GANTT PROJECT

Design Patterns & Code Smell Identification

S TEAM

Ana Antunes – *61025*

Joana Wang – *60225*

Nelson Matos – *60483*

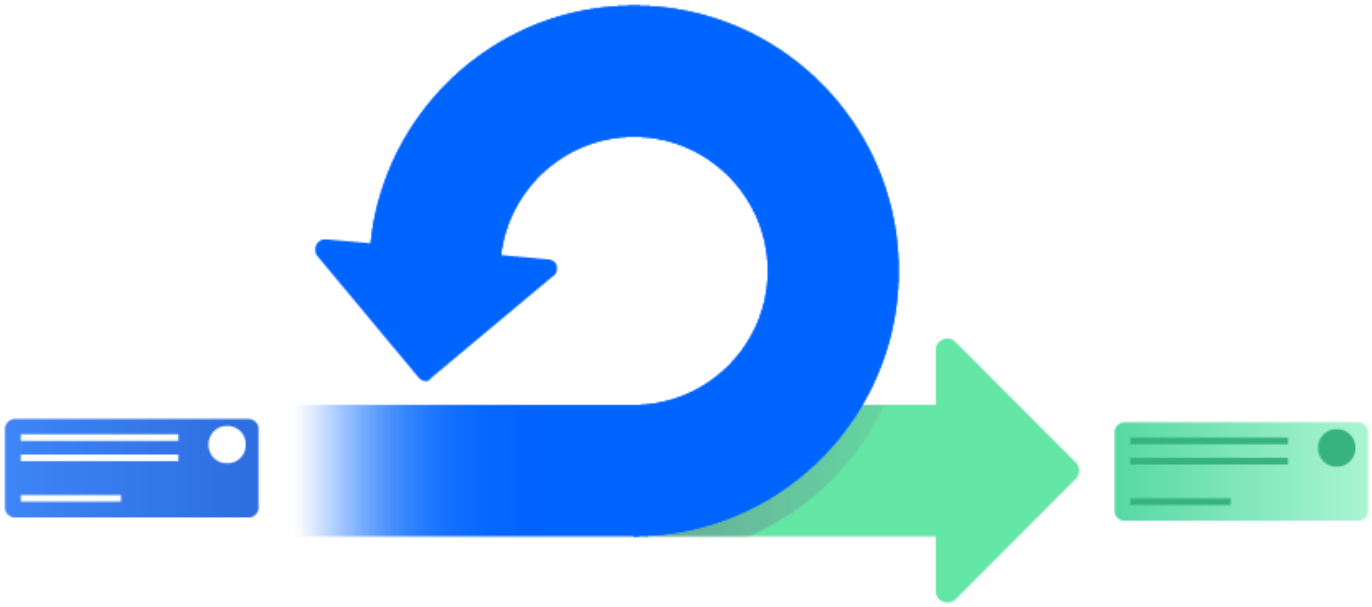
Pedro Estróia – *60691*

Renato Viola – *60665*

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Índice

NOVA SCHOOL OF SCIENCE & TECHNOLOGY



ANA'S DESIGN PATTERNS

1.1.1. SINGLETON

Location:
ganttproject\src\main\java\net\sourceforge\ganttproject\language

Class: GanttLanguage.java

Code Snippet:

//Private Object

```
private static final GanttLanguage ganttLanguage = new GanttLanguage();
```

//Create only one instance

```
public static GanttLanguage getInstance() { return ganttLanguage; }
```

//Private Constructor

```
private GanttLanguage() {
  dbarashev
  new GPAbstractOption.I18N() {
    {
      setI18N(this);
    }
    dbarashev
    @Override
    protected String i18n(String key) { return getText(key); }
  };
  Properties charsets = new Properties();
  PropertiesUtil.loadProperties(charsets, resource: "/charsets.properties");
  myCharSetMap = new CharSetMap(charsets);
  setLocale(Locale.getDefault());
  PropertiesUtil.loadProperties(myExtraLocales, resource: "/language/extra.properties");
}
```

Explanation: This class, "GanttLanguage" has only one instance, that can be globally called with getInstance() method.

The chosen code examples illustrate very well the Singleton design pattern and the way it is being used to restrict the class to one instance only.However, the explanation was too simple, and does not refer the instance variable that creates the class object nor the private constructor, lacking a better connection between the explanation and the snippets. – ***Reviewed by Pedro***

1.1.2. OBSERVER

Location:

ganttproject\src\main\java\net\sourceforge\ganttproject\gui\scrolling

Class: ScrollingManagerImpl.java

Code Snippet:

//Class that will be "observed" when events happen

```
public class ScrollingManagerImpl implements ScrollingManager
```

// This method add a Listener to notify the Observer("ScrollingListener")

```
public void addScrollingListener(ScrollingListener listener) { myListeners.add(listener); }
```

//Interface that will be notified, when "addScrollingListener" is called

```
public interface ScrollingListener
```

Explanation: Notify the object "ScrollingListener" about events that happen to "Scrolling Manager"

The explanation could be elaborated upon further and worded a bit better. The method to add a listener does not state that it is notified when being created, but rather that there might be some sort of callback mechanism to notify it when already "subscribed" to the subject. That aside, the core concept of the observer pattern was grasped. – ***Reviewed by Renato***

1.1.3. STRATEGY

Location:

ganttproject\src\main\java\net\sourceforge\ganttproject\task\algorithm

Class: CriticalPathAlgorithm.java

Code Snippet:

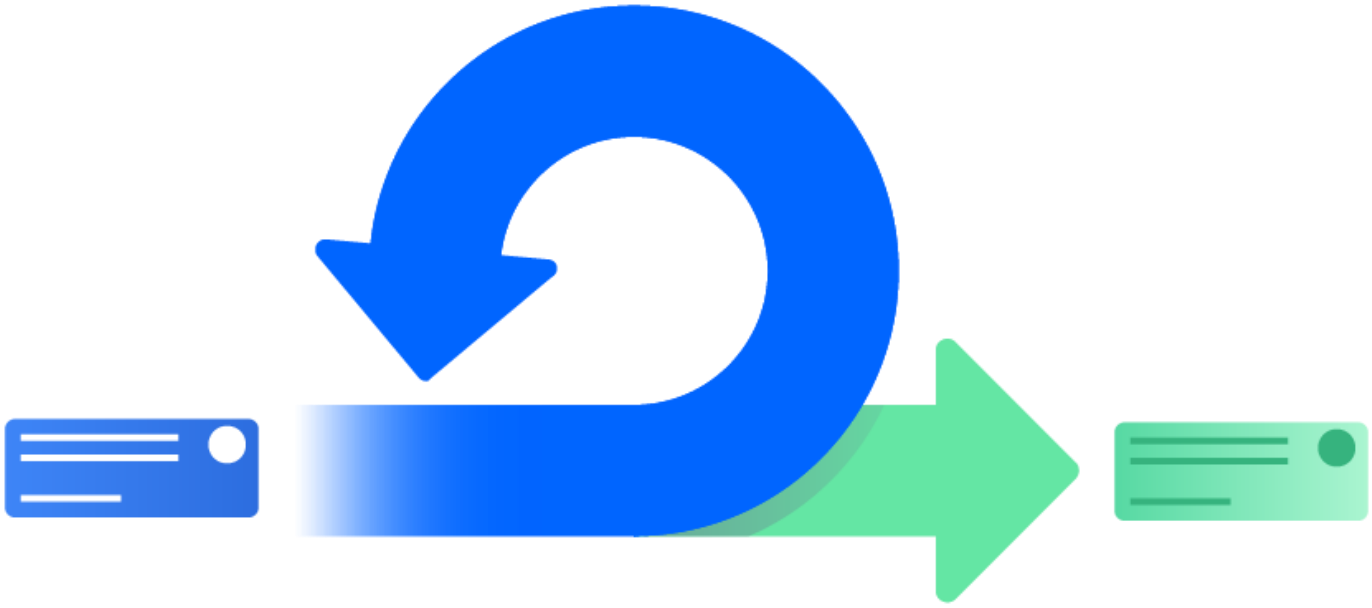
//Strategy Interface

```
public interface CriticalPathAlgorithm {
  /**
   * @return All tasks which are included in any critical path (if there are
   *         many of them).
   */
  12 usages  1 implementation  Dmitry Barashev
  Task[] getCriticalTasks();
  1 implementation  Dmitry Barashev
  boolean isEnabled();
  1 implementation  Dmitry Barashev
  void setEnabled(boolean value);
}
```

Explanation: With the "Strategy object" above we can have a family of different search Algorithm implementations. This way, we can add new algorithms or modify existing ones without changing the code of these classes.

