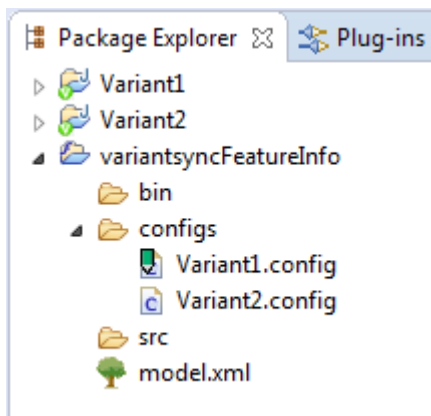


Workflow: Synchronizing Variants Using VariantSync

0. Preconditions

Performing the first steps described in README-file of VariantSync, the package explorer should look similar to this screenshot:



Variant1 and *Variant2* are supported by VariantSync. The Feature-IDE project *variantSyncFeatureInfo* contains a feature mode and a feature configuration representing each variant.

In the menu bar on top of eclipse, VariantSync should provide the following little toolbar:



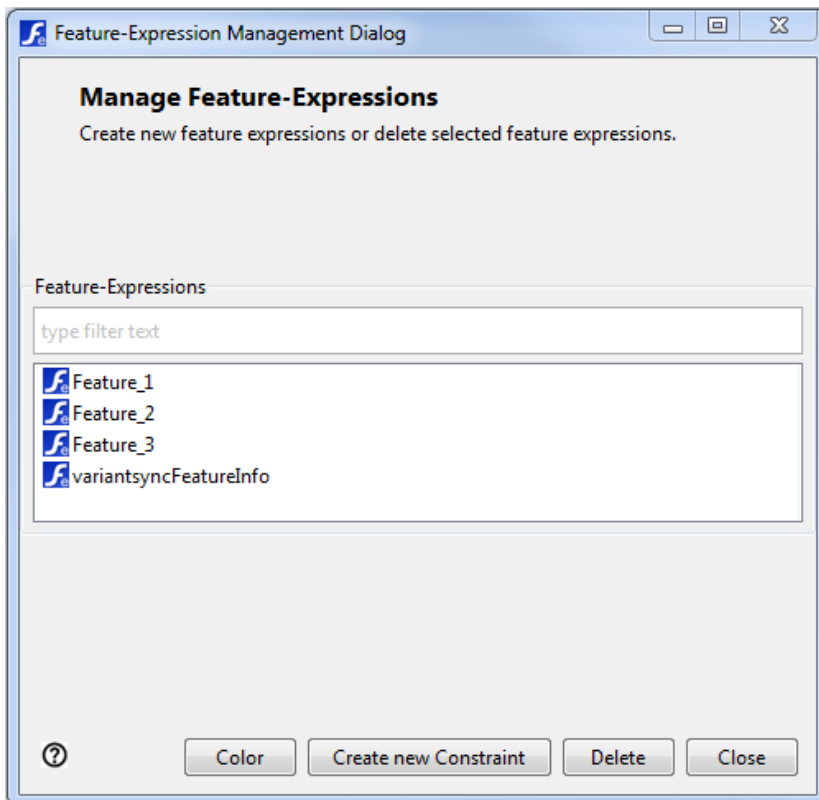
The blue button opens the *feature-expression management dialog*. The green pull-down button shows a list of feature expressions that can be used to activate a context. The selection of one feature expression starts a context. The red sign indicates that a context is active. In this case, context of feature *Feature_1* is active. All actions performed on a variant are tagged to that context. A click on the red sign stops the active context.

Workflow: Synchronizing Variants Using VariantSync

1. Preferences

As first step, we introduce management of feature expressions.

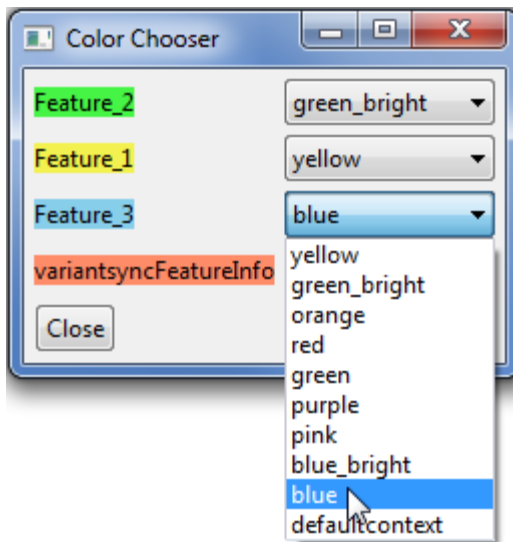
In the *feature-expression management dialog*, all feature expressions used in the domain are listed. Feature expressions can be created, deleted or their color for code-highlighting can be changed.



