproject.impex.htmlpdf 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 impex 〉 htmlpdf 〉 fonts 〉 TTFontCache

Synopsis: This pattern is used when a client sends a request and then this request is propagates until a class is able to handle it.

Rationale: As we can see in the images above, this classes propagate the first request which was the createFontPanel() in line 87. The request is being propagated until it reaches a class that can handle it, TTFontCache.

Design Pattern 3: Visitor Design Pattern

Visitor Class:



```
/.../
package org.ganttproject.impex.htmlpdf.itext;

import ...

public class FontSubstitutionModel {

  public static class FontSubstitution {
    final TTFontCache myFontCache;
    final String myOriginalFamily;
    private final Preferences myPrefs;

    public FontSubstitution(String family, Preferences prefs, TTFontCache fontCache) {
      myFontCache = fontCache;
      myOriginalFamily = family;
      myPrefs = prefs;
    }

    public boolean isResolved() { return getSubstitutionFont() != null; }

    public void setSubstitutionFamily(String family) { myPrefs.put(myOriginalFamily, family); }

    public String getSubstitutionFamily() { return myPrefs.get(myOriginalFamily, myOriginalFamily); }

    public Font getSubstitutionFont() {
      return myFontCache.getAwtFont(getSubstitutionFamily());
    }
  }

  private final TTFontCache myFontCache;
  private final Map<String, FontSubstitution> mySubstitutions = new LinkedHashMap<>();
  private final ArrayList<FontSubstitution> myIndexedSubstitutions = new ArrayList<>();
  private final ITextStylesheet myStylesheet;
  private final Preferences myPrefs;

  public FontSubstitutionModel(TTFontCache fontCache, ITextStylesheet stylesheet, Preferences prefs) {
    myFontCache = fontCache;
    myStylesheet = stylesheet;
    myPrefs = prefs;
  }
```

One of the Composite Class:

```java
/.../
package org.ganttproject.impex.htmlpdf.fonts;

import ...

/**
 * This class collects True Type fonts from .ttf files in the registered
 * directories and provides mappings of font family names to plain AWT fonts and
 * iText fonts.
 *
 * @author dbarashev
 */
public class TTFontCache {
  private static final org.slf4j.Logger ourLogger = GPLogger.create("Export.Pdf.Fonts").delegate();
  private final Map<String, AwtFontSupplier> myMap_Family_RegularFont = new TreeMap<>();
  private final Map<FontKey, com.itextpdf.text.Font> myFontCache = new HashMap<>();
  private final Map<String, Function<String, BaseFont>> myMap_Family_ItextFont = new HashMap<>();
  private Properties myProperties;
  private Function<String, BaseFont> myFallbackFont;

  public TTFontCache() {
    try {
      myFallbackFont = createFontSupplier(FontManager.INSTANCE.getFallbackFontFile(), isEmbedded: true);
      registerFontFile(FontManager.INSTANCE.getFallbackFontFile());
    } catch (Exception e) {
      ourLogger.error("Failed to create fallback font", e);
    }
  }

```

Location:

ganttproject 〉 org.ganttproject.impex.htmlpdf 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 impex 〉 htmlpdf 〉 itext 〉 FontSubstitutionModel

ganttproject 〉 org.ganttproject.impex.htmlpdf 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 impex 〉 htmlpdf 〉 fonts 〉 TTFontCache

Synopsis: Allows you to add operations to a Composite structure without changing the structure itself, and so you can check the state of the composite without changing it.

Rationale: We have the Visitor class being FontSubstituitonModel, which has some methods that can check and change the state of the Composite in TTFontCache, such as 49, 53, 57.

Guilherme Franco:

