JFace – Toolbar Button – One Trace
Use Full Trace (No Slicing)

Supporting Applications : [JDT]

```
import org.eclipse.jface.util.PropertyChangeEvent;
import org.eclipse.swt.graphics.FontData;
import org.eclipse.jface.util.SafeRunnable;
import org.eclipse.jface.util.TransferDragSourceListener;
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.action.IMenuListener;
import org.eclipse.jface.util.TransferDropTargetListener;
import org.eclipse.jface.text.source.IAnnotationModelListener;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.swt.widgets.Item;
import org.eclipse.jface.window.IShellProvider;
import org.eclipse.jface.viewers.ILabelProvider;
import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.jface.resource.JFaceResources;
import org.eclipse.swt.widgets.ToolBar;
import org.eclipse.jface.action.ContributionItem;
import org.eclipse.jface.viewers.IOpenListener;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.graphics.Point;
import org.eclipse.jface.viewers.TreePath;
import org.eclipse.swt.widgets.CoolBar;
import java.util.EventObject;
import org.eclipse.jface.dialogs.TrayDialog;
import java.util.List;
import org.eclipse.jface.preference.PreferenceConverter;
import org.eclipse.jface.preference.IPreferenceStore;
import java.util.Iterator;
import org.eclipse.jface.dialogs.IDialogSettings;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.action.ActionContributionItem;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.swt.widgets.Table;
import org.eclipse.jface.viewers.LabelProviderChangedEvent;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.jface.action.IContributionManager;
import org.eclipse.jface.viewers.ILightweightLabelDecorator;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.DecoratingLabelProvider;
import org.eclipse.swt.widgets.Widget;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.resource.CompositeImageDescriptor;
import org.eclipse.jface.viewers.IColorProvider;
import org.eclipse.jface.action.IContributionItem;
import org.eclipse.jface.viewers.ILabelProviderListener;
import org.eclipse.jface.util.Assert;
import org.eclipse.swt.widgets.Button;
import org.eclipse.jface.viewers.CheckboxTableViewer;
import org.eclipse.swt.graphics.Color;
import org.eclipse.jface.viewers.IDecorationContext;
import org.eclipse.swt.graphics.FontMetrics;
import org.eclipse.jface.viewers.ICheckStateListener;
import org.eclipse.swt.widgets.Tree;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.ViewerFilter;
import org.eclipse.jface.viewers.ILabelDecorator;
import org.eclipse.swt.graphics.Device;
import org.eclipse.swt.dnd.DropTargetListener;
import org.eclipse.swt.widgets.Control;
import org.eclipse.jface.text.source.IAnnotationModelListenerExtension;
import org.eclipse.swt.dnd.DragSourceListener;
import org.eclipse.swt.dnd.Transfer;
import org.eclipse.jface.action.GroupMarker;
import org.eclipse.jface.viewers.IElementComparer;
import org.eclipse.swt.graphics.ImageData;
import org.eclipse.jface.viewers.ViewerSorter;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.jface.viewers.ITreeContentProvider;
import org.eclipse.jface.viewers.ArrayContentProvider;
import org.eclipse.jface.viewers.ITreeViewerListener;
import org.eclipse.jface.action.IMenuCreator;
import org.eclipse.jface.action.Action;

public class AppLabelProvider
implements IPropertyChangeListener, IColorProvider
extends LabelProvider {

      public  AppLabelProvider() {
            AppLabelProvider appLabelProvider = new AppLabelProvider(appTreeContentProvider && appLabelProvider);
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
            AppLightweightLabelDecorator appLightweightLabelDecorator = new AppLightweightLabelDecorator();
            AppLabelDecorator appLabelDecorator = new AppLabelDecorator();
            /*                      Cyclic Statements                   */
            IPreferenceStore.addPropertyChangeListener(appLabelProvider);
            boolean app_boolean4 = IPreferenceStore.getBoolean(String);
            appDecoratingLabelProvider.setDecorationContext(IDecorationContext);
      }

      public String getText(Object) {
            String string2 = appLabelDecorator.decorateText(string1);   // REPEATED!
      }

      public void removeListener(ILabelProviderListener) {
            appLabelDecorator.removeListener(ILabelProviderListener);
      }

      public void dispose() {
            IPreferenceStore.removePropertyChangeListener(appLabelProvider && ilabelProvider);
            appLabelProvider.dispose();   // REPEATED!