The explanation was a bit confusing. We mainly use this design pattern when we find or want an algotithm that's prone to reocurrent changes, the interface purpose is to declare all the common methods shared amongst the algorithms whilst each of them is implemented in a different class. This way we prevent the occurence of unnecessarily complex and large classes. – ***Reviewed by Nelson***

NOVA SCHOOL OF SCIENCE & TECHNOLOGY



JOANA'S DESIGN PATTERNS

1.2.1. COMMAND

Code snippet:

```
public abstract class GPAction extends AbstractAction implements GanttLanguage.Listener, EventHandler<javafx.event.ActionEvent> {
  private final String myIconSize;
  private String myFontAwesomeLabel;
```

Location:

Ganttproject>src>main>java>net>sourceforge>ganttproject>action>edit

>EditMenu.java

Explanation:

A request is encapsulated as an object, which serves as a command, invoked by the invoker object to complete tasks that it supposed to do.

The chosen code snippet could've been a little more informative of the command pattern and does not complement the explanation very well. Even though the actual explanation is correct, it could've benefited from being more specific and related to the code example. – ***Reviewed by Pedro***

1.2.2. SINGLETON

Code snippet:

```
private static GPCalendarProvider ourInstance;
```

```
public static synchronized GPCalendarProvider getInstance() {
  if (ourInstance == null) {
    List<GPCalendar> calendars = readCalendars();
    Collections.sort(calendars, new Comparator<GPCalendar>() {
      public int compare(GPCalendar o1, GPCalendar o2) {
        return o1.getName().compareTo(o2.getName());
      }
    });
    ourInstance = new GPCalendarProvider(calendars);
  }
  return ourInstance;
}
```

Location:

ganttproject>src>main>java>net>sourceforge>ganttproject>calender>

GPCalendarProvider.java

Explanation:

In this pattern, a class is restricted to having only one instance. Thus, the instance is stored in a private static variable and a static method is provided that returns a reference to the instance.

Despite unmentioned in the explanation, we are able to observe that its constructor is private which is indicative of a Singleton pattern after further analysing the GPCalendarProvider. – ***Reviewed by Nelson***

1.2.3. TEMPLATE METHOD

Code snippet:

```
abstract class ResourceAction extends GPAction implements ActionDelegate {
public class ResourceNewAction extends ResourceAction {
public class ResourceDeleteAction extends ResourceAction {
public class ResourcePropertiesAction extends ResourceAction {
```

Location:

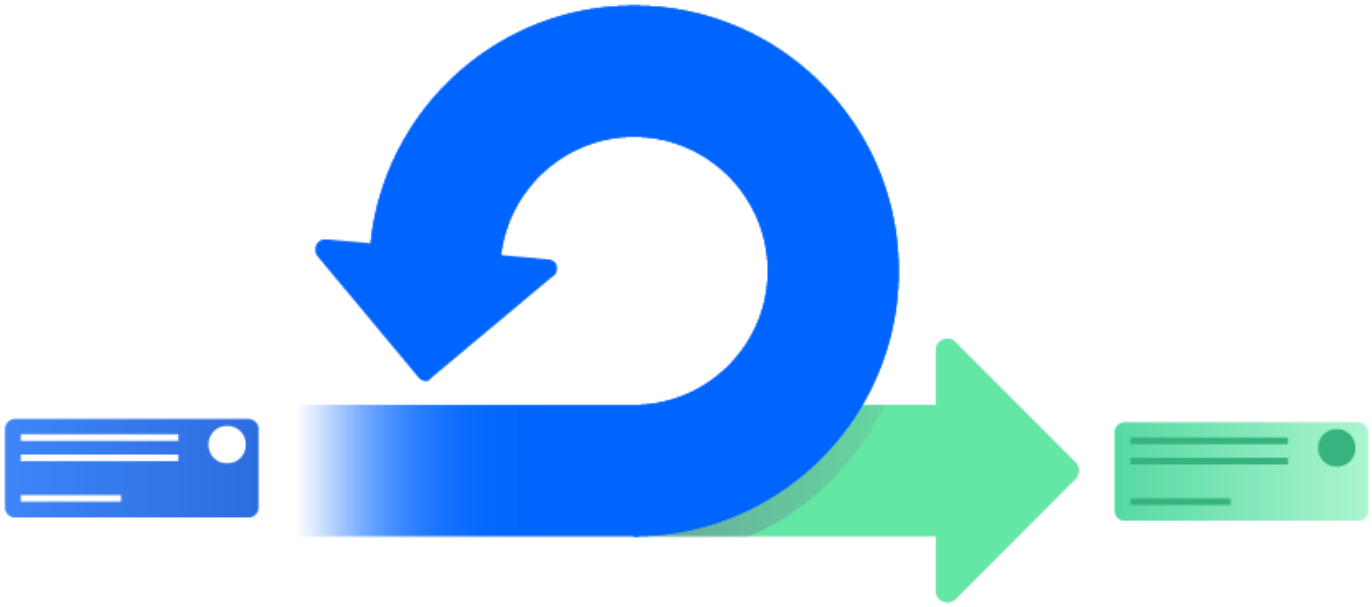
- ganttproject>src>main>java>net>sourceforge>ganttproject>action>resource>ResourceAction.java
- ganttproject>src>main>java>net>sourceforge>ganttproject>action>resource>ResourceNewAction.java
- ganttproject>src>main>java>net>sourceforge>ganttproject>action>resource>ResourceDeleteAction.java
- ganttproject>src>main>java>net>sourceforge>ganttproject>action>resource>ResourcePropertiesAction.java

Explanation:

This method design pattern is to define an algorithm in general, leaving the implementation details to its subclasses. In this specific case, the *ResourceAction.java* is an abstract class, extended by three other subclasses, where each one implements their own abstract method.

The explanation could've been worded a bit better, in order to avoid misunderstanding it. To say this pattern leaves an algorithm's implementation details to its subclasses can be interpreted as a bit of a generalization, since not all the steps of the algorithm necessarily need to be handled in its subclasses, only some. That said, it is still a fairly correct explanation. – ***Reviewed by Renato***

NOVA SCHOOL OF SCIENCE & TECHNOLOGY



NELSON'S DESIGN PATTERNS

1.3.1. MEMENTO

Code Snippet:

```
private Document saveFile() throws IOException {
  Document doc = myManager.getDocumentManager().newAutosaveDocument();
  doc.write();
  return doc;
}

private void restoreDocument(Document document) throws IOException,
DocumentException {
  myManager.getProject().restore(document);
}
```

Exact location:

- saveFile() -> ganttproject/src/main/java/net.sourceforge.ganttproject/undo/UndoableEditImpl (line 60)
- restoreDocument() -> ganttproject/src/main/java/net.sourceforge.ganttproject/undo/UndoableEditImpl (line 108)

Explanation:

This is an usual and multi-used design pattern in the software engineering companies. Nowadays most of the productivity applications such as excel,

word, etc., let the user save its current work so it can be accessed in the future.
The two methods above express these actions.

Simple description and well defined idea with the code snippet above. The explanation could be a little more complete, adding the fact that the "UndoableEditImpl" is storing a copy of the previous work state, so, this way, the project can be restored in the future(the restored work can be acessed by "UndoableEditImpl"). – ***Reviewed by Ana***

1.3.2. PROXY

Code Snippet:

```
/**
 * Gets the username used to authenticate to the storage container
 *
 * @return username
 */
public String getUsername();

/**
 * Gets the password used to authenticate to the storage container
 *
 * @return password
 */
public String getPassword();
```

Exact location:

- getUsername() -> ganttproject/src/main/java/net.sourceforge.ganttproject/document/Document (line 122)
- getPassword() -> ganttproject/src/main/java/net.sourceforge.ganttproject/document/Document (line 129)

Explanation:

This design pattern is usually used when we need to make a bridge between a client and the real subject at hand. This “bridge” (proxy) is helpful because it can perform somewhat like the real subject but in a simplified way. In this case we can see that the proxy controls access of the real document as in, the user needs to specify its name and password to have access to it.

