JFace – Content Assist – One Trace
Use Full Trace (No Slicing)

Supporting Applications : [JSP Editor]

```
import org.eclipse.jface.text.contentassist.IContentAssistProcessor;
import org.eclipse.jface.util.PropertyChangeEvent;
import org.eclipse.jface.text.reconciler.DirtyRegion;
import org.eclipse.jface.util.SafeRunnable;
import org.eclipse.jface.text.source.IOverviewRuler;
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.viewers.ISelectionProvider;
import org.eclipse.jface.text.IInformationControlCreator;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.text.Position;
import org.eclipse.jface.text.GapTextStore;
import org.eclipse.jface.window.IShellProvider;
import org.eclipse.jface.text.source.ICharacterPairMatcher;
import org.eclipse.jface.text.reconciler.IReconcilableModel;
import org.eclipse.jface.text.source.IVerticalRuler;
import org.eclipse.jface.text.source.SourceViewerConfiguration;
import org.eclipse.jface.text.IDocumentRewriteSessionListener;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.jface.resource.JFaceResources;
import org.eclipse.swt.graphics.Image;
import org.eclipse.jface.text.DocumentPartitioningChangedEvent;
import org.eclipse.jface.text.ITextHoverExtension;
import org.eclipse.jface.text.DefaultLineTracker;
import org.eclipse.jface.text.source.Annotation;
import org.eclipse.jface.text.ITypedRegion;
import org.eclipse.jface.text.source.AnnotationModel;
import java.util.List;
import org.eclipse.jface.text.ISelectionValidator;
import org.eclipse.jface.text.contentassist.ICompletionProposalExtension;
import org.eclipse.jface.preference.PreferenceConverter;
import org.eclipse.jface.text.source.IAnnotationHover;
import org.eclipse.jface.text.IDocumentExtension4;
import org.eclipse.swt.graphics.RGB;
import org.eclipse.jface.text.IDocumentExtension;
import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.jface.text.IDocumentExtension3;
import org.eclipse.jface.viewers.IPostSelectionProvider;
import java.util.Iterator;
import org.eclipse.jface.text.ITextInputListener;
import org.eclipse.jface.viewers.SelectionChangedEvent;
import org.eclipse.jface.text.IDocumentListener;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.text.contentassist.ICompletionProposal;
import org.eclipse.jface.text.TextAttribute;
import org.eclipse.jface.text.ITextHover;
import org.eclipse.jface.text.reconciler.IReconcileResult;
import org.eclipse.jface.text.TextUtilities;
import org.eclipse.jface.text.IDocumentPartitioner;
import org.eclipse.jface.action.IContributionManager;
import java.util.Map;
import org.eclipse.jface.util.ListenerList;
import org.eclipse.jface.text.reconciler.IReconcilingStrategyExtension;
import org.eclipse.jface.text.reconciler.AbstractReconcileStep;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.text.IUndoManager;
import org.eclipse.jface.text.source.SourceViewer;
import org.eclipse.swt.graphics.Color;
import org.eclipse.jface.text.hyperlink.IHyperlinkPresenter;
import org.eclipse.jface.text.hyperlink.IHyperlinkDetector;
import org.eclipse.jface.text.ITextStore;
import org.eclipse.jface.text.contentassist.CompletionProposal;
import org.eclipse.jface.text.DefaultPositionUpdater;
import org.eclipse.jface.text.DocumentEvent;
import org.eclipse.jface.text.reconciler.IReconcilerExtension;
import org.eclipse.swt.custom.StyledText;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.swt.graphics.Device;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.contentassist.ContentAssistant;
import org.eclipse.swt.widgets.Control;
import org.eclipse.jface.text.source.IAnnotationModel;
import org.eclipse.jface.action.GroupMarker;
import org.eclipse.jface.text.IRegion;
import org.eclipse.jface.text.reconciler.IReconciler;
import org.eclipse.jface.text.contentassist.ICompletionProposalExtension2;
import org.eclipse.jface.text.information.IInformationPresenter;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.jface.text.information.InformationPresenter;
import org.eclipse.jface.text.source.IVerticalRulerInfo;
import org.eclipse.jface.text.contentassist.IContentAssistant;
import org.eclipse.jface.text.reconciler.IReconcilingStrategy;
import org.eclipse.jface.text.IAutoEditStrategy;
import org.eclipse.jface.action.Action;

public class AppCompletionProposal
implements ICompletionProposal, ICompletionProposalExtension, ICompletionProposalExtension2  {

      public  AppCompletionProposal() {
            CompletionProposal completionProposal = new CompletionProposal(image);
      }

      public Image getImage() {
            Image image1 = completionProposal.getImage();   // REPEATED!