Design Pattern 1: Adapter Pattern

IcsConverter.java | IcsFileImporter.java

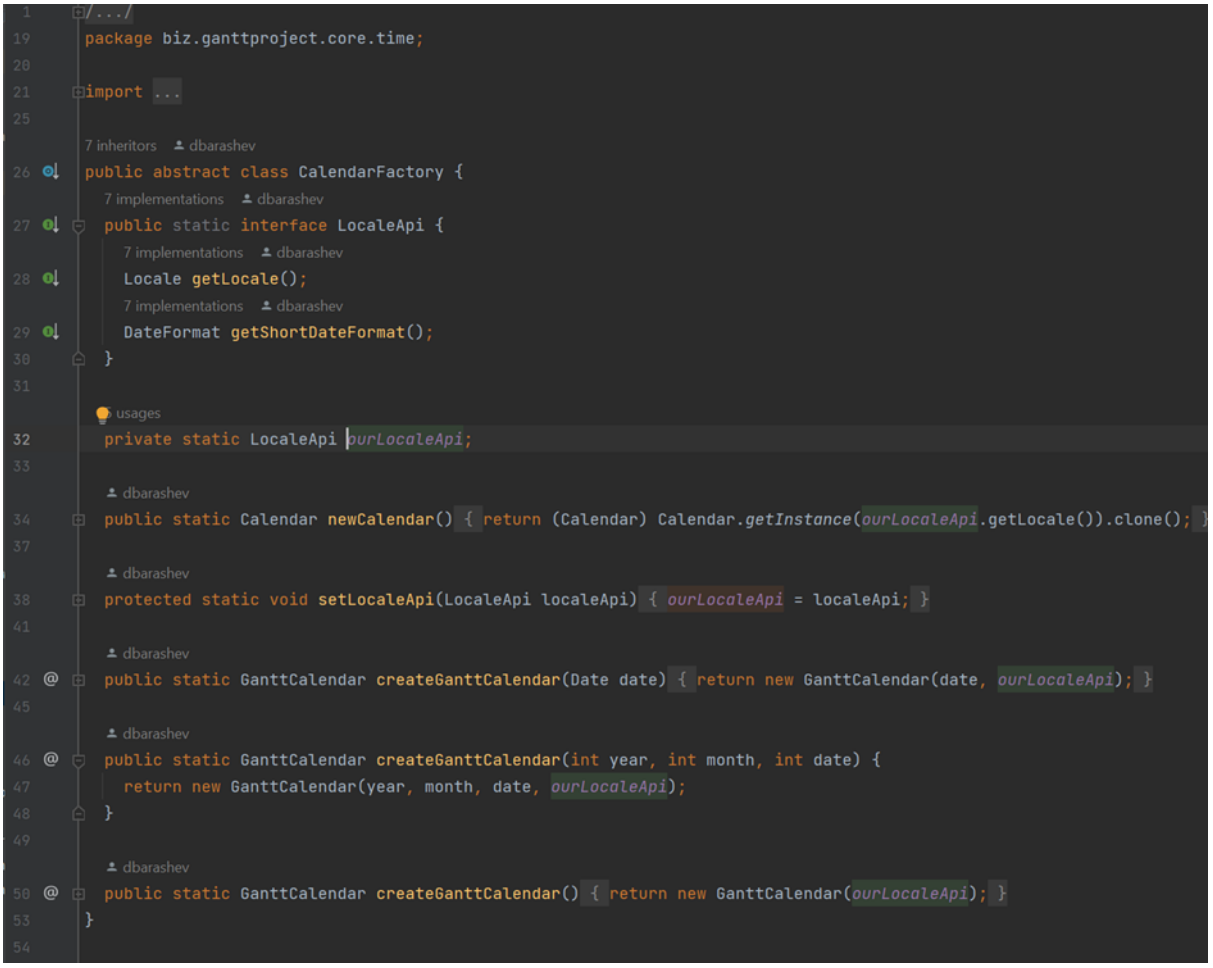
```
/.../
package biz.ganttproject.impex.ical;

import ...

/**
 * Converts ics file to GanttProject calendar format.
 *
 * @author Dmitry Barashev
 */
dbarashev
public class IcsConverter {
  2 usages  dbarashev
  public static class Args {
    2 usages
    @Parameter(description = "Input file name", required = true)
    public List<String> file = null;
  }

  dbarashev
  public static void main(String[] args) throws Exception {
    Args mainArgs = new Args();
    try {
      new JCommander(new Object[] { mainArgs }, args);
    } catch (Throwable e) {
      e.printStackTrace();
      return;
    }
    if (mainArgs.file.size() != 1) {
      System.err.println("Only 1 input file should be specified");
      return;
    }
    CalendarBuilder builder = new CalendarBuilder();
    Calendar c = builder.build(new UnfoldingReader(new FileReader(mainArgs.file.get(0))));
    for (Component comp : (List<Component>)c.getComponents()) {
      System.err.println(comp);
    }
  }
}
```

Location:

Grantt 〉 biz.ganttproject.impex.ical 〉 src 〉 main 〉 java 〉 biz 〉 ganttproject 〉 impex 〉 ical 〉 IcsConverter

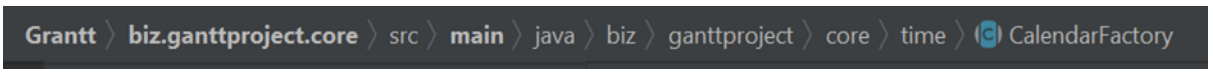
Rationale: This class receives a ics file and converts it into a format our Calendar Class can read so I believe this is an Adaptor Design pattern.

Design Pattern 2: Factory Method Pattern



```
/.../
package biz.ganttproject.core.time;

import ...

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

  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); }
}
```

Location:



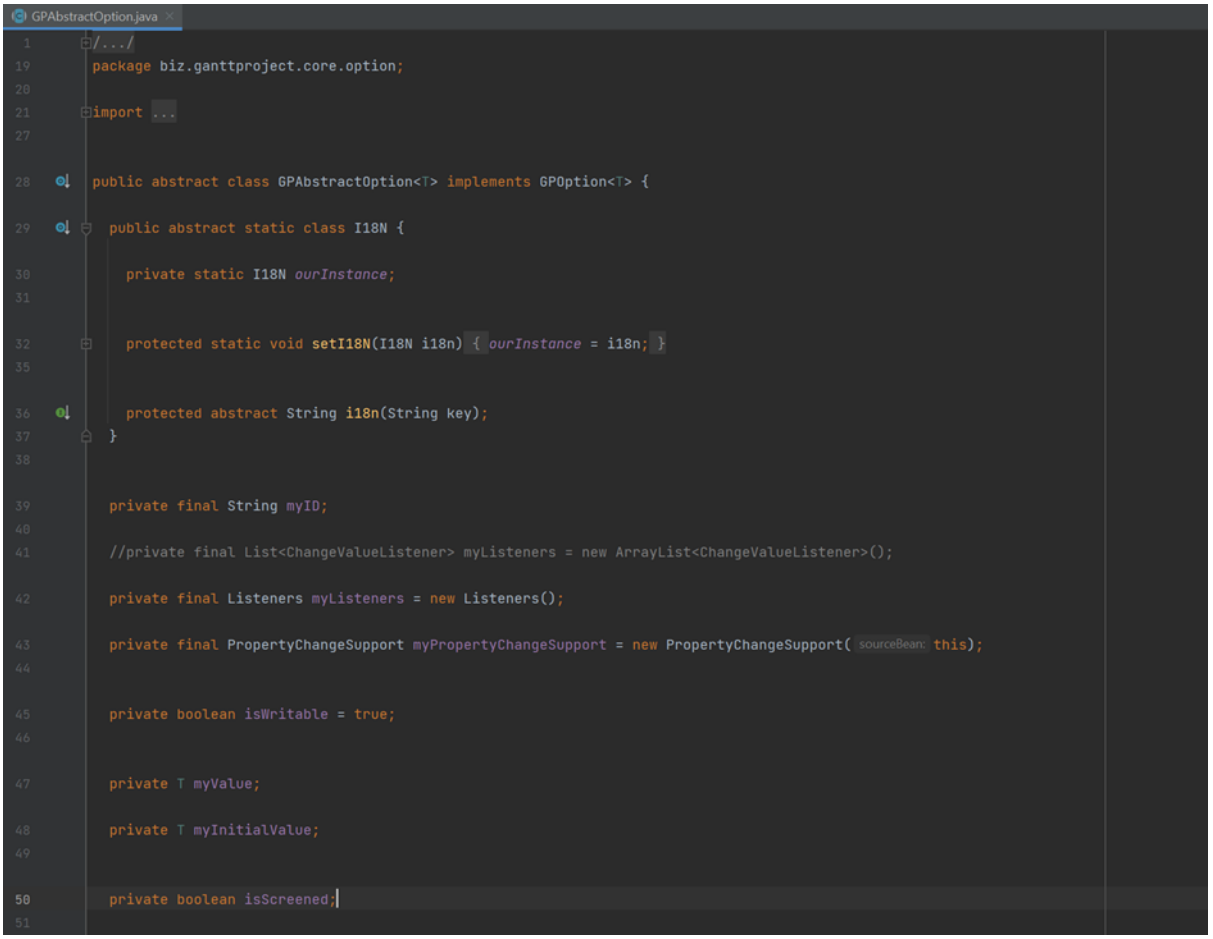
Rationale: This class's purpose is to hide the creation of instances of Calendars behind an abstract class "CalendarFactory". Therefore, I believe this is a Factory Method design Pattern.

