Nelson Matos 60483

CODE SMELLS

Switch statement

Illustrating 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 else statements, it’s more than enough for what we have at the moment.

Duplicated code

Illustrating 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.

DEAD CODE

Illustrating 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:

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.