      }

      public String getDisplayString() {
            String string1 = completionProposal.getDisplayString();     // REPEATED!
      }

      public String getAdditionalProposalInfo() {
            String string = completionProposal.getAdditionalProposalInfo();
      }

      public void unselected(sourceViewer) {
      }

      public void selected(sourceViewer) {
      }
}

public class AppCharacterPairMatcher
implements ICharacterPairMatcher  {

      public IRegion match(idocument) {
```

```
            char app_char = idocument.getChar(int);   // REPEATED!
      }
}

public class AppSelectionChangedListener
implements ISelectionChangedListener  {

      public void selectionChanged(SelectionChangedEvent) {
            ISelectionChangedListener.selectionChanged(SelectionChangedEvent);      // REPEATED!
            AppSafeRunnable appSafeRunnable = new AppSafeRunnable(appPostSelectionProvider && appSelectionValidator);
            Object[] objectArray = listenerList.getListeners();   // REPEATED!
      }
}

public class AppDocument
implements IDocument, IDocumentExtension3, IDocumentExtension4, IDocumentExtension  {

      public  AppDocument() {
            DefaultPositionUpdater defaultPositionUpdater = new DefaultPositionUpdater(String);
            DocumentEvent documentEvent = new DocumentEvent(appDocument);
            DefaultLineTracker defaultLineTracker = new DefaultLineTracker();
            AppTextStore appTextStore = new AppTextStore();
      }

      public Position[] getPositions(String) {
      }

      public long getModificationStamp() {
      }

      public void someMethod() {
            AppDocumentPartitioner appDocumentPartitioner = new AppDocumentPartitioner();
            IDocument idocument = sourceViewer.getDocument();
            DocumentPartitioningChangedEvent documentPartitioningChangedEvent = new DocumentPartitioningChangedEvent(appDocument);
            documentPartitioningChangedEvent.setPartitionChange(String,int,int);
            boolean app_boolean1 = IPreferenceStore.getBoolean(String); // REPEATED!
            /*                    Cyclic Statements                    */
            idocument.get(int,int):(String)||():(String);   // REPEATED!
            int app_int1 = gapTextStore.getLength();
            gapTextStore.get(int,int):(String)||(int):(char);     // REPEATED!
            char app_char = idocument.getChar(int);
            String string4 = appReconciler.getDocumentPartitioning();   // REPEATED!
            TextUtilities.computePartitioning(string4 && idocument && string8);     // REPEATED!
            IReconcilingStrategy ireconcilingStrategy = appReconciler.getReconcilingStrategy(string6);
            String string6 = ITypedRegion.getType();  // REPEATED!
            int app_int6 = dirtyRegion.getOffset();   // REPEATED!
            int app_int5 = dirtyRegion.getLength();   // REPEATED!
            int app_int2 = idocument.getLength();     // REPEATED!
      }
}

public class AppDocumentListener
implements IDocumentListener  {

      public  AppDocumentListener() {
            JFaceResources.getColorRegistry();
            AppContentAssistProcessor appContentAssistProcessor = new AppContentAssistProcessor(appReconcilingStrategyExtension && appDocumentListener);
            /*                    Cyclic Statements                    */
            String string5 = IPreferenceStore.getString(String);
            appDocument.get(int,int):(String)||():(String);
            IPreferenceStore.addPropertyChangeListener(appContentAssistProcessor);
            TextAttribute textAttribute = new TextAttribute(color2);
            IDocument.addDocumentListener(appDocumentListener);
      }

      public void documentChanged(DocumentEvent) {
            String string4 = appReconciler.getDocumentPartitioning();
            TextUtilities.computePartitioning(string4 && idocument && string8);
            StyledText styledText = sourceViewer.getTextWidget(); // REPEATED!
            String string6 = ITypedRegion.getType();
            /*                    Cyclic Statements                    */
            boolean app_boolean = textAttribute.equals(textAttribute);  // REPEATED!
            appDocument.get(int,int):(String)||():(String); // REPEATED!
            int app_int7 = DocumentEvent.getLength(); // REPEATED!
            int app_int = textAttribute.getStyle();   // REPEATED!
            DirtyRegion dirtyRegion = new DirtyRegion();
            String string7 = DocumentEvent.getText(); // REPEATED!
