*Renato Viola - 60665*

3 Code Smells in Gantt Project

**REMINDER COMMENT**

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

**Class**: CommandLineExportApplication.java

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.

**DEAD CODE**

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

**Class**: CutAction.java

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**Explanation**: Nearly half the body of the function *actionPerformed()* is obsolete code of a function that no longer exists. This clutters the method with useless and outdated information, which can indicate sloppy coding.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

**LONG METHOD**

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

**Class**: CommandLineExportApplication.java

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.