      }

      public void addListener(ILabelProviderListener) {
            AppAnnotationModelListenerExtension appAnnotationModelListenerExtension = new AppAnnotationModelListenerExtension();
            AppAnnotationModelListener appAnnotationModelListener = new AppAnnotationModelListener();
            appLabelDecorator.addListener(ILabelProviderListener);
```

```
            AppLabelProvider appLabelProvider = new AppLabelProvider(appTreeContentProvider && appLabelProvider);
            IPreferenceStore.addPropertyChangeListener(appLabelProvider);
      }

      public Image getImage(Object) {
            Assert.isTrue();  // REPEATED!
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
            AppCompositeImageDescriptor appCompositeImageDescriptor = new AppCompositeImageDescriptor(point && point1);
            appCompositeImageDescriptor.createImage(Device):(Image)||(boolean):(Image)||():(Image)||(boolean,Device):(Image); // REPEATED!
            Image image1 = appLabelDecorator.decorateImage(image);      // REPEATED!
      }

      public void someMethod() {
            AppLabelProvider appLabelProvider = new AppLabelProvider(appTreeContentProvider && appLabelProvider);
            AppLabelDecorator appLabelDecorator = new AppLabelDecorator();
            AppLightweightLabelDecorator appLightweightLabelDecorator = new AppLightweightLabelDecorator();
            /*                    Cyclic Statements                    */
            IPreferenceStore.addPropertyChangeListener(appLabelProvider);     // REPEATED!
            boolean app_boolean4 = IPreferenceStore.getBoolean(String); // REPEATED!
      }
}

public class AppTreeViewer
extends TreeViewer {

      public void handleLabelProviderChanged(LabelProviderChangedEvent) {
            Object[] objectArray5 = LabelProviderChangedEvent.getElements();
            Object object1 = EventObject.getSource();
            LabelProviderChangedEvent labelProviderChangedEvent = new LabelProviderChangedEvent(object1);
      }

      public void internalRefresh(control && object && tree) {
      }

      public void doUpdateItem(Widget,Object,boolean) {
      }

      public void unmapAllElements() {
      }

      public void mapElement(Object,Widget) {
      }

      public void isExpandable((Object):(boolean)||(Item,TreePath,Object):(boolean)) {
            boolean app_boolean2 = appTreeViewer.hasFilters();    // REPEATED!
      }

      public Object[] getFilteredChildren(object) {
            boolean app_boolean = appViewerFilter.select(appTreeViewer && object);  // REPEATED!
            /*                    Cyclic Statements                    */
            Object[] objectArray = appTreeViewer.getRawChildren(object);      // REPEATED!
            boolean app_boolean2 = appTreeViewer.hasFilters();    // REPEATED!
            ViewerFilter[] viewerFilterArray = appTreeViewer.getFilters();    // REPEATED!
      }

      public void preservingSelection(Runnable) {
      }

      public void setSelectionToWidget((ISelection,boolean):()||(List,boolean):()) {
      }
}

public class AppDecoratingLabelProvider
extends DecoratingLabelProvider {

      public  AppDecoratingLabelProvider() {
            appDecoratingLabelProvider.setDecorationContext(IDecorationContext);
      }

      public Color getBackground(Object) {
            ILabelProvider ilabelProvider = appDecoratingLabelProvider.getLabelProvider();      // REPEATED!
            Color color = ilabelProvider.getBackground(Object);   // REPEATED!
      }

      public Color getForeground(Object) {
            ILabelProvider ilabelProvider = appDecoratingLabelProvider.getLabelProvider();      // REPEATED!
            Color color1 = ilabelProvider.getForeground(Object);  // REPEATED!
      }
}

public class AppTreeContentProvider
implements ITreeContentProvider  {

      public  AppTreeContentProvider() {
            AppLabelProvider appLabelProvider = new AppLabelProvider(appTreeContentProvider && appLabelProvider);
      }

      public Object[] getElements(object) {
            Object[] objectArray4 = appTreeContentProvider.getChildren(object);     // REPEATED!
      }

      public void inputChanged(appTreeViewer && object) {
      }

      public void dispose() {
            IPreferenceStore.removePropertyChangeListener(appLabelProvider && ilabelProvider);
      }

      public boolean hasChildren(Object) {
      }
}

public class AppViewerSorter
extends ViewerSorter {

      public int compare(appTreeViewer) {
            int app_int1 = appViewerSorter.category(Object);      // REPEATED!
      }
}

public class AppContributionItem
extends ContributionItem {

      public  AppContributionItem() {
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
      }
```

```
      public String getId() {
      }

      public void fill((Menu,int):()||(ToolBar,int):()||(CoolBar,int):()||(Composite):()) {
            int app_int4 = appAction.getStyle();      // REPEATED!
            ImageDescriptor imageDescriptor = appAction.getImageDescriptor(); // REPEATED!
