**Code smells Ricardo Pereira**

* Not using try-catch mechanism for error-checking (DesktopAdapter) [DesktopAdapter.java](https://github.com/anamfrancisco/ganttproject_SE/blob/master/biz.ganttproject.desktop/src/biz/ganttproject/desktop/DesktopAdapter.java)

 @Override

      public void openFiles(OpenFilesEvent e) {

        List<File> files = e.getFiles();

        if (files.isEmpty()) {

          return;

        }

        File file = files.get(0);

        if (!file.isFile() || !file.canRead()) {

          return;

        }

        api.openFile(file);

      }

—-------------------------------------------------------------

* No comments explaining the code (ExporterToHTML)  [ExporterToHTML.java](https://github.com/anamfrancisco/ganttproject_SE/blob/d4c23fa8a2a7992488d2e46a21d9c2aa2d8a2f63/org.ganttproject.impex.htmlpdf/src/main/java/org/ganttproject/impex/htmlpdf/ExporterToHTML.java)

(Whole class has no comments explaining the code, even on long, obscure methods)

private ExporterJob createGenerateGanttChartJob(final File outputFile, final List<File> resultFiles) {

    ExporterJob result = new ExporterJob("generate gantt chart") {

      @Override

      protected IStatus run() {

        try {

          int zoomLevel = getPreferences().getInt("zoom", -1);

          var exportSettings = createExportSettings();

          RenderedImage ganttChartImage = getGanttChart().asPrintChartApi().exportChart(

              exportSettings.getStartDate(), exportSettings.getEndDate(), zoomLevel, exportSettings.isCommandLineMode());

          File ganttChartImageFile;

          ganttChartImageFile = replaceExtension(outputFile, GANTT\_CHART\_FILE\_EXTENSION);

          ImageIO.write(ganttChartImage, PNG\_FORMAT\_NAME, ganttChartImageFile);

          resultFiles.add(ganttChartImageFile);

        } catch (IOException e) {

          getUIFacade().showErrorDialog(e);

          return Status.CANCEL\_STATUS;

        } catch (OutOfMemoryError e) {

          getUIFacade().showErrorDialog(new RuntimeException("Out of memory when creating Gantt chart image", e));

          return Status.CANCEL\_STATUS;

        }

        return Status.OK\_STATUS;

      }

    };

    return result;

}

  private ExporterJob createGenerateResourceChartJob(final File outputFile, final List<File> resultFiles) {

    ExporterJob result = new ExporterJob("Generate resource chart") {

      @Override

      protected IStatus run() {

        try {

          int zoomLevel = getPreferences().getInt("zoom", -1);

          var exportSettings = createExportSettings();

          RenderedImage resourceChartImage = getResourceChart().asPrintChartApi().exportChart(

              exportSettings.getStartDate(), exportSettings.getEndDate(), zoomLevel, exportSettings.isCommandLineMode());

          File resourceChartImageFile = replaceExtension(outputFile, RESOURCE\_CHART\_FILE\_EXTENSION);

          ImageIO.write(resourceChartImage, PNG\_FORMAT\_NAME, resourceChartImageFile);

          resultFiles.add(resourceChartImageFile);

        } catch (IOException e) {

          getUIFacade().showErrorDialog(e);

          return Status.CANCEL\_STATUS;

        } catch (OutOfMemoryError e) {

          getUIFacade().showErrorDialog(new RuntimeException("Out of memory when creating resource chart image", e));

          return Status.CANCEL\_STATUS;

        }

        return Status.OK\_STATUS;

      }

    };

    return result;

  }

—-------------------------------------------------------------

* Long parameter list (ExporterToHTML) [ExporterToHTML](https://github.com/anamfrancisco/ganttproject_SE/blob/d4c23fa8a2a7992488d2e46a21d9c2aa2d8a2f63/org.ganttproject.impex.htmlpdf/src/main/java/org/ganttproject/impex/htmlpdf/ExporterToHTML.java)

RenderedImage ganttChartImage = getGanttChart().asPrintChartApi().exportChart(

              exportSettings.getStartDate(), exportSettings.getEndDate(), zoomLevel, exportSettings.isCommandLineMode());

**GoF José Pereira**

* **Abstract Factory** (StylesheetExporterBase) [StylesheetExporterBase.java](https://github.com/anamfrancisco/ganttproject_SE/blob/d4c23fa8a2a7992488d2e46a21d9c2aa2d8a2f63/org.ganttproject.impex.htmlpdf/src/main/java/org/ganttproject/impex/htmlpdf/StylesheetExporterBase.java)