Design Pattern 3: Decorator Pattern

GPOption.java | UIFacadeImpl.java | ExporterToMsProjectFile.java

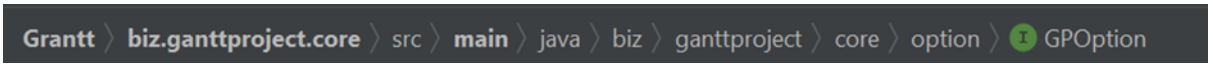
```java
/.../
package biz.ganttproject.core.option;

import java.beans.PropertyChangeListener;

/**
 * @author bard
 */
73 implementations  dbarashev +1
public interface GPOption<T> {
    3 implementations  dbarashev
    T getValue();

    14 implementations  dbarashev
    void setValue(T value);
    2 implementations  dbarashev
    void setValue(T value, Object clientId);

    1 implementation  dbarashev
    String getID();

    1 implementation  dbarashev
    void lock();

    23 implementations  dbarashev
    void commit();

    1 implementation  dbarashev
    void rollback();

    16 implementations  dbarashev
    String getPersistentValue();

    17 implementations  dbarashev
    void loadPersistentValue(String value);

    2 implementations  dbarashev
    boolean isChanged();

    1 implementation  dbarashev
    Runnable addChangeValueListener(ChangeValueListener listener);
```

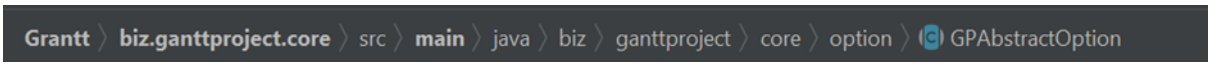
This is the Interface



This is the Decorator Class

Location:





Rationale: There are 73 implementations of the Interface GPOption, each one using a different subclass of the GPAbstractOption (which implements the GPOption Interface) and most of these subclasses override certain methods, adding their own touch to it, So I believe that GPAbstractOption is the Decorator class, and we are in the presence of a Decorator Design Pattern.

Tiago Francisco:

Design Pattern 1: Open/ Closed

Exporter base

```java
/.../
package net.sourceforge.ganttproject.export;

import ...

12 usages  6 inheritors  dbarashev +1
public abstract class ExporterBase implements Exporter {
  2 usages
  private IGanttProject myProject;
  6 usages
  private Chart myGanttChart;
  2 usages
  private Chart myResourceChart;
  2 usages
  private UIFacade myUIFacade;
  6 usages
  private Preferences myRootPreferences;
  6 usages
  private DefaultDateOption myExportRangeStart;
  6 usages
  private DefaultDateOption myExportRangeEnd;

  protected static final GanttLanguage language = GanttLanguage.getInstance();

  1 usage
  static protected Object EXPORT_JOB_FAMILY = "Export job family";

  1 override  Dmitry Barashev +1
  @Override
  public void setContext(IGanttProject project, UIFacade uiFacade, Preferences prefs) {
    myGanttChart = uiFacade.getGanttChart();
    myResourceChart = uiFacade.getResourceChart();
    myProject = project;
    myUIFacade = uiFacade;
    myRootPreferences = prefs;
    Preferences prefNode = prefs.node( s: "/instance/net.sourceforge.ganttproject/export");
    myExportRangeStart = new DefaultDateOption( id: "export.range.start", myGanttChart.getStartDate());
    myExportRangeStart.loadPersistentValue(prefNode.get(
        s: "export-range-start", DateParser.getIsoDate(myGanttChart.getStartDate())));
    myExportRangeStart.addChangeValueListener(event -> {
      prefNode.put( s: "export-range-start", myExportRangeStart.getPersistentValue());
    });
    myExportRangeEnd = new DefaultDateOption( id: "export.range.end", myGanttChart.getEndDate());
    myExportRangeEnd.loadPersistentValue(prefNode.get(
        s: "export-range-end", DateParser.getIsoDate(myGanttChart.getEndDate())));
    myExportRangeEnd.addChangeValueListener(event -> {
      prefNode.put( s: "export-range-end", myExportRangeEnd.getPersistentValue());
    });
```

Location:

ganttproject 〉 **ganttproject** 〉 src 〉 **main** 〉 java 〉 net 〉 sourceforge 〉 ganttproject 〉 export 〉 ExporterBase

ExporterToMsProjectFile

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

import ...