Good explanation, explicit to everyone. The ilustrating image isn´t the best matching with the description. Probably showing the control class that gives the "Documents" acess and a little discription of the image would help to understand this pattern. – ***Reviewed by Ana***

1.3.3. ITERATOR

Code Snippet:

```
private DefaultMutableTreeTableNode buildTree() {

  DefaultMutableTreeTableNode root = new DefaultMutableTreeTableNode();
  List<HumanResource> listResources = myResourceManager.getResources();
  Iterator<HumanResource> itRes = listResources.iterator();

  while (itRes.hasNext()) {
    HumanResource hr = itRes.next();
    ResourceNode rnRes = new ResourceNode(hr); // the first for the resource
    root.add(rnRes);
  }
  return root;
}
```

Exact location:

- buildTree() -> ganttproject/src/main/java/net.sourceforge.ganttproject/ResourceTreeTableModel (line 112)

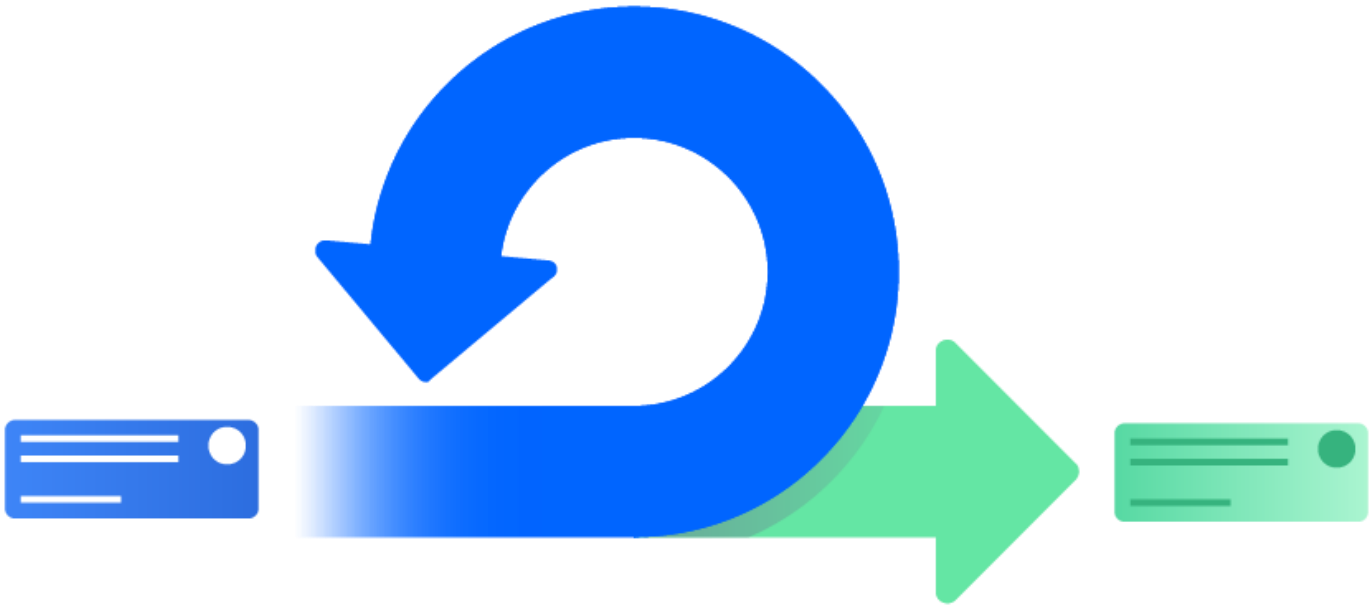
Explanation:

This method makes use of an iterator that helps us traverse a collection of objects without exposing its underlying representation.

The design pattern is relatively simple, since the iterator is extremely used (doing the same as 'for/while' cycles, but without accessing object representations), so the explanation is concise and complete.

*– **Reviewed by Joana***

NOVA SCHOOL OF SCIENCE & TECHNOLOGY



PEDRO'S DESIGN PATTERNS

1.4.1. ITERATOR

Location: ganttproject/src/main/java/net/sourceforge/ganttproject/task

Class: CustomColumnsStorage.java

Code Snippet:

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

Explanation: This method uses an Iterator to traverse a collection of objects of the CustomPropertyListener class, without exposing its internal properties.

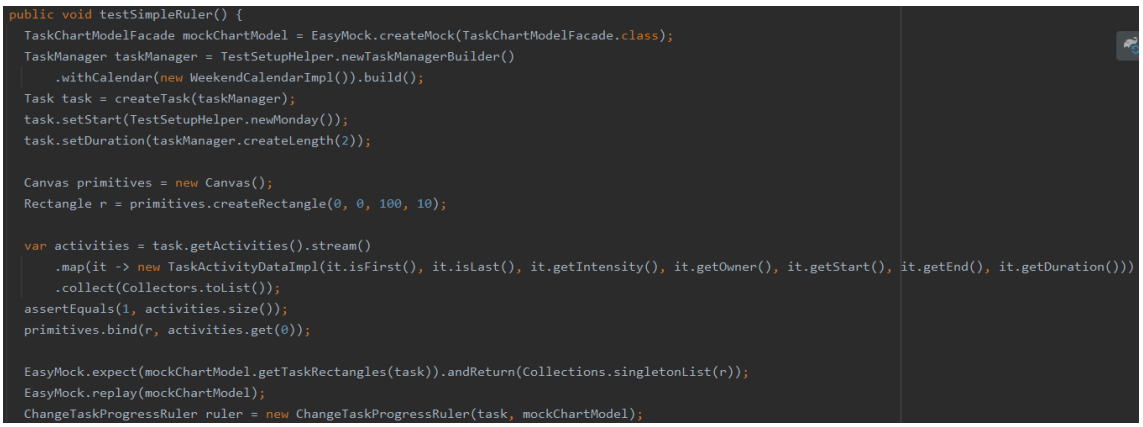
This design pattern is pretty straightforward and easy to understand so there's harshly anything to add to the explanation. – ***Reviewed by Nelson***

1.4.2. FACADE

Location: ganttproject-tester/test/net/sourceforge/ganttproject/chart/mouse

Class: ChangeTaskProgressRulerTest.java

Code Snippet:



```
public void testSimpleRuler() {
  TaskChartModelFacade mockChartModel = EasyMock.createMock(TaskChartModelFacade.class);
  TaskManager taskManager = TestSetupHelper.newTaskManagerBuilder()
      .withCalendar(new WeekendCalendarImpl()).build();
  Task task = createTask(taskManager);
  task.setStart(TestSetupHelper.newMonday());
  task.setDuration(taskManager.createLength(2));

  Canvas primitives = new Canvas();
  Rectangle r = primitives.createRectangle(0, 0, 100, 10);

  var activities = task.getActivities().stream()
      .map(it -> new TaskActivityDataImpl(it.isFirst(), it.isLast(), it.getIntensity(), it.getOwner(), it.getStart(), it.getEnd(), it.getDuration()))
      .collect(Collectors.toList());
  assertEquals(1, activities.size());
  primitives.bind(r, activities.get(0));

  EasyMock.expect(mockChartModel.getTaskRectangles(task)).andReturn(Collections.singletonList(r));
  EasyMock.replay(mockChartModel);
  ChangeTaskProgressRuler ruler = new ChangeTaskProgressRuler(task, mockChartModel);
```

Explanation: In this class, we can see the use of the Facade pattern when creating a TaskChartModelFacade object, with the purpose of accessing the subsystems of the mockChartModels easily and hiding their complexity.