(produce families of related objects without specifying their concrete classes)

public abstract class StylesheetExporterBase extends ExporterBase {

  private GPOptionGroup myOptions;

  protected EnumerationOption createStylesheetOption(String optionID, final List<Stylesheet> stylesheets) {

    final List<String> names = new ArrayList<String>();

  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;

  }

  @Override

  public abstract String[] getFileExtensions();

  protected abstract List<Stylesheet> getStylesheets();

  protected abstract void setSelectedStylesheet(Stylesheet stylesheet);

  protected abstract String getStylesheetOptionID();

  public StylesheetExporterBase() {

  }

  @Override

  public Component getCustomOptionsUI() {

    return null;

  }

  @Override

  public void setContext(IGanttProject project, UIFacade uiFacade, Preferences prefs) {

    super.setContext(project, uiFacade, prefs);

    createStylesheetOption(getStylesheets());

  }

  private void createStylesheetOption(List<Stylesheet> stylesheets) {

    EnumerationOption stylesheetOption = createStylesheetOption(getStylesheetOptionID(), stylesheets);

    stylesheetOption.setValue(stylesheets.get(0).getLocalizedName());

    myOptions = new GPOptionGroup("exporter.html", new GPOption[] { stylesheetOption });

    myOptions.setTitled(false);

  }

  protected void setCommandLineStylesheet() {

    // Check if we are running from command line, if yes then we need to define the

    // stylesheet we are using

    if (getPreferences().getBoolean("commandLine", false) == true) {

      // Get the list of stylesheets

      List<Stylesheet> stylesheets = getStylesheets();

      // Set the first entry of list as default

      setSelectedStylesheet(stylesheets.get(0));

      // Test if a style is present in the arguments from command line

      // Iterate the list of style sheets to find it

      if (getPreferences().get("stylesheet", null) != null) {

        for (Stylesheet sheet : stylesheets) {

          if (sheet.getLocalizedName().compareTo(getPreferences().get("stylesheet", null)) == 0) {

            setSelectedStylesheet(sheet);

            break;

          }

        }

      }

    }

  }

  @Override

  public GPOptionGroup getOptions() {

    return myOptions;

  }

}

* **Builder** (ExporterToHTML) [ExporterToHTML.java](https://github.com/anamfrancisco/ganttproject_SE/blob/d4c23fa8a2a7992488d2e46a21d9c2aa2d8a2f63/org.ganttproject.impex.htmlpdf/src/main/java/org/ganttproject/impex/htmlpdf/ExporterToHTML.java)

(doesn’t implement a constructor, uses methods to define variables instead)

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(this);

  @Override

  public String getFileTypeDescription() {

    return language.getText("impex.html.description");

  }

  @Override

  protected void setSelectedStylesheet(Stylesheet stylesheet) {

    mySelectedStylesheet = (HTMLStylesheet) stylesheet;

  }

 @Override

  public List<GPOptionGroup> getSecondaryOptions() {

    return null;

  }

  @Override

  public String getFileNamePattern() {

    return "html";

  }

(...)

* **Singleton** (ImporterFromMsProjectFile) [ImporterFromMsProjectFile.java](https://github.com/anamfrancisco/ganttproject_SE/blob/d4c23fa8a2a7992488d2e46a21d9c2aa2d8a2f63/biz.ganttproject.impex.msproject2/src/main/java/biz/ganttproject/impex/msproject2/ImporterFromMsProjectFile.java)

(Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.)

 private void findChangedDates(Map<GanttTask, Date> originalDates, Map<Task, Task> buffer2realTask,

      List<Pair<Level, String>> errors) {

    List<Pair<Level, String>> dateChangeMessages = Lists.newArrayList();

    for (Task bufferTask : originalDates.keySet()) {

      Date startPerMsProject = originalDates.get(bufferTask);

      if (startPerMsProject == null) {

        continue;

      }

      Task realTask = buffer2realTask.get(bufferTask);

      if (realTask == null) {

        continue;

      }

      Date startPerGanttProject = realTask.getStart().getTime();

      if (!startPerMsProject.equals(startPerGanttProject)) {

        dateChangeMessages.add(Pair.create(Level.WARNING, GanttLanguage.getInstance().formatText(

            "impex.msproject.warning.taskDateChanged", realTask.getName(), startPerMsProject, startPerGanttProject)));

      }

    }

    if (!dateChangeMessages.isEmpty()) {

      errors.add(Pair.create(Level.INFO, GanttLanguage.getInstance().formatText(

          "impex.msproject.warning.taskDateChanged.heading", dateChangeMessages.size(), originalDates.size())));

      errors.addAll(dateChangeMessages);

    }

  }