            Color color1 = textAttribute.getForeground();   // REPEATED!
            Color color = textAttribute.getBackground();    // REPEATED!
            int app_int8 = DocumentEvent.getOffset(); // REPEATED!
            int app_int13 = idocument.getLineOffset(int);
            int app_int9 = idocument.getLineOfOffset(int);
            int app_int2 = appDocument.getLength();   // REPEATED!
      }

      public void documentAboutToBeChanged(DocumentEvent) {
            String string6 = ITypedRegion.getType();
            String string4 = appReconciler.getDocumentPartitioning();
            TextUtilities.computePartitioning(string4 && idocument && string8);
            int app_int2 = appDocument.getLength();
            /*                    Cyclic Statements                    */
            int app_int8 = DocumentEvent.getOffset();
            int app_int7 = DocumentEvent.getLength();
      }
}

public class AppContentAssistProcessor
implements IContentAssistProcessor  {

      public  AppContentAssistProcessor() {
            /*                    Cyclic Statements                    */
            String string5 = IPreferenceStore.getString(String);
            boolean app_boolean1 = IPreferenceStore.getBoolean(String);
            IPreferenceStore.addPropertyChangeListener(appContentAssistProcessor);
      }

      public ICompletionProposal[] computeCompletionProposals(sourceViewer) {
            ImageDescriptor.createImage(Device):(Image)||(boolean):(Image)||():(Image)||(boolean,Device):(Image); // REPEATED!
            String string4 = appReconciler.getDocumentPartitioning();
            JFaceResources.getImageRegistry();  // REPEATED!
            ImageDescriptor.getMissingImageDescriptor();    // REPEATED!
            /*                    Cyclic Statements                    */
            String string1 = appCompletionProposal.getDisplayString();  // REPEATED!
            idocument.get(int,int):(String)||():(String);
            appReconcilingStrategy.reconcile(dirtyRegion);
            DirtyRegion dirtyRegion = new DirtyRegion();
            IDocument idocument = sourceViewer.getDocument();     // REPEATED!
            char app_char = idocument.getChar(int);
            TextUtilities.computePartitioning(string4 && idocument && string8);
            Iterator iterator = iannotationModel.getAnnotationIterator();
            String string6 = ITypedRegion.getType();
            int app_int6 = dirtyRegion.getOffset();
            int app_int5 = dirtyRegion.getLength();
```

```
            IAnnotationModel iannotationModel = sourceViewer.getAnnotationModel();  // REPEATED!
            int app_int2 = idocument.getLength();     // REPEATED!
            AppCompletionProposal appCompletionProposal = new AppCompletionProposal(image);
      }

      public char[] getContextInformationAutoActivationCharacters() {
      }

      public char[] getCompletionProposalAutoActivationCharacters() {
      }

      public String getErrorMessage() {
      }
}

public class AppAbstractReconcileStep
extends AbstractReconcileStep {

      public IReconcileResult[] reconcileModel(dirtyRegion) {
            /*                    Cyclic Statements                    */
            IReconcilableModel ireconcilableModel = appAbstractReconcileStep.getModel();  // REPEATED!
            IProgressMonitor iprogressMonitor = appAbstractReconcileStep.getProgressMonitor();  // REPEATED!