1.1 Color-Dialog

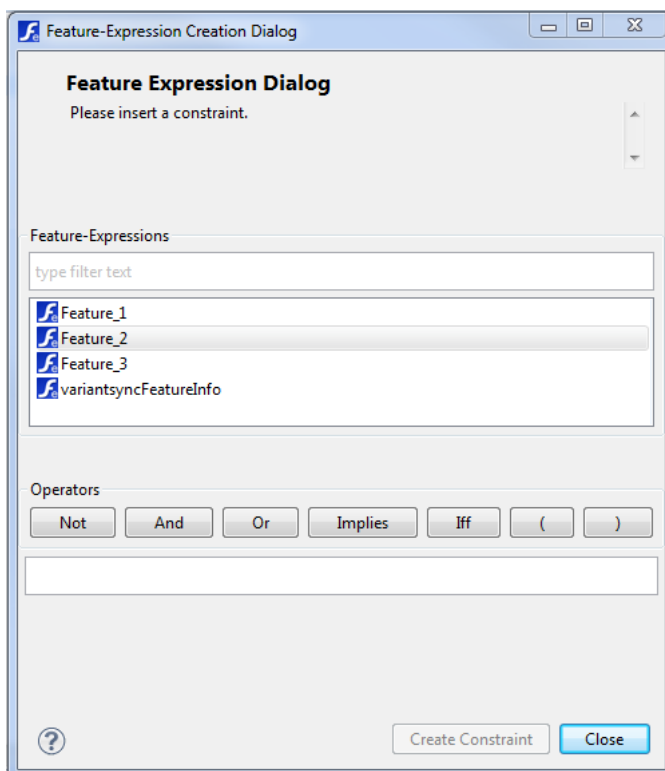
The color dialog enables the user to assign each feature expression a specific color for code-highlighting. Inside a context, code is colored with a certain color to visualize code tagging. Default color is yellow. The user has the option between nine different predefined colors. The color of a feature expression should only be changed if no context is active.

Workflow: Synchronizing Variants Using VariantSync



1.2 Create Feature Expression Dialog

This dialog provides functions to create constraints on features. A constraint represents a feature expression. The dialog is adapted by Feature IDE.



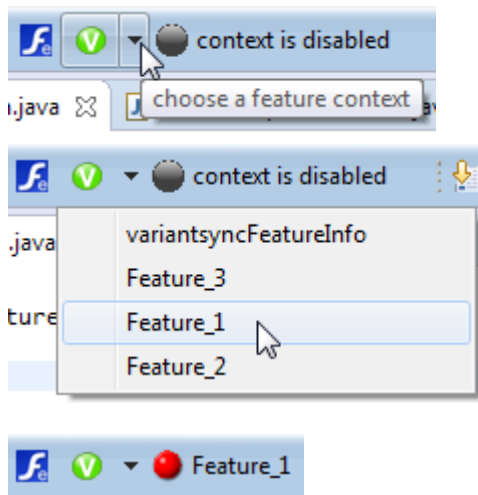
Workflow: Synchronizing Variants Using VariantSync

2. Working in a context

To tag code changes to features and feature expressions, a context covers all changes a user performs on a variant.

2.1 Activating a context

Choose the desired context from the *VariantSync*-pull-down button to activate the context. In our example, we activate the context of *Feature_1*.

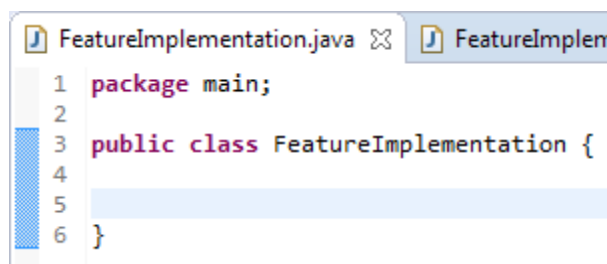


2.2 Implementing a variant

Now, we can work on a variant. The context tags each change to a feature expression after saving a file. The tagging is visualized by highlighting the changed code and adding an annotation to each change. The annotation names the tagged feature.

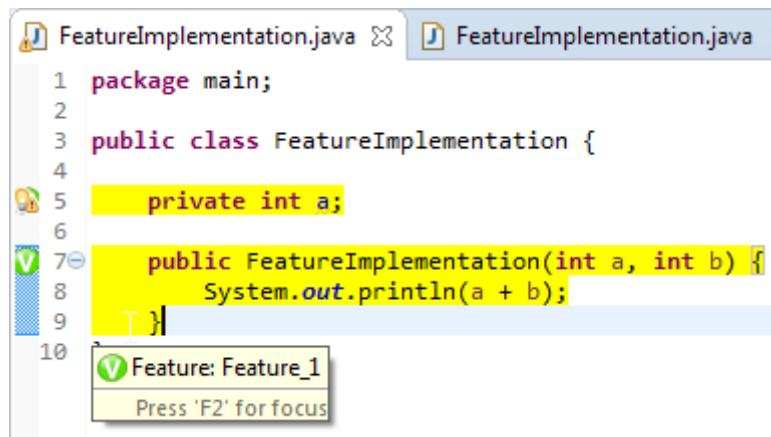
Exemplary tagging:

- 1) Class *FeatureImplementation.java* of *Variant1* is empty.