/**
 * @author dbarashev (Dmitry Barashev)
 */
4 usages  dbarashev +1
public class ExporterToMsProjectFile extends ExporterBase {

  10 usages
  private static final String[] FILE_FORMAT_IDS = new String[] { "impex.msproject.fileformat.mpx",
      "impex.msproject.fileformat.mspdi" };

  4 usages
  private static final String[] FILE_EXTENSIONS = new String[] { "mpx", "xml" };

  7 usages
  private String myFileFormat = FILE_FORMAT_IDS[0];

  6 usages  dbarashev
  private EnumerationOption myFileFormatOption = new DefaultEnumerationOption<Object>( id: "impex.msproject.fileformat",
      FILE_FORMAT_IDS) {
    dbarashev
    @Override
    public void commit() {
      super.commit();
      ExporterToMsProjectFile.this.myFileFormat = getValue();
    }
  };

  4 usages
  private LocaleOption myLanguageOption = new LocaleOption();

  2 usages
  private GPOptionGroup myOptions = new GPOptionGroup( id: "exporter.msproject", new GPOption[] { myFileFormatOption });

  2 usages
  private GPOptionGroup myMPXOptions = new GPOptionGroup( id: "exporter.msproject.mpx", new GPOption[] { myLanguageOption });

  dbarashev
  public ExporterToMsProjectFile() {
    myOptions.setTitled(false);
    myMPXOptions.setTitled(false);
    myFileFormatOption.lock();
    myFileFormatOption.setValue(FILE_FORMAT_IDS[0]);
    myFileFormatOption.commit();
```

Location:

ganttproject 〉 biz.ganttproject.impex.msproject2 〉 src 〉 main 〉 java 〉 biz 〉 ganttproject 〉 impex 〉 msproject2 〉 ExporterToMsProjectFile

Rationale: ExporterBase is the abstract class that is closed because it is used in eleven more classes beyond ExporterToMsProjectFile, so if we changed something we would need to change it in twelve different places .The ExporterToMsProjectFile is the class that extends ExporterBase to implement new functionalities avoiding introducing undesirable side effects.

Design Pattern 2: Decorator Design Pattern

```
/.../
package net.sourceforge.ganttproject.export;

import ...

12 usages  6 inheritors  dbarashev +1 *
public abstract class
ExporterBase implements Exporter {
  2 usages
  private IGanttProject myProject;
  6 usages
  private Chart myGanttChart;
  2 usages
  private Chart myResourceChart;
  2 usages
  private UIFacade myUIFacade;
  6 usages
  private Preferences myRootPreferences;
  6 usages
  private DefaultDateOption myExportRangeStart;
  6 usages
  private DefaultDateOption myExportRangeEnd;

  protected static final GanttLanguage language = GanttLanguage.getInstance();

  1 usage
  static protected Object EXPORT_JOB_FAMILY = "Export job family";

  1 override  Dmitry Barashev +1
  @Override
  public void setContext(IGanttProject project, UIFacade uiFacade, Preferences prefs) {
    myGanttChart = uiFacade.getGanttChart();
    myResourceChart = uiFacade.getResourceChart();
    myProject = project;
    myUIFacade = uiFacade;
    myRootPreferences = prefs;
    Preferences prefNode = prefs.node( s: "/instance/net.sourceforge.ganttproject/export");
    myExportRangeStart = new DefaultDateOption( id: "export.range.start", myGanttChart.getStartDate());
    myExportRangeStart.loadPersistentValue(prefNode.get(
        s: "export-range-start", DateParser.getIsoDate(myGanttChart.getStartDate())));
    myExportRangeStart.addChangeValueListener(event -> {
      prefNode.put( s: "export-range-start", myExportRangeStart.getPersistentValue());
    });
    myExportRangeEnd = new DefaultDateOption( id: "export.range.end", myGanttChart.getEndDate());
    myExportRangeEnd.loadPersistentValue(prefNode.get(
        s: "export-range-end", DateParser.getIsoDate(myGanttChart.getEndDate())));
    myExportRangeEnd.addChangeValueListener(event -> {
      prefNode.put( s: "export-range-end", myExportRangeEnd.getPersistentValue());
```

The decorator class

Location:

ganttproject 〉 ganttproject 〉 src 〉 main 〉 java 〉 net 〉 sourceforge 〉 ganttproject 〉 export 〉 ExporterBase

Rationale: The decorator is an abstract class that implements Exporter the component interface that defines the common type for all the classes. The StylesheetExporterBase is a concrete decorator that extends ExporterBase the decorator.

Design Pattern 3: Façade

```
1 implementation  dbarashev
38  static interface TimelineFacade {
      1 usage  1 implementation  dbarashev
39      ScrollingSession createScrollingSession(int xpos, int ypos);
40
      6 usages  1 implementation  dbarashev
41      Date getDateAt(int x);
42
      1 usage  1 implementation  dbarashev
43      TimeDuration createTimeInterval(TimeUnit timeUnit, Date startDate, Date endDate);
44
      1 implementation  dbarashev
45      TimeUnitStack getTimeUnitStack();
46
      1 implementation  dbarashev
47      GPCalendarCalc getCalendar();
48
      1 implementation  dbarashev
49      Date getEndDateAt(int i);
50    }
51  }
```

Location:

ganttproject 〉 ganttproject 〉 src 〉 main 〉 java 〉 net 〉 sourceforge 〉 ganttproject 〉 chart 〉 mouse 〉 MouseInteraction 〉 TimelineFacade

Rationale: Façade provides a unified interface to a complex subsystem and that complexity in this case is hidden in TimelineFacade.

Pedro Gouveia:

Open/Closed

```
public abstract class ChartModelBase implements /* TimeUnitStack.Listener, */ChartModel, TimelineLabelRendererImpl.ChartModelApi {
  10 usages  2 implementations  dbarashev
  public static interface ScrollingSession {
    2 implementations  dbarashev
    void scrollTo(int xpos, int ypos);