      }

      public void someMethod() {
            IReconcilableModel ireconcilableModel1 = appAbstractReconcileStep.getInputModel();
      }
}

public class AppAnnotationModel
extends AnnotationModel {

      public void removeAnnotations(List,boolean,boolean) {
      }

      public void connected() {
      }
}

public class AppSourceViewerConfiguration
extends SourceViewerConfiguration {

      public IInformationPresenter getInformationPresenter(sourceViewer) {
            String string8 = appSourceViewerConfiguration.getConfiguredDocumentPartitioning(sourceViewer);
            AppInformationControlCreator appInformationControlCreator = new AppInformationControlCreator(appSourceViewerConfiguration);
            InformationPresenter informationPresenter = new InformationPresenter(appInformationControlCreator);
            informationPresenter.setSizeConstraints(int,int,boolean,boolean);
            informationPresenter.setDocumentPartitioning(string8);
      }

      public int getTabWidth(sourceViewer) {
      }

      public void getTextHover(sourceViewer) {
            AppTextHoverExtension appTextHoverExtension = new AppTextHoverExtension();
            AppTextHover appTextHover = new AppTextHover();
      }

      public IReconciler getReconciler(sourceViewer) {
            AppReconciler appReconciler = new AppReconciler(appSourceViewerConfiguration);
      }

      public String getConfiguredDocumentPartitioning(ISourceViewer) {
      }

      public String[] getIndentPrefixes(sourceViewer) {
            int app_int12 = IPreferenceStore.getInt(String);
            String string5 = IPreferenceStore.getString(String);
      }

      public IContentAssistant getContentAssistant(sourceViewer) {
            ContentAssistant contentAssistant = new ContentAssistant();
            AppContentAssistProcessor appContentAssistProcessor = new AppContentAssistProcessor(appReconcilingStrategyExtension && appDocumentListener);
            /*                    Cyclic Statements                    */
            contentAssistant.setInformationControlCreator(appInformationControlCreator && iinformationControlCreator);
            contentAssistant.setDocumentPartitioning(string8);
            contentAssistant.setContextInformationPopupOrientation(int);
            String string8 = appSourceViewerConfiguration.getConfiguredDocumentPartitioning(sourceViewer);
            contentAssistant.setAutoActivationDelay(int);
            contentAssistant.setProposalPopupOrientation(int);
            IInformationControlCreator iinformationControlCreator = appSourceViewerConfiguration.getInformationControlCreator(sourceViewer);
            contentAssistant.setContentAssistProcessor(appContentAssistProcessor);
            contentAssistant.enableAutoActivation(boolean);
      }

      public IAutoEditStrategy[] getAutoEditStrategies(sourceViewer) {
            AppAutoEditStrategy appAutoEditStrategy = new AppAutoEditStrategy();
      }

      public IAnnotationHover getAnnotationHover(sourceViewer) {
            AppAnnotationHover appAnnotationHover = new AppAnnotationHover();
      }

      public IAnnotationHover getOverviewRulerAnnotationHover(sourceViewer) {
      }

      public IInformationControlCreator getInformationControlCreator(sourceViewer) {
            AppInformationControlCreator appInformationControlCreator = new AppInformationControlCreator(appSourceViewerConfiguration);
      }

      public IHyperlinkPresenter getHyperlinkPresenter(sourceViewer) {
      }

      public int getHyperlinkStateMask(sourceViewer) {
      }

      public int[] getConfiguredTextHoverStateMasks(sourceViewer) {
      }

      public IUndoManager getUndoManager(sourceViewer) {
      }

      public IHyperlinkDetector[] getHyperlinkDetectors(sourceViewer) {
      }
}

public class AppReconciler
implements IReconciler, IReconcilerExtension  {

      public  AppReconciler() {
            AppDocumentRewriteSessionListener appDocumentRewriteSessionListener = new AppDocumentRewriteSessionListener(ireconciler && appReconciler);
            AppDocumentListener appDocumentListener = new AppDocumentListener(ireconciler && appReconciler);
      }
```

```
      public void install(sourceViewer) {
            AppTextInputListener appTextInputListener = new AppTextInputListener(ireconciler && appReconciler);
            sourceViewer.addTextInputListener(appTextInputListener);
      }
}

public class AppTextInputListener
implements ITextInputListener  {

      public void inputDocumentAboutToBeChanged(IDocument,IDocument) {
      }

      public void inputDocumentChanged(IDocument,IDocument) {
            AppReconcilingStrategyExtension appReconcilingStrategyExtension = new AppReconcilingStrategyExtension(sourceViewer);
            IDocument idocument = sourceViewer.getDocument();
            AppReconcilingStrategy appReconcilingStrategy = new AppReconcilingStrategy(sourceViewer);
            appReconcilingStrategy.setDocument(idocument);  // REPEATED!
            /*                      Cyclic Statements                   */
            idocument.get(int,int):(String)||():(String);
            idocument.removeDocumentRewriteSessionListener(appDocumentRewriteSessionListener);
            DirtyRegion dirtyRegion = new DirtyRegion();
            IDocument.addDocumentListener(appDocumentListener);
            int app_int2 = idocument.getLength();     // REPEATED!
            idocument.addDocumentRewriteSessionListener(appDocumentRewriteSessionListener);     // REPEATED!