Workflow: Synchronizing Variants Using VariantSync

2) Code is added to the class and the changes were saved. Now, the added code is tagged to *Feature_1*.

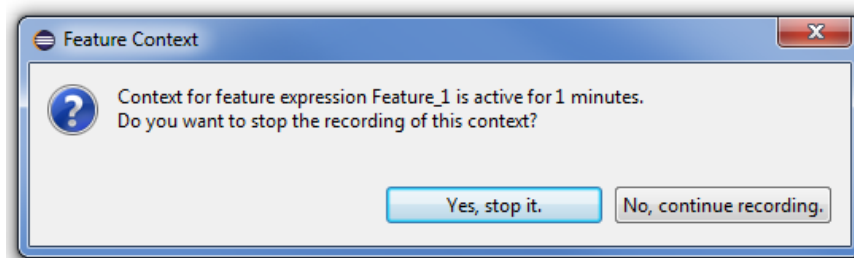


2.3 Stopping a context

Click on the red sign with the name of the active context to stop the context.



Confirm the stop-action.



Now, the context is stopped.

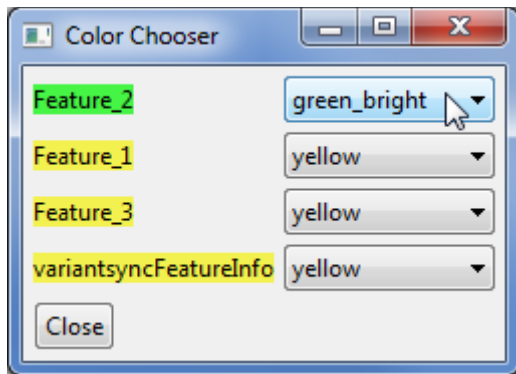


Workflow: Synchronizing Variants Using VariantSync

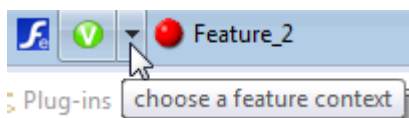
2.4 Working with different features

In 2.2, we implemented code for *Feature_1*. Now, we have stopped context of *Feature_1* and start working on *Feature_2* in the same file.

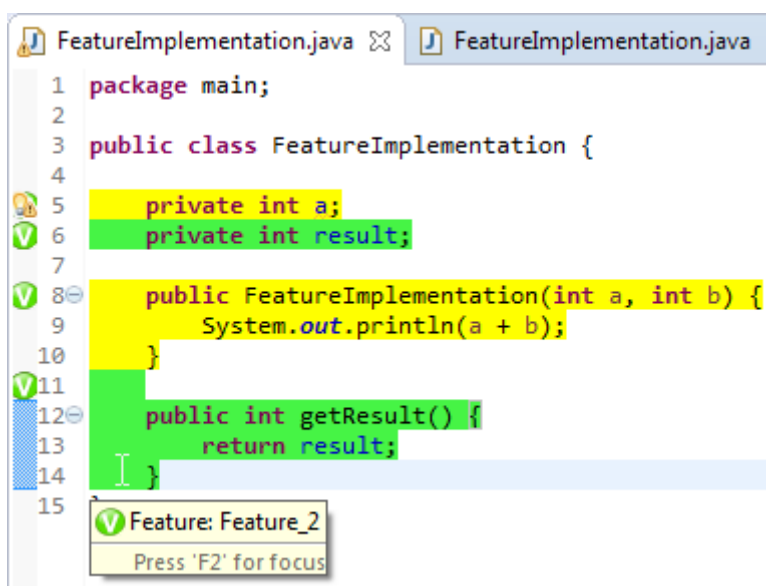
First, we set the color of *Feature_2* to *green_bright* in the color-dialog.



Then, we activate the context for *Feature_2*.



We extend the class by adding a new variable and a new method. Afterwards, we save our changes.