    2 implementations  dbarashev
    void finish();
  }
```

Location:

ganttproject 〉 ganttproject 〉 src 〉 main 〉 java 〉 net 〉 sourceforge 〉 ganttproject 〉 chart 〉 ChartModelBase

Rationale: There is an abstract class called *ChartModelBase* to which there are 2 inheritors implementations called *ChartModelImpl* and *ChartModelResource* that are open to change

and the abstract class is closed because methods in the abstract superclass are preserved, and the system can be extended by providing different implementations for each method in the subclasses.

Observer

```
public ActivityOnNodePertChart() {
  setBackground(Color.WHITE.brighter());

  dbarashev +1
  this.addMouseMotionListener(new MouseMotionListener() {
    dbarashev +1
    @Override
    public void mouseDragged(final MouseEvent e) {
      if (myPressedGraphicalNode != null) {
        myPressedGraphicalNode.x = e.getX() - myXClickedOffset;
        myPressedGraphicalNode.y = e.getY() - myYClickedOffset;
        if (e.getX() > getPreferredSize().getWidth()) {
          ActivityOnNodePertChart.this.setPreferredSize(new Dimension( width: myPressedGraphicalNode.x + getNodeWidth() + getxGap(),
              (int) getPreferredSize().getHeight()));
          revalidate();
        }
        if (e.getY() > getPreferredSize().getHeight()) {
          ActivityOnNodePertChart.this.setPreferredSize(new Dimension((int) getPreferredSize().getWidth(),
              height: myPressedGraphicalNode.y + getNodeHeight() + getyGap()));
          revalidate();
        }
        repaint();
      }
    }

    dbarashev
    @Override
    public void mouseMoved(MouseEvent e) {
      // nothing to do...
    }
  });

  dbarashev +1
  this.addMouseListener(new MouseListener() {
    dbarashev
    @Override
    public void mouseClicked(MouseEvent arg0) {
      // nothing to do...
    }
```

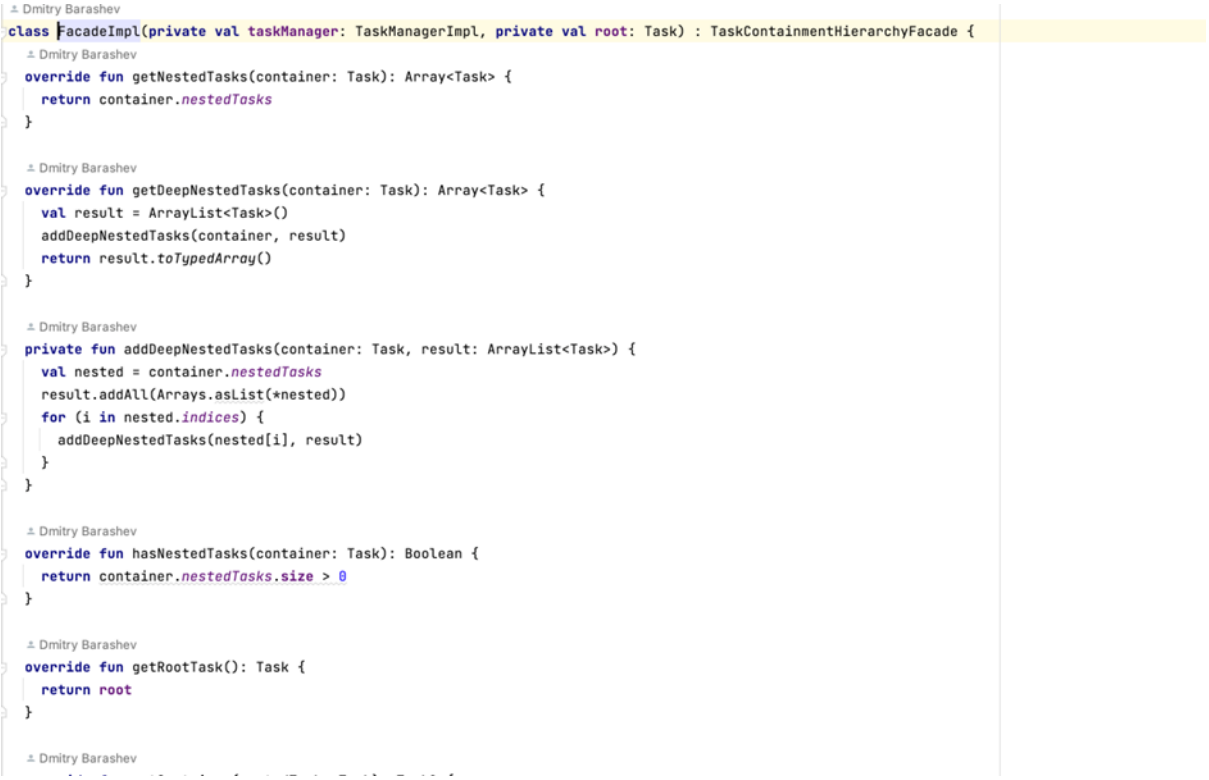
Location:

ganttproject 〉 org.ganttproject.chart.pert 〉 src 〉 main 〉 java 〉 org 〉 ganttproject 〉 chart 〉 pert 〉 C ActivityOnNodePertChart

Rationale:

This class methods are called whenever there are some actions on a node of the Pert Chart, changing some data in the main app. These actions can be a mouse drag, a mouse move, a mouse click, etc.

Façade



```
Dmitry Barashev
class FacadeImpl(private val taskManager: TaskManagerImpl, private val root: Task) : TaskContainmentHierarchyFacade {
  Dmitry Barashev
  override fun getNestedTasks(container: Task): Array<Task> {
    return container.nestedTasks
  }

  Dmitry Barashev
  override fun getDeepNestedTasks(container: Task): Array<Task> {
    val result = ArrayList<Task>()
    addDeepNestedTasks(container, result)
    return result.toTypedArray()
  }