The use of the facade pattern could've been explained in a more complete manner, such as mentioning how this pattern is intended to minimize dependencies between subsystems. Yet, the explanation was accurate and concisive overall. – ***Reviewed by Renato***

1.4.3. SINGLETON

Location: ganttproject/src/main/java/net/sourceforge/ganttproject/gui

Class: GanttLooksAndFeels.java

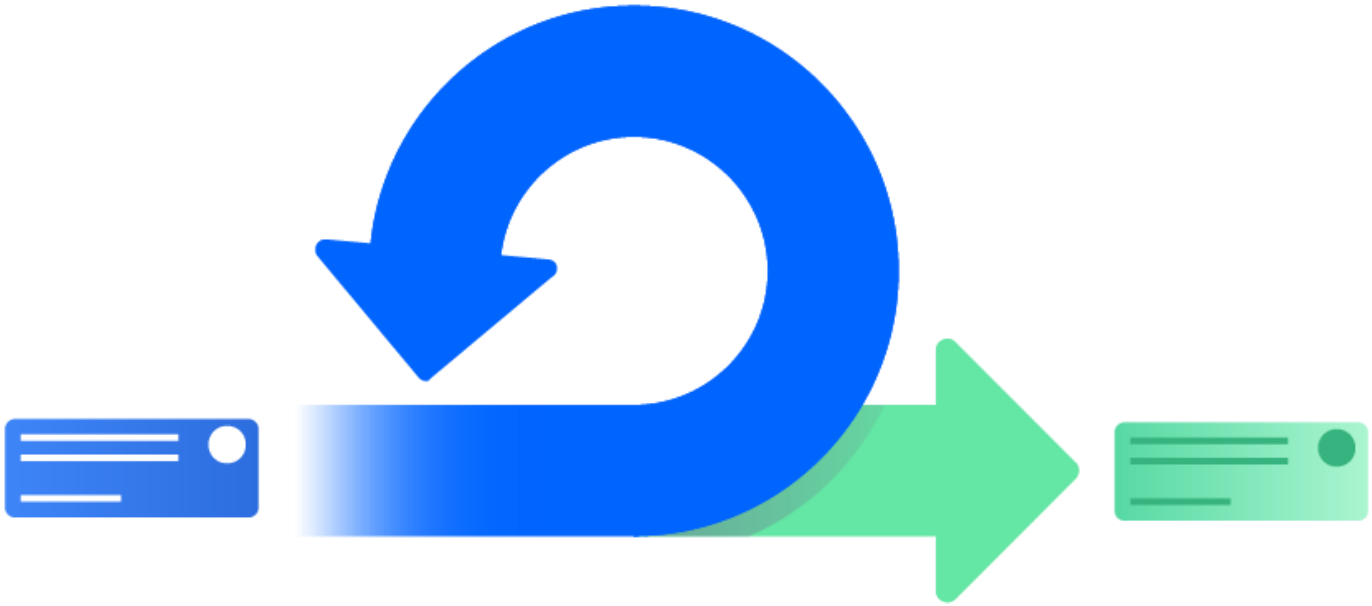
Code Snippet:

```
public static GanttLookAndFeels getGanttLookAndFeels() {
  if (singleton == null) {
    singleton = new GanttLookAndFeels();
  }
  return singleton;
}
```

Explanation: This class makes use of the Singleton pattern, by allowing the class to be instantiated only once, being the created object in this method the only global access point to this class.

Good and clean explanation. The only detail would be a more specific explanation to this concrete example, so that everyone could understand better how it´s done there. – ***Reviewed by Ana***

NOVA SCHOOL OF SCIENCE & TECHNOLOGY



RENATO'S DESIGN PATTERNS

1.5.1. CHAIN OF RESPONSIBILITY

Location:
ganttproject/src/main/java/net/sourceforge/ganttproject/parser

Class: ViewTagHandler.java

Code snippet:

```
public class ViewTagHandler extends AbstractTagHandler {
  private final UIFacade myUIFacade;
  private final String myViewId;
  private final TaskDisplayColumnsTagHandler myFieldsHandler;

  public ViewTagHandler(String viewId, UIFacade uiFacade,
TaskDisplayColumnsTagHandler fieldsHandler) {
    super("view");
    myViewId = viewId;
    myFieldsHandler = fieldsHandler;
    myUIFacade = uiFacade;
  }

  @Override
  protected boolean onStartElement(Attributes attrs) {
    if (Objects.equal(myViewId, attrs.getValue("id"))) {
      loadViewState(attrs);
      myFieldsHandler.setEnabled(true);
      return true;
    }
    return false;
  }
```

Explanation: The method *onStartElement()* processes a request and passes it on to another handler class, by enabling the latter.

This code snippet was a good choice for this pattern, describing very well how the created handler works. However, there could also be present a code snippet of the TaskDisplayColumnsTagHandler class, showing how the request is handled. About the explanation, it could be a little bit more descriptive, and could describe briefly how the handler class works. – ***Reviewed by Pedro***

1.5.2. FACADE

Location:
ganttproject/src/main/java/net/sourceforge/ganttproject/export

Class: CommandLineExportAplication.java

Code snippet:

```
private boolean export(Exporter exporter, Args args, IGanttProject
project, UIFacade uiFacade) {
    logger.debug("Using exporter {}", new Object[]{exporter}, new
HashMap<>());
    ConsoleUIFacade consoleUI = new ConsoleUIFacade(uiFacade);
    GPLogger.setUIFacade(consoleUI);
    // TODO: bring back task expanding
//    if (myArgs.expandTasks) {
//      for (Task t : project.getTaskManager().getTasks()) {
//        project.getUIFacade().getTaskTree().setExpanded(t, true);
//      }
//    }
```

Explanation: In its *export()* method, the class CommandLineExportAplication makes use of a facade class named ConsoleUIFacade in order to facilitate the use of the console UI's subsystems.

I don't think it's the exact place to prove this design pattern. However, the explanation of what the facade pattern is for and the representative class example were correct. In the code shown, the method was creating a new object of the ConsoleUIFacade class and then calling a method to update with this new object. Therefore, it is not clear that said class was hiding the complexity of the subsystem. I suggest showing the class itself or the respective interface for this pattern. – ***Reviewed by Joana***

1.5.3. ADAPTER

Location: ganttproject/src/main/java/net/sourceforge/ganttproject/gui/options/model

Class: CustomPropertyDefaultValueAdapter.java

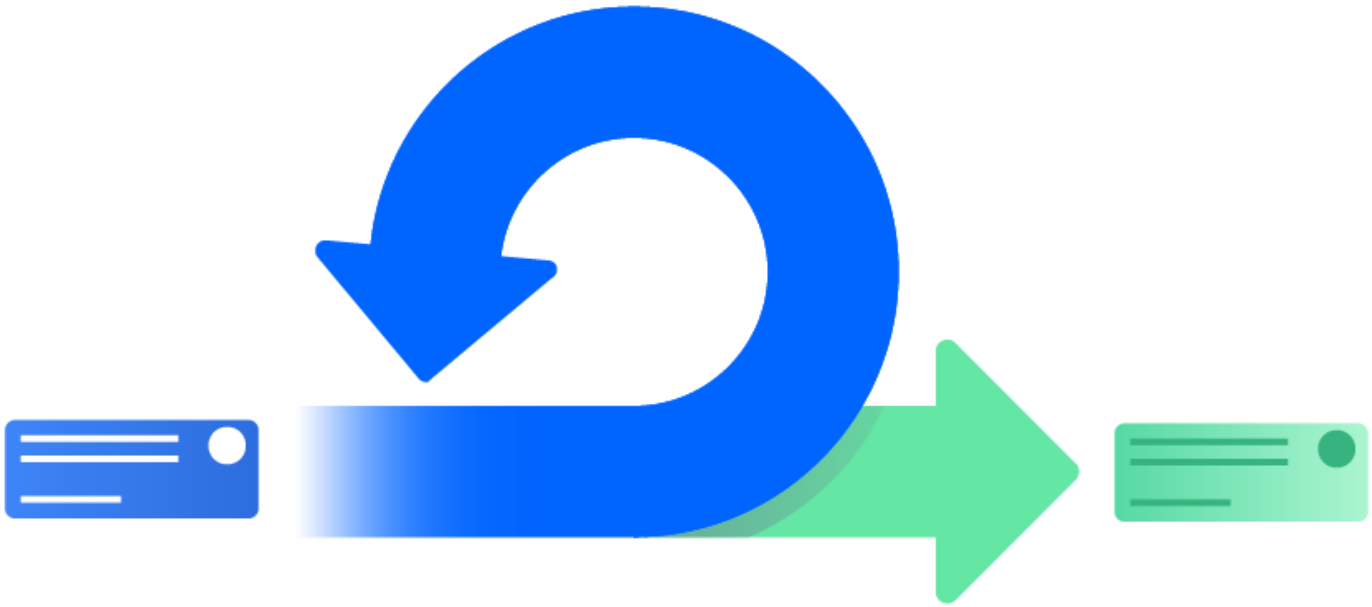
Code snippet:

```
public static GPOption createDefaultValueOption(final
CustomPropertyClass propertyClass,
    final CustomPropertyDefinition def) {
  switch (propertyClass) {
  case TEXT:
    class TextDefaultValue extends DefaultStringOption {
      TextDefaultValue() {
        super("customPropertyDialog.defaultValue.text",
def.getDefaultValueAsString());
      }

      @Override
      public void commit() {
        if (isChanged()) {
          assert propertyClass == def.getPropertyClass();
          super.commit();
          def.setDefaultValueAsString(getValue());
        }
      }

      @Override
      public void setValue(String value) {
        super.setValue(value);
        commit();
      }
    }
    return new TextDefaultValue();
```

Explanation:

The class CustomPropertyDefaultValueAdapter converts requests from CustomPropertyClass into messages that the CustomPropertyDefinition class can understand. In a sense, this adapter acts as a means for these two classes to communicate with eachother seamlessly, despite being coded in different languages.