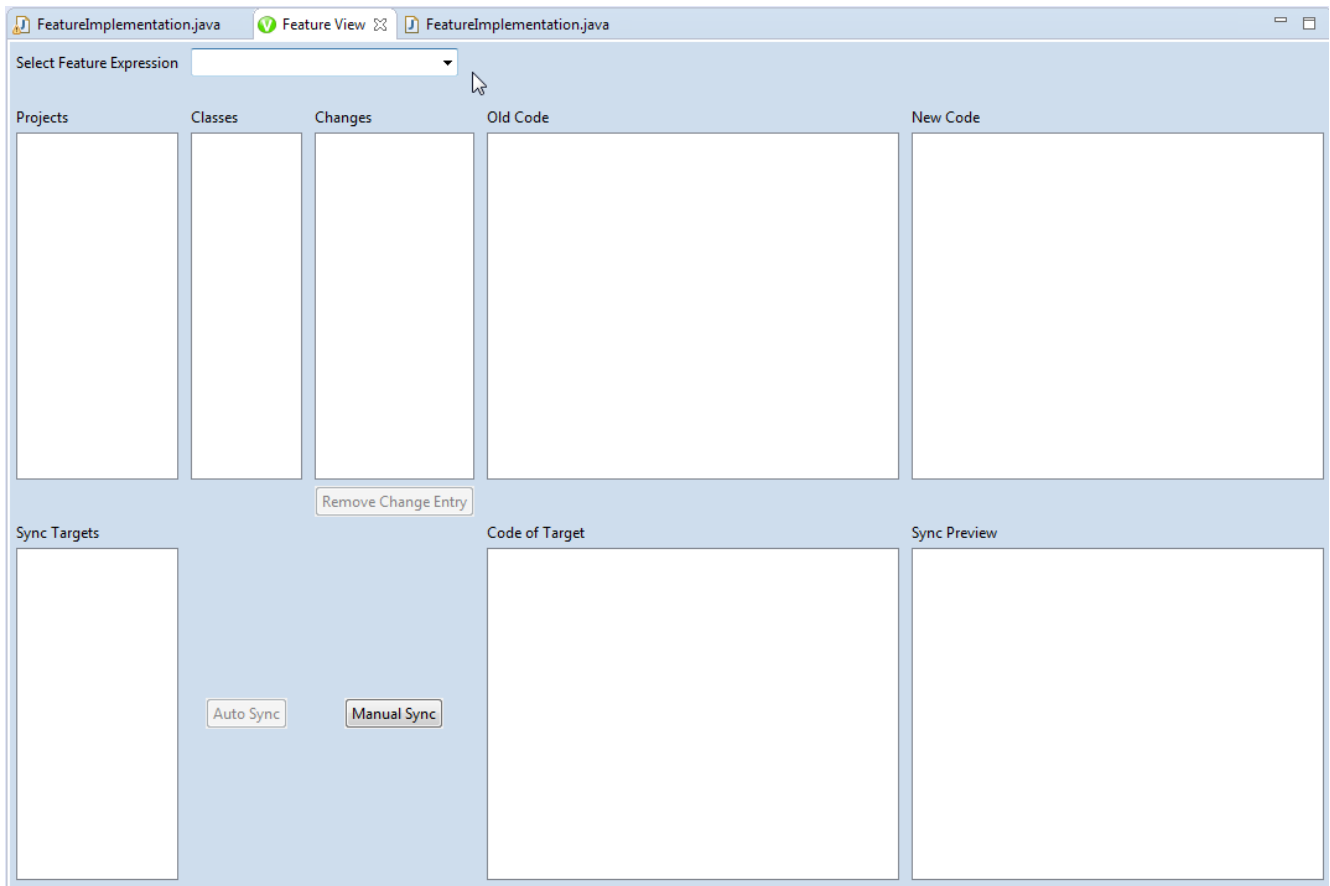
As result, code changes are colored green and tagged to *Feature_2*.

Workflow: Synchronizing Variants Using VariantSync

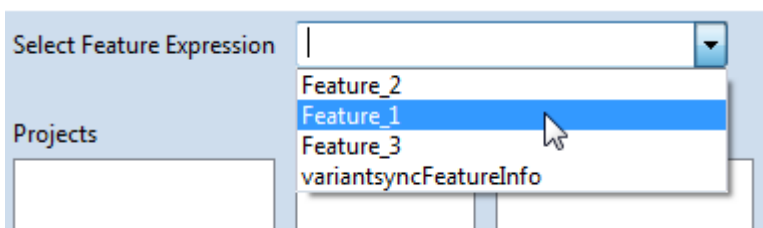
3. Synchronizing Changes with the Feature-View

After performing changes on class *FeatureImplementation.java* of *Variant1*, we synchronize these changes to *Variant2*.

In eclipse menu, we find the feature-view under *Window -> Show View -> Variant Sync -> Feature View*.

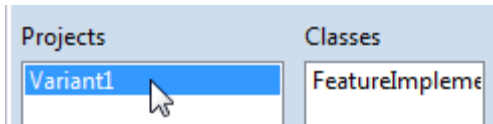


We select the feature we want to synchronize. In our case, it is *Feature_1*.