  Dmitry Barashev
  private fun addDeepNestedTasks(container: Task, result: ArrayList<Task>) {
    val nested = container.nestedTasks
    result.addAll(Arrays.asList(*nested))
    for (i in nested.indices) {
      addDeepNestedTasks(nested[i], result)
    }
  }

  Dmitry Barashev
  override fun hasNestedTasks(container: Task): Boolean {
    return container.nestedTasks.size > 0
  }

  Dmitry Barashev
  override fun getRootTask(): Task {
    return root
  }

  Dmitry Barashev
```

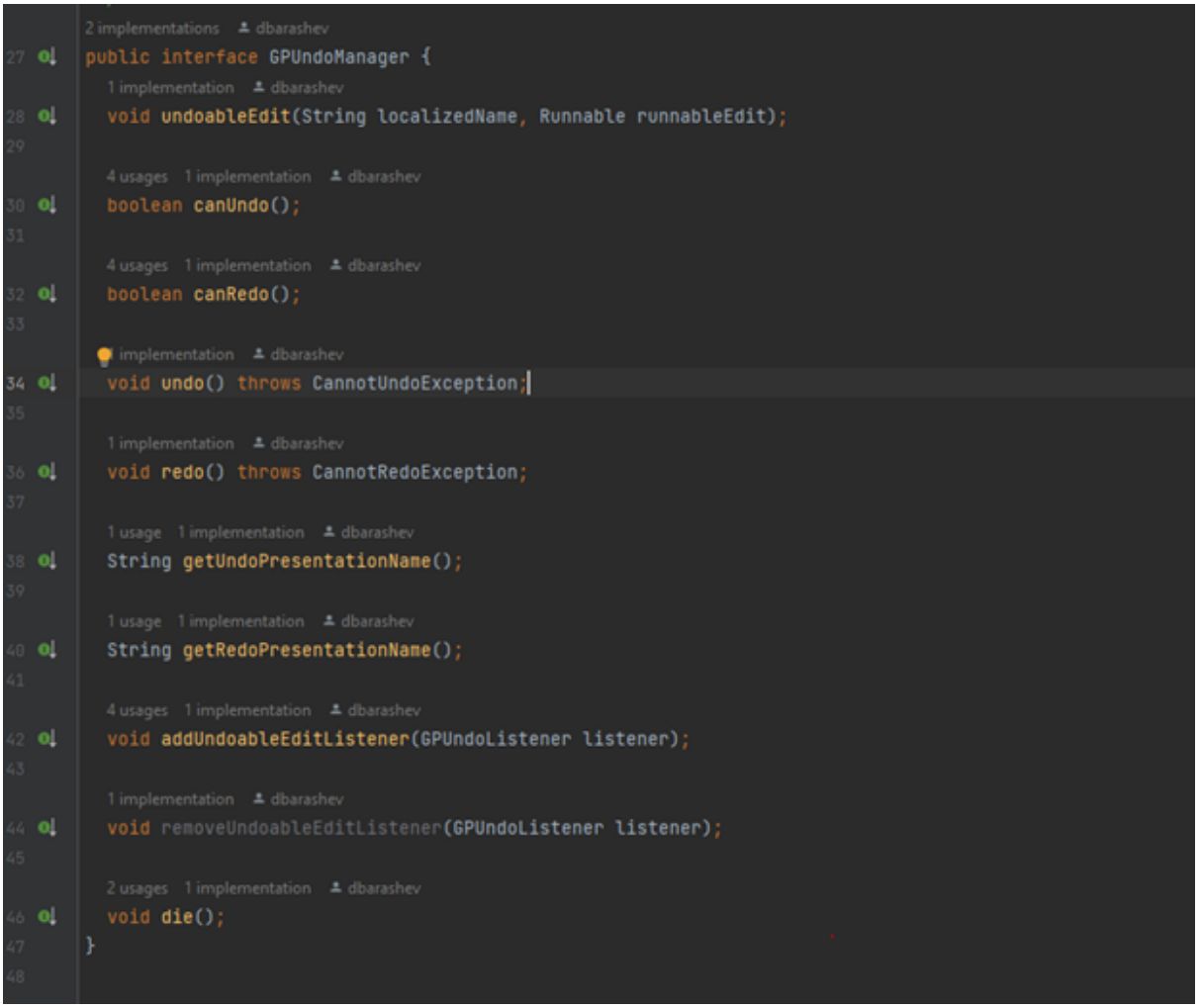
Location:

Rationale: This façade serves as a way for the user to interact with the task hierarchy tree hiding more complex functionalities like interacting with left and right nodes, iterating the tree and so on from the other classes. It saves the user the task and trouble to make a class to interact with the Task Hierarchy tree.

Luís Abreu:

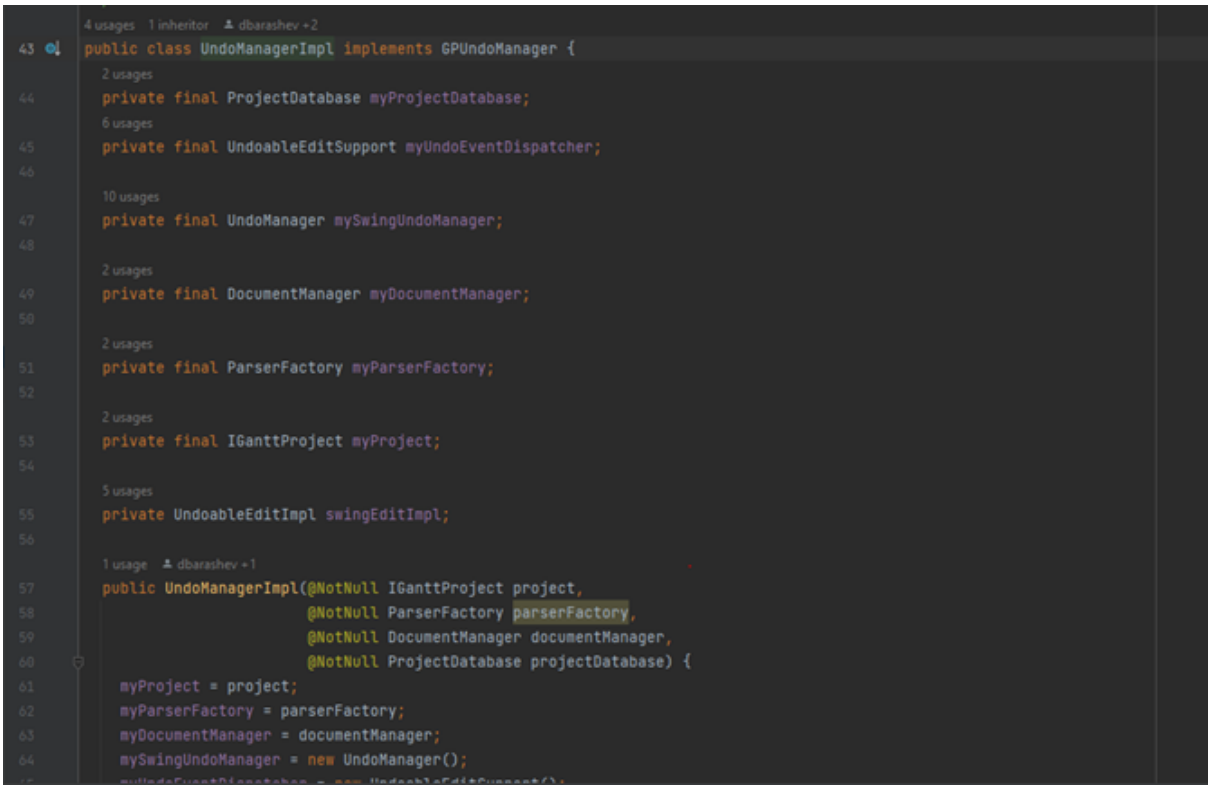
Design Pattern 1: Memento Pattern

This is the interface:



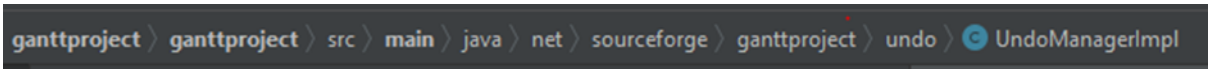
```
2 implementations  dbarashev
public interface GPUndoManager {
    1 implementation  dbarashev
    void undoableEdit(String localizedName, Runnable runnableEdit);

