*Renato Viola - 60665*

3 Design Patterns in Gantt Project

**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.

**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.

**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 "translates" requests from the CustomPropertyClass class 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.