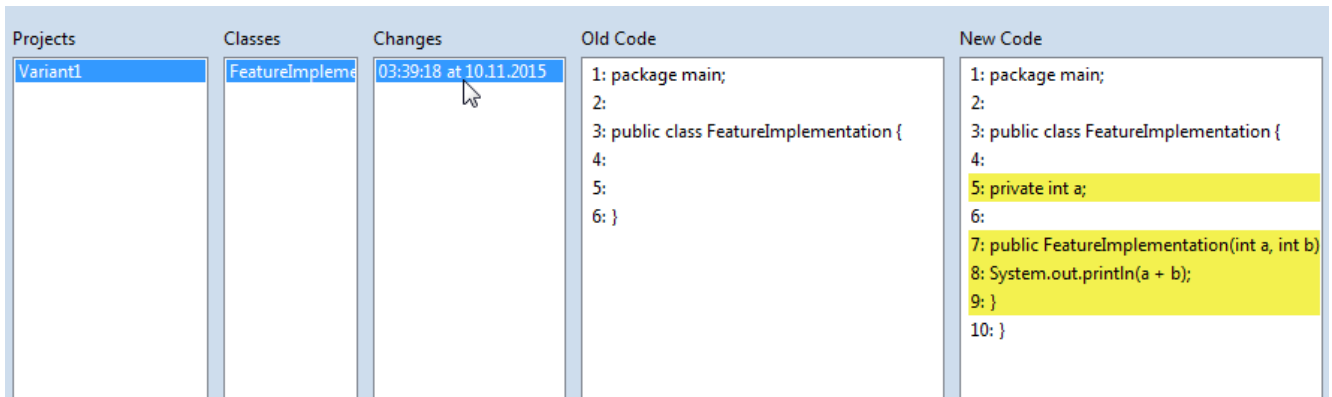


Workflow: Synchronizing Variants Using VariantSync

Then, all projects are listed that contain code changes which are tagged to *Feature_1*. We choose *Variant1*.



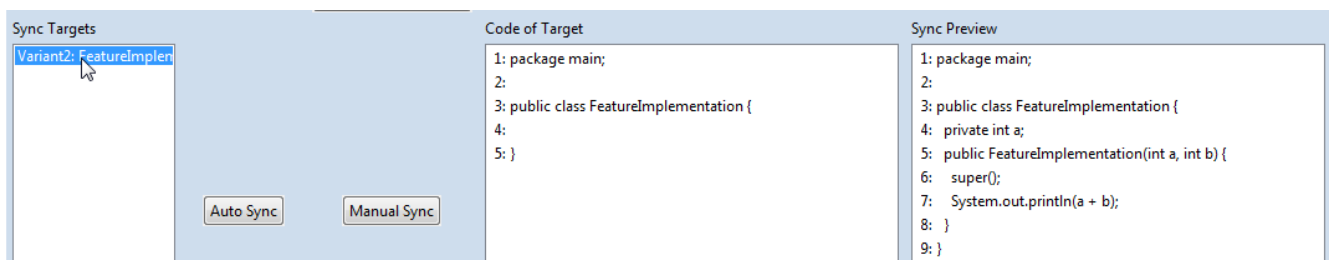
As result, all classes containing changes tagged to *Feature_1* appear. After choosing *FeatureImplementation.java*, all change entries are listed. If we choose a change entry, the old code without our change and the new code containing our change are shown.



Now, VariantSync computes possible synchronization targets and shows these classes under *Sync Targets*. Synchronization targets have to meet three requirements:

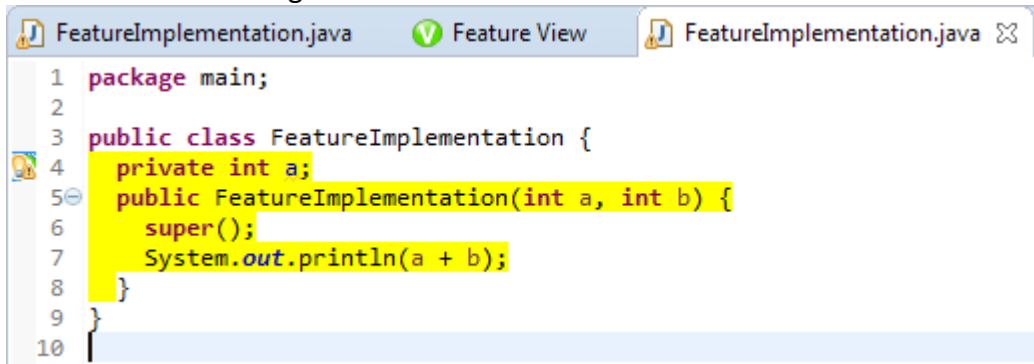
1. Synchronization targets are classes that have the same name than the changed class.
2. Synchronization targets are not in the same variant than the changed class.
3. Synchronization targets are in variants that implement the same feature the changed code is tagged to.