    4 usages  1 implementation  dbarashev
    boolean canUndo();

    4 usages  1 implementation  dbarashev
    boolean canRedo();

    implementation  dbarashev
    void undo() throws CannotUndoException;

    1 implementation  dbarashev
    void redo() throws CannotRedoException;

    1 usage  1 implementation  dbarashev
    String getUndoPresentationName();

    1 usage  1 implementation  dbarashev
    String getRedoPresentationName();

    4 usages  1 implementation  dbarashev
    void addUndoableEditListener(GPUndoListener listener);

    1 implementation  dbarashev
    void removeUndoableEditListener(GPUndoListener listener);

    2 usages  1 implementation  dbarashev
    void die();
}
```

This is the class that implements the interface:



```
4 usages  1 inheritor  dbarashev +2
public class UndoManagerImpl implements GPUndoManager {
  2 usages
  private final ProjectDatabase myProjectDatabase;
  6 usages
  private final UndoableEditSupport myUndoEventDispatcher;

  10 usages
  private final UndoManager mySwingUndoManager;

  2 usages
  private final DocumentManager myDocumentManager;

  2 usages
  private final ParserFactory myParserFactory;

  2 usages
  private final IGanttProject myProject;

  5 usages
  private final UndoableEditImpl swingEditImpl;

  1 usage  dbarashev +1
  public UndoManagerImpl(@NotNull IGanttProject project,
                         @NotNull ParserFactory parserFactory,
                         @NotNull DocumentManager documentManager,
                         @NotNull ProjectDatabase projectDatabase) {
    myProject = project;
    myParserFactory = parserFactory;
    myDocumentManager = documentManager;
    mySwingUndoManager = new UndoManager();
```

Location:



Rationale: This class allows to return an object to one of its previous states. In this case, our class extends an interface that has methods (*canUndo()* and *undo()*) that allow the object to check if it has saved a previous state and if that is the case, it can return to them.

Design Pattern 2: Factory Method Pattern

```
*/
3 usages  dbarashev +1
public class IndentTargetFunctionFactory implements Function<Collection<Task>, Function<Task, Task>> {
    2 usages
    private final TaskManager myTaskManager;
    3 usages  dbarashev +1
    public IndentTargetFunctionFactory(TaskManager taskManager) { myTaskManager = taskManager; }

    dbarashev
    @Override
    public Function<Task, Task> apply(final Collection<Task> indentRoots) {
        dbarashev
        return new Function<Task, Task>() {
            1 usage
            private final TaskContainmentHierarchyFacade myTaskHierarchy = myTaskManager.getTaskHierarchy();