Concise explanation and clarifying this concrete classes and the interaction between them. A little detail would be reforcing that without the adapter class, libraries would conflite and classes couldn't communicate. – ***Reviewed by Joana***



ANA'S CODE SMELLS

2.1.1 DUPLICATE CODE

Location: ganttproject\src\main\java\net\sourceforge\ganttproject\undo (line 77-186)

Class: UndoableEditImpl.java

Code Snippet:

```java
public void redo() throws CannotRedoException {
  try {
    restoreDocument(myDocumentAfter);
    if (projectDatabaseTxn != null) {
      try {
        projectDatabaseTxn.redo();
      } catch (ProjectDatabaseException e) {
        GPLogger.log(e);
      }
    }
  } catch (DocumentException | IOException e) {
    undoRedoExceptionHandler(e);
  }
}
```

```java
public void undo() throws CannotUndoException {
  try {
    restoreDocument(myDocumentBefore);
    if (projectDatabaseTxn != null) {
      try {
        projectDatabaseTxn.undo();
      } catch (ProjectDatabaseException e) {
        GPLogger.log(e);
      }
    }
  } catch (DocumentException | IOException e) {
    undoRedoExceptionHandler(e);
  }
}
```

Explanation: These methods have with almost the same code.

Refactoring idea: Create a new auxiliar method, with these code(of 1 method) and a parameter(ex: boolean true > “undo” false -> “redo”) with an if inside the second(“try..catch”) with a condition to choose the “ProjectDatabaseTxn.undo()” or “ProjectDatabaseTxn.redo()” call, the other part it´s just the same.

Firstly, I am totally agreed with that example. The refactoring proposed it´s also a possible solution, but in my opinion, you can give more alternatives even betters. – ***Reviewed by Joana***

2.1.2. LONG PARAMETER LIST

Location:

ganttproject\src\main\java\net\sourceforge\ganttproject\calendar (line 208)

Class: CalendarEditorPanel.java

Code Snippet:

```
public Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, final int row, int column) {
  TableCellRenderer renderer = table.getCellRenderer(row, column);
  Component c = renderer.getTableCellRendererComponent(table, value, isSelected,  hasFocus: true, row, column);
```

Explanation: There is a very long parameter list of the function "getTableCellEditorComponent".

Refactoring idea: Pass by parameter of the function only the "Component" object, that has 5 of the parameters passed through.

The "render" object is only useful to create c, and the row and value parameters that are used in the function we could call with a get function of component class. This would turn the code shorter and without not useful object creations, like "render".

It is a good example for long parameter code smell. However, you could be more specific in identifying the problem in this code and the problems it causes. – ***Reviewed by Joana***

2.1.3. LONG METHOD

Location:

ganttproject\src\main\java\net\sourceforge\ganttproject\document (line 118-161)

Class: DocumentCreator.java

Code Snippet:

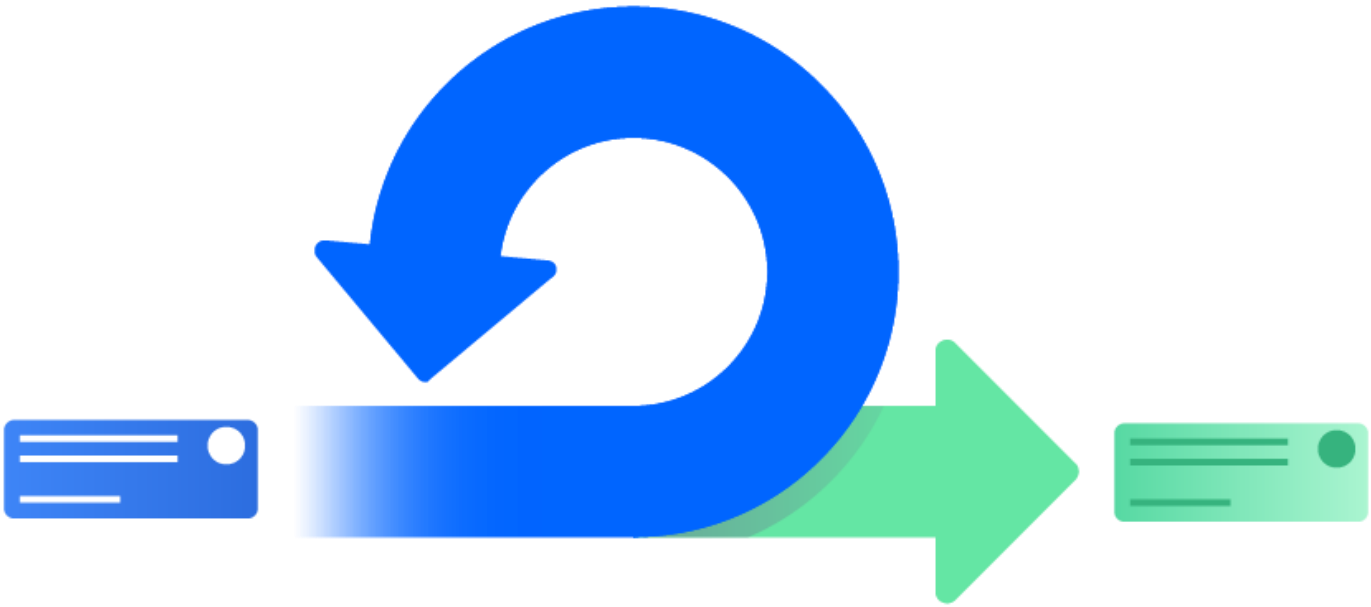
```
private Document createDocument(String path, File relativePathRoot, String user, String pass) {
  assert path != null;
  path = path.trim();
  String lowerPath = path.toLowerCase();
  if (lowerPath.startsWith("http://") || lowerPath.startsWith("https://")) {
    try {
      if (user == null && pass == null) {
        WebDavServerDescriptor server = myWebDavStorage.findServer(path);
        if (server != null) {
          user = server.getUsername();
          pass = server.getPassword();
        }
      }
      return new HttpDocument(path, user, pass, myWebDavStorage.getProxyOption());
    } catch (IOException e) {
      GPLogger.log(e);
      return null;
    } catch (WebDavException e) {
      GPLogger.log(e);
      return null;
    }
  } else if (lowerPath.startsWith("cloud://")) {
    var patchedUrl : Pair<URL, String> = DocumentKt.asDocumentUrl(lowerPath);
    if (patchedUrl.component2().equals("cloud")) {
      return new GPCloudDocument(
          teamRefid: null,
          DocumentUri.LocalDocument.createPath(patchedUrl.component1().getPath()).getParent().getFileName(),
          patchedUrl.component1().getHost(),
          DocumentUri.LocalDocument.createPath(patchedUrl.component1().getPath()).getFileName(),
          projectJson: null
      );
    }
  } else if (lowerPath.startsWith("ftp:")) {
    return new FtpDocument(path, myFtpUserOption, myFtpPasswordOption);
  } else if (!lowerPath.startsWith("file://") && path.contains("://")) {
    // Generate error for unknown protocol
    throw new RuntimeException("Unknown protocol: " + path.substring(0, path.indexOf("://")));
  }
  File file = new File(path);
  if (file.toPath().isAbsolute()) {
    return new FileDocument(file);
  }
  File relativeFile = new File(relativePathRoot, path);
  return new FileDocument(relativeFile);
}
```

Explanation: There is a very long method, confusing and imperceptible.