We choose the synchronization target. VariantSync shows the code of the target file and a synchronization preview. This preview shows the code after merging the changed into the target file. If no merge conflicts appear, then the synchronization can be performed automatically by clicking on button *Auto Sync*. In case of merge conflicts, these conflicts are shown in the *sync preview* and need to be merged manually by clicking on button *Manual Sync*.



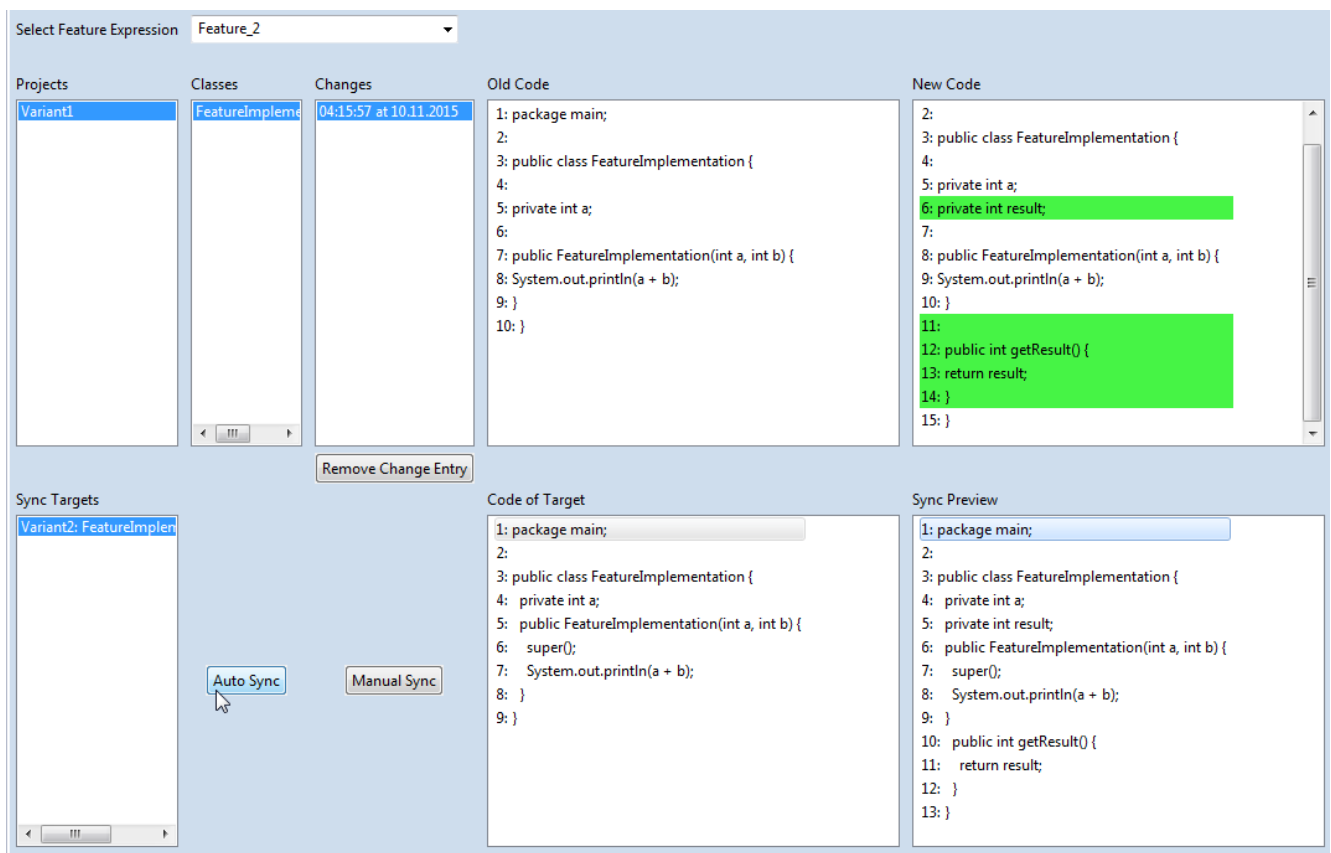
Workflow: Synchronizing Variants Using VariantSync

After automatic synchronization, class *FeatureImplementation.java* of *Variant2* is synchronized and contains the changed code.



```
1 package main;
2
3 public class FeatureImplementation {
4     private int a;
5     public FeatureImplementation(int a, int b) {
6         super();
7         System.out.println(a + b);
8     }
9 }
10
```

Furthermore, *Variant2* also implements *Feature_2*. So, we synchronize these changes, too.