      }
}

public class AppAutoEditStrategy
implements IAutoEditStrategy  {

      public void customizeDocumentCommand(idocument) {
      }
}

public class AppSafeRunnable
extends SafeRunnable {
}

public class AppAction
extends Action {

      public  AppAction() {
            appAction.setEnabled(boolean);
            appAction.setText(String);
            Control control = IVerticalRulerInfo.getControl();
      }
}

public class AppInformationControlCreator
implements IInformationControlCreator  {
}

public class AppDocumentPartitioner
implements IDocumentPartitioner  {

      public  AppDocumentPartitioner() {
            AppRegion appRegion = new AppRegion();
            boolean app_boolean1 = IPreferenceStore.getBoolean(String);
            AppTypedRegion appTypedRegion = new AppTypedRegion();
      }
}

public class AppTextStore
implements ITextStore  {

      public  AppTextStore() {
            GapTextStore gapTextStore = new GapTextStore(int,int);
      }

      public void someMethod() {
            /*                      Cyclic Statements                   */
            gapTextStore.replace(int,int,String);
            int app_int1 = gapTextStore.getLength();
            gapTextStore.get(int,int):(String)||(int):(char);     // REPEATED!
      }
}

public class AppReconcilingStrategyExtension
implements IReconcilingStrategyExtension  {

      public  AppReconcilingStrategyExtension() {
            AppAbstractReconcileStep appAbstractReconcileStep = new AppAbstractReconcileStep(appReconcilingStrategyExtension);
            AppContentAssistProcessor appContentAssistProcessor = new AppContentAssistProcessor(appReconcilingStrategyExtension && appDocumentListener);
      }
}

public class AppReconcilingStrategy
implements IReconcilingStrategy  {

      public  AppReconcilingStrategy() {
            AppAbstractReconcileStep appAbstractReconcileStep = new AppAbstractReconcileStep(appReconcilingStrategyExtension);
            AppContentAssistProcessor appContentAssistProcessor = new AppContentAssistProcessor(appReconcilingStrategyExtension && appDocumentListener);
      }

      public void someMethod() {
            AppReconcilableModel appReconcilableModel = new AppReconcilableModel(idocument);
            appAbstractReconcileStep.setInputModel(appReconcilableModel);
            appAbstractReconcileStep.reconcile(dirtyRegion);
            IPreferenceStore.addPropertyChangeListener(appContentAssistProcessor);
            IAnnotationModel iannotationModel = sourceViewer.getAnnotationModel();  // REPEATED!
            /*                      Cyclic Statements                   */
            Iterator iterator = iannotationModel.getAnnotationIterator();
            iannotationModel.replaceAnnotations(Annotation[],Map);
      }
}

public class AppRegion
implements IRegion  {
}

public class AppTypedRegion
implements ITypedRegion  {
}

public class AppShellProvider
implements IShellProvider  {
}

public class AppDocumentRewriteSessionListener
implements IDocumentRewriteSessionListener  {
}
```

```
public class AppAnnotationHover
implements IAnnotationHover  {
}

public class AppTextHover
implements ITextHover  {
}

public class AppTextHoverExtension
implements ITextHoverExtension  {
}

public class AppPostSelectionProvider
implements IPostSelectionProvider  {

      public  AppPostSelectionProvider() {
            ListenerList listenerList = new ListenerList(int)||();
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(appPostSelectionProvider && appSelectionValidator);
            ISelectionProvider.addSelectionChangedListener(appSelectionChangedListener);
            IPostSelectionProvider.addPostSelectionChangedListener(appSelectionChangedListener);
      }
}

public class AppSelectionValidator
implements ISelectionValidator  {

      public  AppSelectionValidator() {
            ListenerList listenerList = new ListenerList(int)||();
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(appPostSelectionProvider && appSelectionValidator);
            ISelectionProvider.addSelectionChangedListener(appSelectionChangedListener);
            IPostSelectionProvider.addPostSelectionChangedListener(appSelectionChangedListener);
      }
}

public class AppReconcilableModel
implements IReconcilableModel  {
}

public class SomeClass {

      public void someMethod() {
            SourceViewer sourceViewer = new SourceViewer(Composite,IVerticalRuler,int)||(Composite,IVerticalRuler,IOverviewRuler,boolean,int);
            AppReconcilingStrategyExtension appReconcilingStrategyExtension = new AppReconcilingStrategyExtension(sourceViewer);
            AppReconcilingStrategy appReconcilingStrategy = new AppReconcilingStrategy(sourceViewer);
            appReconcilingStrategy.setDocument(idocument);
            ImageDescriptor.createImage(Device):(Image)||(boolean):(Image)||():(Image)||(boolean,Device):(Image);
            ISelection iselection = ISelectionProvider.getSelection();  // REPEATED!
            int app_int4 = iselection.getOffset();     // REPEATED!