Refactoring idea: Add new auxiliar methods for the different tasks of this big method and do some “document” and “file” verifications at those specific classes.

By the nomenclature and parameters of the method we already know that by nature it is gonna be pretty exhaustive regarding conditional checking. Therefore a common approach is to extract some of the conditions, as in, it is better to decompose these verifications into either private methods or with object methods as said in the refactoring proposal. – ***Reviewed by Nelson***

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY



JOANA'S CODE SMELLS

2.2.1. DATA CLASS

Code Snippet:

```
public class ZoomActionSet {
  private final ZoomInAction myZoomIn;

  private final ZoomOutAction myZoomOut;

  public ZoomActionSet(ZoomManager zoomManager) {
    myZoomIn = new ZoomInAction(zoomManager);
    myZoomIn.putValue(Action.SHORT_DESCRIPTION, newValue: null);
    myZoomOut = new ZoomOutAction(zoomManager);
    myZoomOut.putValue(Action.SHORT_DESCRIPTION, newValue: null);
  }

  public GPAction getZoomInAction() {
    return myZoomIn;
  }

  public GPAction getZoomOutAction() {
    return myZoomOut;
  }
}
```

Location:

ganttproject>src>main>java>net>sourceforge>ganttproject>action>zoom>ZoomActionSet.java

Explanation:

It´s a too small class because only contain data and two getter method. There is no real functionality and then it´s no necessary class.

Refactoring Proposal:

These behaviors can be directly placed on the class that need interact with those objects.

We can see that this class is just encapsulating methods or variables that could've easily been put in the class that handles them, there is no extra functionalities being provided therefore I do agree with the observation and explanatory statement. – ***Reviewed by Nelson***

2.2.2. MESSAGE CHAIN

Code Snippet:

```
77  protected Date shift(Date date, int shift) {
78    if (shift == 0) {
79      // No shifting is required
80      return date;
81    }
82    return myDependency.getDependant().getManager().getCalendar().shiftDate(date,
83        myDependency.getDependant().getManager().createLength(shift));
84  }
```

Location:

ganttproject>src>main>java>net>sourceforge>ganttproject>task>dependency>constraint>ConstraintImpl.java (line 82-83)

Explanation:

The code is calling to get an object back, and then calling again, getting another object to invoke another method and so on. It has long message chains, which can cause rigidity, complexity and makes code harder to test independently.

Refactoring Proposal:

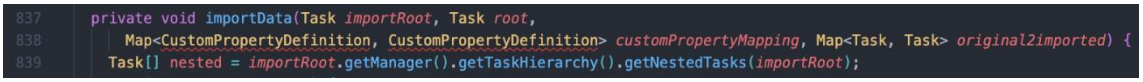
Consider the Law of Demeter, trying to not violate them, as they specify which methods are allowed to call.

While the explanation itself is quite informative, the refactoring proposal could be more complete, as it lacks some rationale behind the use of the law of Demeter. Some more of what this law states could've been touched upon, such as making it so that the method can only call other methods encapsulated within the same class or within a parameter of the method.

Other than that, solid explanation and possible solution. – ***Reviewed by Renato***

2.2.3. LONG PARAMETER LIST

Code Snippet:



```
837    private void importData(Task importRoot, Task root,
838        Map<CustomPropertyDefinition, CustomPropertyDefinition> customPropertyMapping, Map<Task, Task> original2imported) {
839      Task[] nested = importRoot.getManager().getTaskHierarchy().getNestedTasks(importRoot);
```

Location:

ganttproject>src>main>java>net>sourceforge>ganttproject>task>

TaskManagerImpl.java

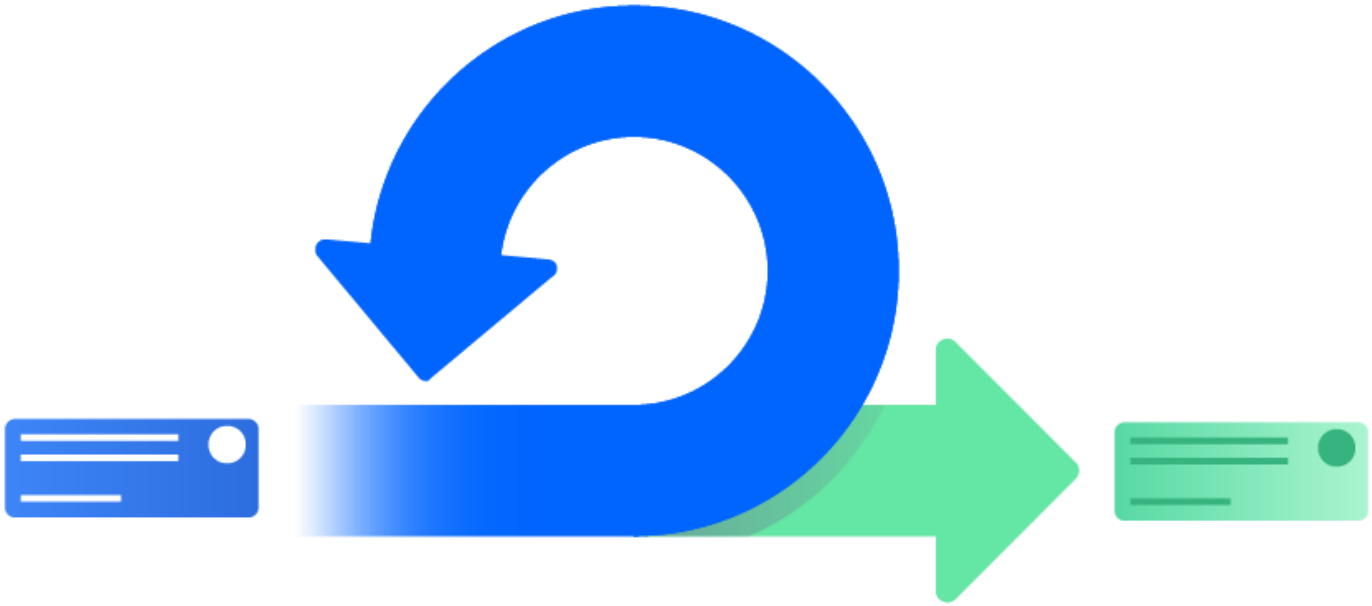
Explanation:

This method has a long parameter list that can be difficult to use and confused to understand. That can also increase the chance of something going wrong.

Refactoring Proposal:

Introduce parameter objects.

The method from the snippet indeed presents a very long list of parameters, making the code not so easy to read. The presented solution is in fact the best option to solve this issue. – ***Reviewed by Pedro***



NELSON'S CODE SMELLS

2.3.1. SWITCH STATEMENT

Code Snippet:

```
for (int i = 0; i < MAX_ATTEMPTS; i++) {
  HttpResponse result = httpClient.execute(getRssUrl);
  switch (result.getStatusLine().getStatusCode()) {
    case HttpStatus.SC_OK:
      processResponse(result.getEntity().getContent());
      return;
  }
}
```

Exact location:

ganttproject/src/main/java/net.sourceforge.ganttproject/client/RssFeedChecker (line 189)

Explanation:

Here we can see a switch statement being used in a not-so-great way. Usually, we opt for using switch statements when there are multiple case labels which is not the case. To simplify the code, we could just replace it with an if statement.

Refactoring:

To simplify the code, we could just replace it with an if statement. Thought it's not as flexible as a switch statement, as in if we need to add more cases, we will have multiple if statements, it's more than enough for what we have at the moment.

Both the explanation and the refactoring proposal are informative, compact and easy to understand. There is nothing to add. – ***Reviewed by Renato***

2.3.2. DUPLICATED CODE

Code Snippet:

```
public CopyAction asToolbarAction() {
  final CopyAction result = new CopyAction(myViewmanager);
  result.setFontAwesomeLabel(UIUtil.getFontawesomeLabel(result));
  this.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
      if ("enabled".equals(evt.getPropertyName())) {
        result.setEnabled((Boolean)evt.getNewValue());
      }
    }
  });
  result.setEnabled(this.isEnabled());
  return result;
}
-------------------------------------//-------------------------------------
public CutAction asToolbarAction() {
  final CutAction result = new CutAction(myViewmanager, myUndoManager);
  result.setFontAwesomeLabel(UIUtil.getFontawesomeLabel(result));
  this.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
      if ("enabled".equals(evt.getPropertyName())) {
        result.setEnabled((Boolean)evt.getNewValue());
      }
    }
  });
  result.setEnabled(this.isEnabled());
  return result;
}
```

Exact location: asToolbarAction() -> ganttproject/src/main/java/net.sourceforge.ganttproject/action/edit/copyAction (Class Copy Action (line 47))

Explanation:

As we can see this method is almost identical in these two classes, only difference being the class that implements it.