Select Feature Expression: Feature_2

| Projects | Classes | Changes | Old Code | New Code |
|----------|----------------|------------------------|---|--|
| Variant1 | FeatureImpleme | 04:15:57 at 10.11.2015 | <pre>1: package main; 2: 3: public class FeatureImplementation { 4: 5: private int a; 6: 7: public FeatureImplementation(int a, int b) { 8: System.out.println(a + b); 9: } 10: }</pre> | <pre>2: 3: public class FeatureImplementation { 4: 5: private int a; 6: private int result; 7: 8: public FeatureImplementation(int a, int b) { 9: System.out.println(a + b); 10: } 11: 12: public int getResult() { 13: return result; 14: } 15: }</pre> |

Remove Change Entry

Sync Targets: Variant2: FeatureImplemen

Code of Target:

```
1: package main;
2:
3: public class FeatureImplementation {
4: private int a;
5: public FeatureImplementation(int a, int b) {
6: super();
7: System.out.println(a + b);
8: }
9: }
```

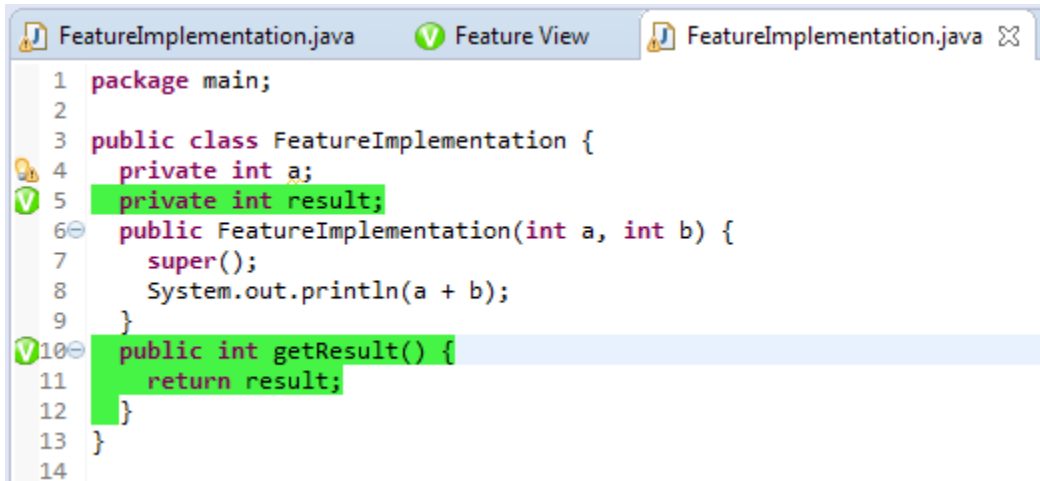
Sync Preview:

```
1: package main;
2:
3: public class FeatureImplementation {
4: private int a;
5: private int result;
6: public FeatureImplementation(int a, int b) {
7: super();
8: System.out.println(a + b);
9: }
10: public int getResult() {
11: return result;
12: }
13: }
```

Auto Sync Manual Sync

Workflow: Synchronizing Variants Using VariantSync

Actually, we are only able to tag the latest merged feature in the target file. *FeatureImplementation.java* of *Variant2* is only tagged to *Feature_2*. The previous changes, tagged to *Feature_1*, are lost.