            dbarashev
            @Override
            public Task apply(Task whatMove) {
                Task moveTarget = whatMove;
                for (; moveTarget != null && indentRoots.contains(moveTarget);
                    moveTarget = myTaskHierarchy.getPreviousSibling(moveTarget));
                return moveTarget;
            }
        };
    }
```

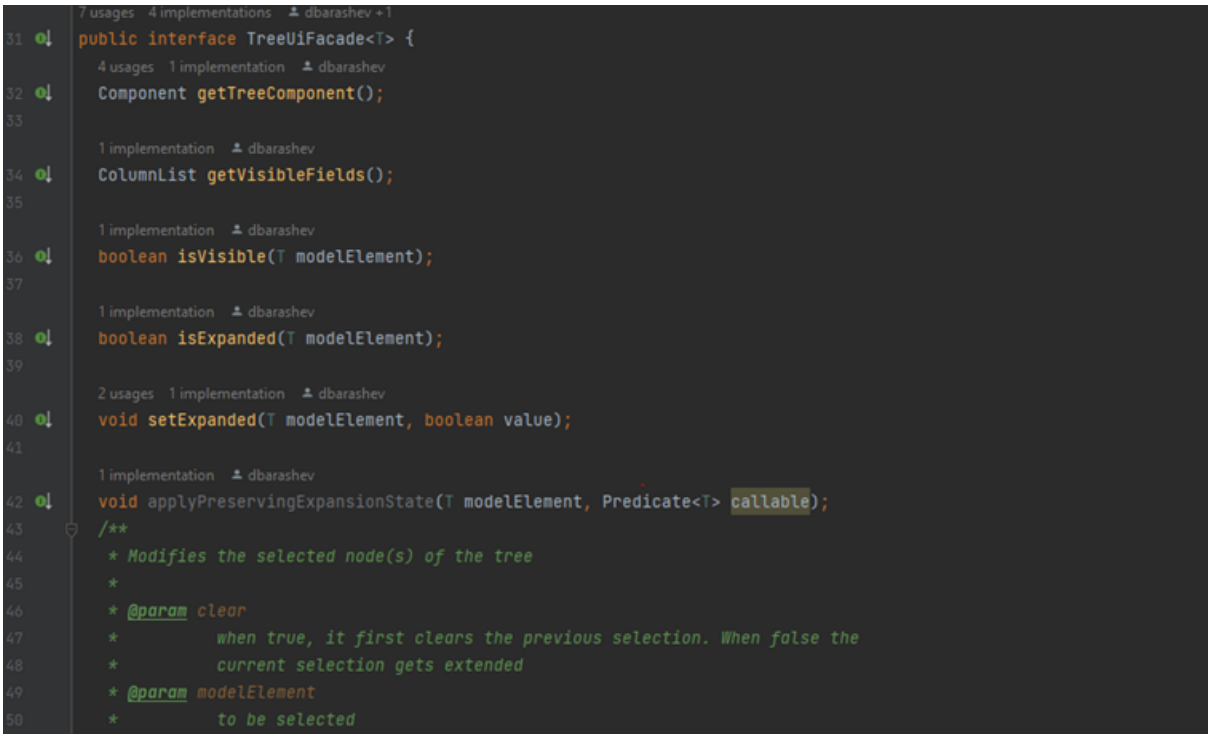
Location:

ganttproject 〉 ganttproject 〉 src 〉 main 〉 java 〉 net 〉 sourceforge 〉 ganttproject 〉 action 〉 task 〉 IndentTargetFunctionFactory

Rationale: This class's purpose is to hide the creation of instances of a given type

Design Pattern 3: Facade Pattern

This is the inferface:

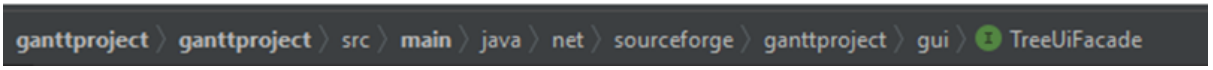


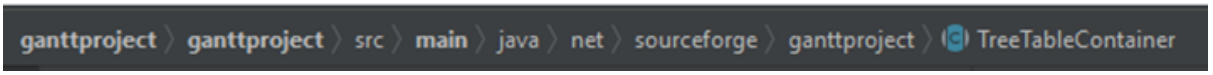
```
7 usages  4 implementations  dbarashev +1
public interface TreeUiFacade<T> {
    4 usages  1 implementation  dbarashev
    Component getTreeComponent();

    1 implementation  dbarashev
    ColumnList getVisibleFields();

    1 implementation  dbarashev
    boolean isVisible(T modelElement);

    1 implementation  dbarashev
    boolean isExpanded(T modelElement);

    2 usages  1 implementation  dbarashev
    void setExpanded(T modelElement, boolean value);

    1 implementation  dbarashev
    void applyPreservingExpansionState(T modelElement, Predicate<T> callable);
    /**
     * Modifies the selected node(s) of the tree
     *
     * @param clear
     *          when true, it first clears the previous selection. When false the
     *          current selection gets extended
     * @param modelElement
     *          to be selected
```

This is the class that implements the interface:

```
3 usages  1 inheritor  dbarashev +3
public abstract class TreeTableContainer<ModelObject, TreeTableClass extends GPTreeTableBase, TreeTableModelClass extends DefaultTreeTableModel>
    extends JPanel implements TreeUiFacade<ModelObject> {
  16 usages
  private final TreeTableClass myTreeTable;
  2 usages
  private final TreeTableModelClass myTreeTableModel;
  3 usages
  private GPAction myNewAction;
  3 usages
  private GPAction myPropertiesAction;
  2 usages
  private GPAction myDeleteAction;

  1 usage  dbarashev
  private class ExpandCollapseAction extends GPAction {
    1 usage  dbarashev
    ExpandCollapseAction() { super( name: "tree.expand"); }

    dbarashev
    @Override
    public void actionPerformed(ActionEvent e) {
      TreePath currentSelection = getTree().getTreeSelectionModel().getSelectionPath();
      if (currentSelection != null) {
        if (getTree().isCollapsed(currentSelection)) {
          getTree().expandPath(currentSelection);
        } else {
          getTree().collapsePath(currentSelection);
        }
      }
    }
  }

  1 usage  dbarashev +1
```

Location:





Rationale: A façade is a wrapper class that encapsulates a subsystem in order to hide the subsystem complexity. The intent is to hide complexity behind an interface. In this case, we hide the complexity in the TreeUiFacade.