Refactoring:

In this folder we could just create an abstract class that has an abstract method called asToolbar()" that will be further implemented by its subclasses.

Simple explanation. Good and traightforward Refactoring Solution, there only should be a little more explicit that we are talking about 2 methods of different classes,"CutAction" and "CopyAction". – ***Reviewed by Ana***

2.3.3. DEAD CODE

Code Snippet:

```
private void createUpdateDialog(String content) {
//     RssUpdate update = parser.parseUpdate(content);
//     if (update != null) {
//       UpdateDialog.show(myUiFacade, update);
//     }
  }
```

Exact location:

createUpdateDialog(String content) -> ganttproject/src/main/java/net.sourceforge.ganttproject/client/RssFeedChecker (line 241)

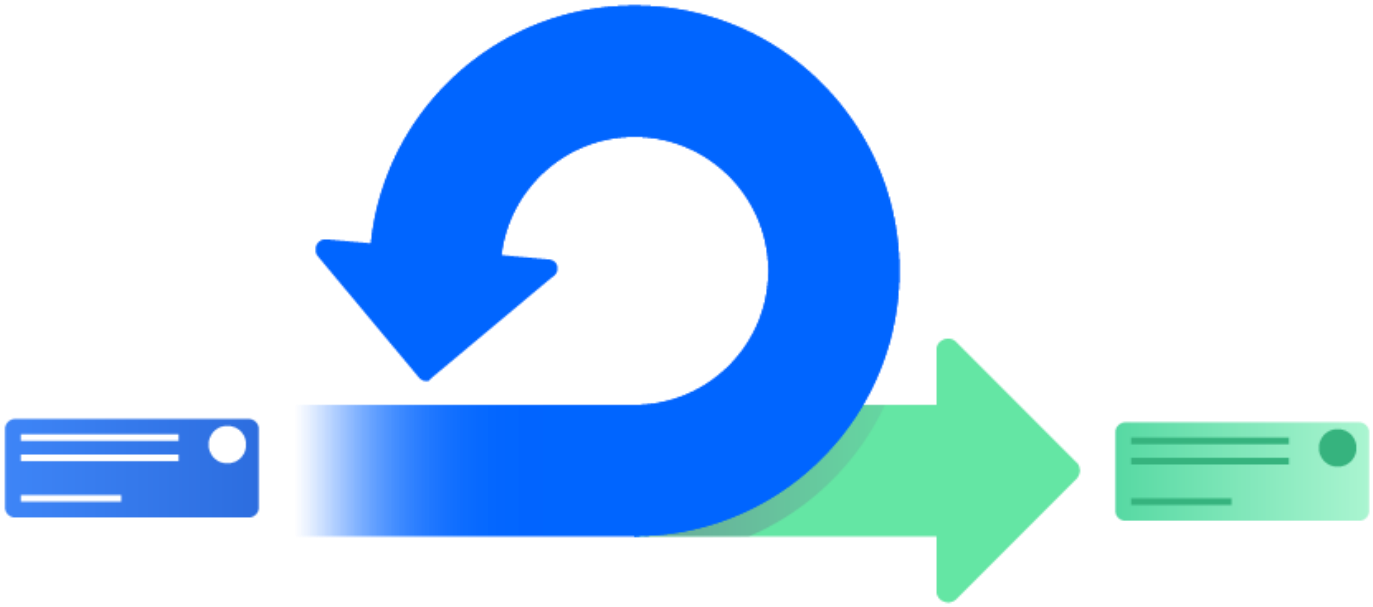
Explanation:

In this method we can see that all of its body is commented which indicates that the method is no longer operational. We want to avoid these types of things because they can easily add up and make our files unnecessarily big.

Refactoring proposal:

We want to remove this method and all its reference calls from the project, so that when future people see the code or even write in this project are not mislead by such methods.

Although this method is called once in the same class, it doesn't do anything since it's empty (only contains commented code): as the explanation confirms, we're in the presence of dead code. The refactoring proposal is the most adequate one, and the best solution for this code smell. – ***Reviewed by Pedro***



PEDRO'S CODE SMELLS

2.4.1. COMMENTS

Location: ganttproject/src/main/java/org/imgscalr

Class: Scalr.java

Code Snippet:

```
/**
 * Used to create a {@Link BufferedImage} with the most optimal RGB TYPE (
 * {@Link BufferedImage#TYPE_INT_RGB} or {@Link BufferedImage#TYPE_INT_ARGB}
 * ) capable of being rendered into from the given <code>src</code>. The
 * width and height of both images will be identical.
 * <p/>
 * This does not perform a copy of the image data from <code>src</code> into
 * the result image; see {@Link #copyToOptimalImage(BufferedImage)} for
 * that.
 * <p/>
 * We force all rendering results into one of these two types, avoiding the
 * case where a source image is of an unsupported (or poorly supported)
 * format by Java2D causing the rendering result to end up looking terrible
 * (common with GIFs) or be totally corrupt (e.g. solid black image).
 * <p/>
 * Originally reported by Magnus Kvalheim from Movellas when scaling certain
 * GIF and PNG images.
 *
 * @param src
 *            The source image that will be analyzed to determine the most
 *            optimal image type it can be rendered into.
 *
 * @return a new {@Link BufferedImage} representing the most optimal target
 *         image type that <code>src</code> can be rendered into.
 *
 * @see <a
 *      href="http://www.mail-archive.com/java2d-interest@capra.eng.sun.com/msg05621.html">How
 *      Java2D handles poorly supported image types</a>
 * @see <a
 *      href="http://code.google.com/p/java-image-scaling/source/browse/trunk/src/main/java/com/mortennobel/imagescaling/MultiStepRescaleOp.java">Thanks
 *      to Morten Nobel for implementation hint</a>
 */
Dmitry Barashev
protected static BufferedImage createOptimalImage(BufferedImage src) {
  return createOptimalImage(src, src.getWidth(), src.getHeight());
}
```

Explanation: In this class we can see too many comments above (and even inside) the methods and variables. This type of commenting can become very confusing to understand, and can be an indicative of bad or complicated design.

Refactoring: A solution could be reducing a bit the size of the comments of each method, making it more organized and easy to understand. The decision can be done through importance of comment, leaving the more important ones, such as succint descriptions of the parameters and methods (basic documentation), and explanations of algorithms.

Direct and assertive explanation. Good Refactoring, giving different solutions and examples to a better "coments" use. Particularly he could evidence here the idea of the principle of "say more with fewer words". – ***Reviewed by Ana***

2.4.2. DATA CLASS

Location: ganttproject/src/main/java/net/sourceforge/ganttproject/task

Class: TaskView.java

Code Snippet:

```
public class TaskView {
  1 usage
  private final Set<Task> myTimelineTasks = Sets.newHashSet();

  /**
   * @return a set of tasks which are shown as labels in the timeline
   */
  dbarashev
  public Set<Task> getTimelineTasks() { return myTimelineTasks; }
}
```

Explanation: This class is composed of an instance variable and its getter method. Hence, the class is too small and has no real funcionality, it only has data. This indicates that it's not really necessary.

Refactoring: One possible solution could be removing the class and replacing it by a variable of the type Set<Task> where the object of this class is being called/used.