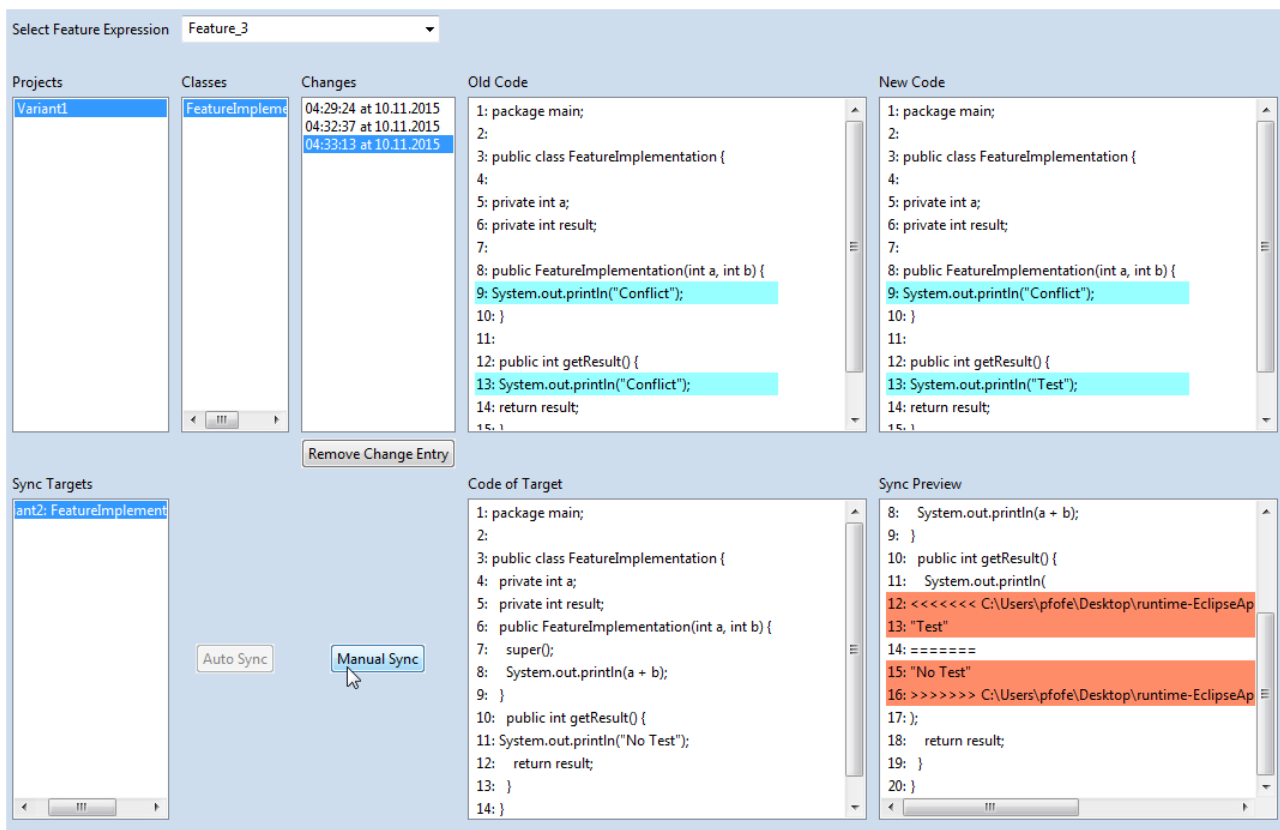


```
1 package main;
2
3 public class FeatureImplementation {
4     private int a;
5     private int result;
6     public FeatureImplementation(int a, int b) {
7         super();
8         System.out.println(a + b);
9     }
10    public int getResult() {
11        return result;
12    }
13 }
14
```

Workflow: Synchronizing Variants Using VariantSync

Manual Sync Dialog

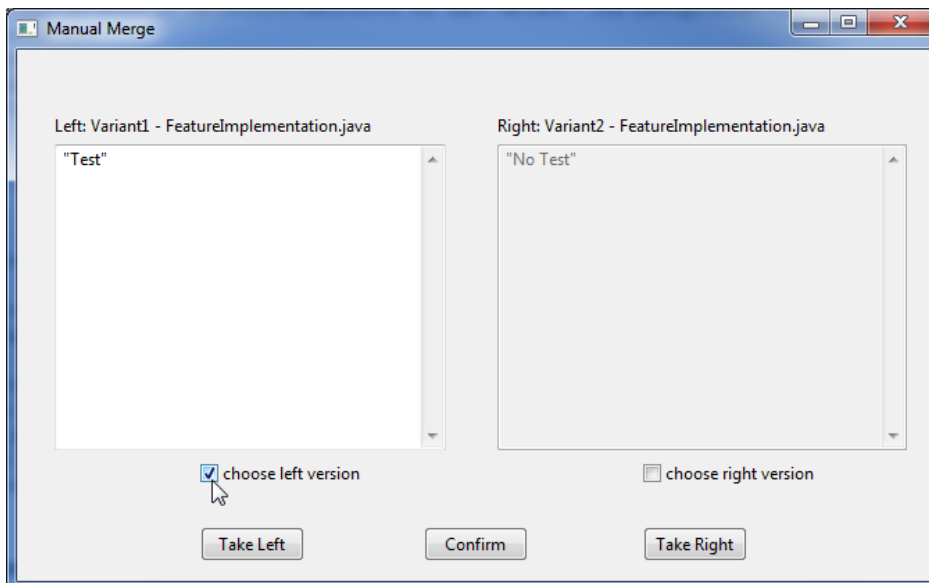
In case of merge conflicts, these conflicts need to be solved manually using the manual merge dialog.



The method `getResult()` writes different strings in an output-stream. In the old version, it was the string *"Conflict"* and we changed it to the string *"Test"*. In the target file, the output-stream writes out the string *"No Test"*. The result is a merge conflict that we have to solve manually.

Workflow: Synchronizing Variants Using VariantSync

The *manual merge dialog* shows the changed code on the left and the code of the target file on the right. We can adapt the code in the text boxes. We need to choose on version to solve the conflict.



We chose the left version. As result, *FeatureImplementation.java* of *Variant2* contains the string *"Test"*.