            AppPostSelectionProvider appPostSelectionProvider = new AppPostSelectionProvider();
            AppTextStore appTextStore = new AppTextStore();
            AppDocument appDocument = new AppDocument(appTextStore);
            appDocumentPartitioner.connect(appDocument);
            appDocument.setDocumentPartitioner(String,IDocumentPartitioner);
            DocumentEvent documentEvent = new DocumentEvent(appDocument);     // REPEATED!
            appDocumentPartitioner.documentAboutToBeChanged(documentEvent);
            boolean app_boolean3 = appDocumentPartitioner.documentChanged(documentEvent);
            defaultPositionUpdater.update(documentEvent);   // REPEATED!
            AppSourceViewerConfiguration appSourceViewerConfiguration = new AppSourceViewerConfiguration();
            AppShellProvider appShellProvider = new AppShellProvider();
            AppDocumentListener appDocumentListener = new AppDocumentListener(ireconciler && appReconciler);
            MenuManager menuManager = new MenuManager(String)||(String,String)||(); // REPEATED!
            PreferenceConverter.setDefault();
            GroupMarker groupMarker = new GroupMarker(String);    // REPEATED!
            defaultLineTracker.replace(int,int,String);
            int app_int3 = defaultLineTracker.getLineOffset(int); // REPEATED!
            AppAnnotationModel appAnnotationModel = new AppAnnotationModel();
            IMenuManager imenuManager = IMenuManager.findMenuUsingPath(String);
            IContributionManager.insertAfter(menuManager);
            AppCharacterPairMatcher appCharacterPairMatcher = new AppCharacterPairMatcher();
            Separator separator = new Separator(String)||();      // REPEATED!
            imenuManager.add(groupMarker && separator);     // REPEATED!
            Action.findModifierString();
            AppAction appAction = new AppAction();
            appAction.setActionDefinitionId(String);  // REPEATED!
            boolean app_boolean2 = appAction.isEnabled();   // REPEATED!
            AppSelectionValidator appSelectionValidator = new AppSelectionValidator();
            listenerList.add(Object);     // REPEATED!
            IPreferenceStore.setDefault(String,String):()||(String,boolean):()||(String,float):()||(String,double):()||(String,long):()||(String,int):();   // REPEATED!
            boolean app_boolean1 = IPreferenceStore.getBoolean(String); // REPEATED!
            String string2 = PropertyChangeEvent.getProperty();   // REPEATED!
            /*                    Cyclic Statements                  */
            appTextStore.replace(int,int,String);
            boolean app_boolean = textAttribute.equals(textAttribute);  // REPEATED!
            TextUtilities.computePartitioning(string4 && idocument && string8);     // REPEATED!
            Color color = textAttribute.getBackground();    // REPEATED!
            IReconcilingStrategy ireconcilingStrategy = appReconciler.getReconcilingStrategy(string6);
            int app_int6 = dirtyRegion.getOffset();   // REPEATED!
            int app_int5 = dirtyRegion.getLength();   // REPEATED!
            appDocument.get(int,int):(String)||():(String); // REPEATED!
            appDocument.stopPostNotificationProcessing();   // REPEATED!
            int app_int = textAttribute.getStyle();   // REPEATED!
            appDocument.resumePostNotificationProcessing(); // REPEATED!
            Iterator iterator = iannotationModel.getAnnotationIterator();
            String string6 = ITypedRegion.getType();  // REPEATED!
            appReconcilingStrategy.reconcile(dirtyRegion);
            int app_int1 = appTextStore.getLength();
            appTextStore.get(int,int):(String)||(int):(char);     // REPEATED!
            DirtyRegion dirtyRegion = new DirtyRegion();
            String string4 = appReconciler.getDocumentPartitioning();   // REPEATED!
            StyledText styledText = sourceViewer.getTextWidget(); // REPEATED!
            Color color1 = textAttribute.getForeground();   // REPEATED!
            IAnnotationModel iannotationModel = sourceViewer.getAnnotationModel();  // REPEATED!
            int app_int9 = idocument.getLineOfOffset(int);  // REPEATED!
            int app_int2 = appDocument.getLength();   // REPEATED!
      }
}
```

Description of False Negatives:

We have one false negative because of not calling the following instruction since it is in org.eclipse.ui, not in JFace:
`TextEditor.setSourceViewerConfiguration(appSourceViewerConfiguration);`