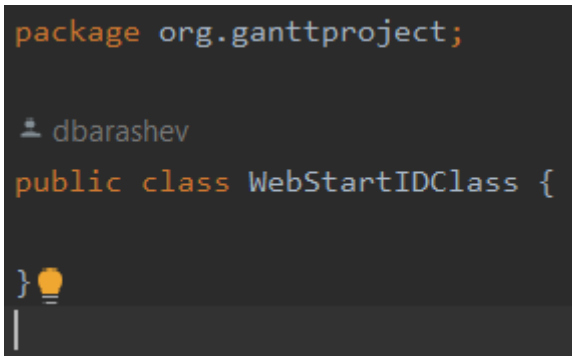
The explanation is clear and assertive. As for the refactoring proposal, more solutions could've been proposed as to merely delete the class and replace it with a variable. Perhaps ponder if such a class could have something else placed in it or what other classes handle this very same data. Overall, decent explanation and refactoring proposal. – ***Reviewed by Renato***

2.4.3. DEAD CODE

Location: ganttproject/src/main/java/org/ganttproject

Class: WebStartIDClass.java

Code Snippet:

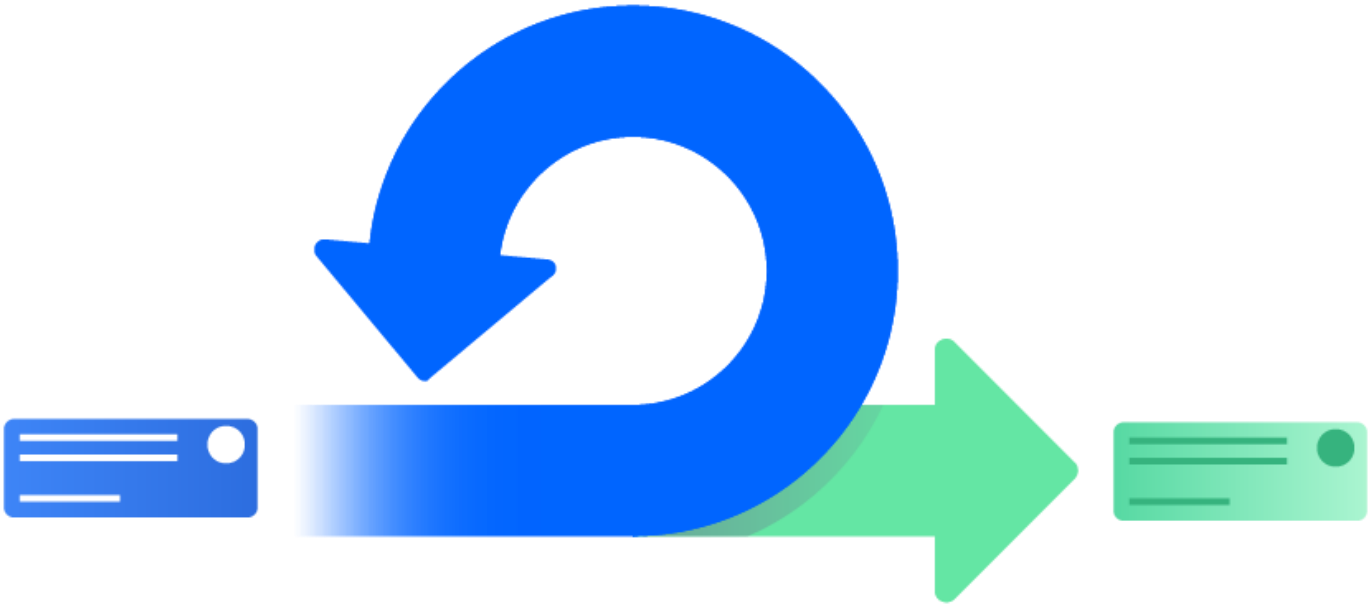


Explanation: This class is never being used in any part of the code, so it is unreachable. Besides that, it doesn't contain any information at all, being even more useless and increasing the code complexity unnecessarily.

Refactoring: The class could be removed, as it is not being used, and the code would be cleaner and simpler, only containing useful implementations.

After reading the refactoring proposal and checking if there are any reference calls of the WebStartIDClass class we can conclude that the class is in fact obsolete.

Therefore I agree with what is being stated above. – ***Reviewed by Nelson***



RENATO'S CODE SMELLS

2.5.1. REMINDER COMMENT

Location:
ganttproject/src/main/java/net/sourceforge/ganttproject/export

Class: CommandLineExportApplication.java

Code Snippet:

```
private boolean export(Exporter exporter, Args args, IGanttProject
project, UIFacade uiFacade) {
    logger.debug("Using exporter {}", new Object[]{exporter}, new
HashMap<>());
    ConsoleUIFacade consoleUI = new ConsoleUIFacade(uiFacade);
    GPLogger.setUIFacade(consoleUI);
    // TODO: bring back task expanding
//    if (myArgs.expandTasks) {
//      for (Task t : project.getTaskManager().getTasks()) {
//        project.getUIFacade().getTaskTree().setExpanded(t, true);
//      }
//    }
```

Explanation: In this case, the TODO comment indicates that task expanding has to be reimplemented and even accompanies itself with a few lines of related but possibly incomplete commented code, which can indicate shady code.

Refactoring proposal: Either implement task expanding or delete unused code and TODO comment.

Good and simple define of the code Smell, with the ilustrating image attached with the details of the code explanation. Easy and certain solution.

– Reviewed by Ana

2.5.2. DEAD CODE

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

Class: CutAction.java

Code Snippet:

```
public void actionPerformed(ActionEvent e) {
    if (calledFromAppleScreenMenu(e)) {
      return;
    }
//    myUndoManager.undoableEdit(getLocalizedName(), new Runnable() {
//      @Override
//      public void run() {

myViewmanager.getSelectedArtefacts().startMoveClipboardTransaction();
//      }
    //});
  }
```

Explanation: Nearly half the body of the function *actionPerformed()* is obsolete code of a function that no longer exists. This clutters the method with useless and outdated information, which can indicate sloppy coding.

Refactoring purposal: Clean up obsolete code.

This is not exactly what I understand by dead code, in the sense that there is no code that is never used/executed.But I agreed with the fact that the method should be clean as there is a commented code which makes the method confused and sloppy. – ***Reviewed by Joana***

2.5.3. LONG METHOD

Location:
ganttproject/src/main/java/net/sourceforge/ganttproject/export

Class: CommandLineExportApplication.java

Code Snippet:

```
private boolean export(Exporter exporter, Args args, IGanttProject
project, UIFacade uiFacade) {
    logger.debug("Using exporter {}", new Object[]{exporter}, new
HashMap<>());
    ConsoleUIFacade consoleUI = new ConsoleUIFacade(uiFacade);
    GPLogger.setUIFacade(consoleUI);
    // TODO: bring back task expanding
//    if (myArgs.expandTasks) {
//      for (Task t : project.getTaskManager().getTasks()) {
//        project.getUIFacade().getTaskTree().setExpanded(t, true);
//      }
//    }

    Job.getJobManager().setProgressProvider(new
ConsoleProgressProvider());
    File outputFile = args.outputFile == null ?
FileChooserPage.proposeOutputFile(project, exporter)
        : args.outputFile;

    Preferences prefs = new PluginPreferencesImpl(null, "");
    prefs.putInt("zoom", args.zooming);
    prefs.put(
        "exportRange",

DateParser.getIsoDate(project.getTaskManager().getProjectStart()) + "
"
            +
DateParser.getIsoDate(project.getTaskManager().getProjectEnd()));
    prefs.putBoolean("commandLine", true);

    // If chart to export is defined, then add a string to prefs
    if (args.chart != null) {
      prefs.put("chart", args.chart);
    }

    // If stylesheet is defined, then add a string to prefs
    if (args.stylesheet != null) {
      prefs.put("stylesheet", args.stylesheet);
    }

    prefs.putBoolean("expandResources", args.expandResources);
```

```
    exporter.setContext(project, consoleUI, prefs);
    final CountDownLatch latch = new CountDownLatch(1);
    try {
      ExportFinalizationJob finalizationJob = exportedFiles ->
latch.countDown();
      exporter.run(outputFile, finalizationJob);
      latch.await();
    } catch (Exception e) {
      consoleUI.showErrorDialog(e);
    }
    return true;
  }
```

Explanation: This method is very lengthy and complex, making it difficult to perceive.

Refactoring proposal: Make use of some auxiliary methods, in order to split the problem into smaller and more percetible parts, thus making the *export()* method itself easier to understand.

This method's structure could be a little bit less complex if it was divided into smaller auxiliary methods (making the verifications, or even returning objects), as said in the refactoring proposal, even though this method can be naturally long in certain parts, because it calls a lot of other functions and needs many verifications. As a sidenote, we could also reduce significantly its size by removing the commented code, which is also a pretty bad code smell.

– Reviewed by Pedro