            String string3 = appSelectionChangedListener.getText();     // REPEATED!
            imageDescriptor.createImage(Device):(Image)||(boolean):(Image)||():(Image)||(boolean,Device):(Image); // REPEATED!
            Assert.isTrue();  // REPEATED!
      }

      public void dispose() {
      }

      public boolean isDynamic() {
      }
}

public class AppAction
extends Action {

      public  AppAction() {
            ISelection iselection = appTreeViewer.getSelection();
            List list = iselection.toList();
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
            Shell shell = IShellProvider.getShell();
            ImageDescriptor.createFromURL();
            AppImageDescriptor appImageDescriptor = new AppImageDescriptor();
            ImageDescriptor.getMissingImageDescriptor();
            /*                      Cyclic Statements                  */
            String string4 = IAction.getToolTipText();
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(appTrayDialog && appAction);
            String string3 = appAction.getText();
            appAction.setText(string3);
            appAction.setActionDefinitionId(String);
            appAction.setDescription(String);
            appAction.setToolTipText(string4);
            appAction.setChecked(boolean);
            appAction.setImageDescriptor();
            appAction.setEnabled(boolean);
            appAction.setHoverImageDescriptor();
            appAction.setDisabledImageDescriptor();
      }

      public void run() {
            AppTrayDialog appTrayDialog = new AppTrayDialog(shell);
            int app_int6 = appTrayDialog.open();      // REPEATED!
            /*                      Cyclic Statements                  */
            Control control = appTreeViewer.getControl();   // REPEATED!
            appTreeViewer.collapseToLevel(object);    // REPEATED!
            Object object = appTreeViewer.getInput(); // REPEATED!
      }
}

public class AppTrayDialog
extends TrayDialog {

      public  AppTrayDialog() {
            int app_int5 = appTrayDialog.getShellStyle();
            appTrayDialog.setShellStyle(int);
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
      }

      public void createButtonsForButtonBar(Composite) {
      }

      public void configureShell(Shell) {
      }

      public int getDialogBoundsStrategy() {
      }

      public Control createDialogArea(Composite) {
            AppLabelProvider appLabelProvider = new AppLabelProvider(appTreeContentProvider && appLabelProvider);
            AppCheckStateListener appCheckStateListener = new AppCheckStateListener(appTrayDialog);
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(appTrayDialog && appAction);
            CheckboxTableViewer.newCheckList();
            Dialog.applyDialogFont(control1);
            ArrayContentProvider arrayContentProvider = new ArrayContentProvider();
            appTrayDialog.initializeDialogUnits(control1);
            Dialog.convertHorizontalDLUsToPixels(int):(int)||(FontMetrics,int):(int);     // REPEATED!
            JFaceResources.getDialogFont();     // REPEATED!
            /*                      Cyclic Statements                  */
            appTrayDialog.convertVerticalDLUsToPixels(FontMetrics,int):(int)||(int):(int);      // REPEATED!
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button); // REPEATED!
            appTrayDialog.convertWidthInCharsToPixels(int):(int)||(FontMetrics,int):(int);
            appTrayDialog.convertHorizontalDLUsToPixels(int):(int)||(FontMetrics,int):(int);    // REPEATED!
            Button button = appTrayDialog.createButton(Composite,int,String,boolean);     // REPEATED!
            appTrayDialog.convertHeightInCharsToPixels(int):(int)||(FontMetrics,int):(int);
      }

      public IDialogSettings getDialogBoundsSettings() {
      }
}

public class AppCompositeImageDescriptor
extends CompositeImageDescriptor {

      public  AppCompositeImageDescriptor() {
            Assert.isTrue();
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
      }

      public void drawCompositeImage(int,int) {
            ImageData imageData = ImageDescriptor.getImageData(); // REPEATED!
            appCompositeImageDescriptor.drawImage(imageData);     // REPEATED!
            Point point = appCompositeImageDescriptor.getSize();  // REPEATED!
      }

      public ImageData getImageData() {
      }

      public Point getSize() {
      }
}
```

```java
public class AppMenuListener
implements IMenuListener  {

      public void menuAboutToShow(IMenuManager) {
            AppContributionItem appContributionItem = new AppContributionItem();
            IContributionManager.insertBefore(appContributionItem);     // REPEATED!
      }
}

public class AppActionA
implements IAction  {

      public boolean isEnabled() {
      }
}

public class AppSelectionChangedListener
implements ISelectionChangedListener  {

      public  AppSelectionChangedListener() {
            ISelection iselection = appTreeViewer.getSelection();
            List list = iselection.toList();
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
            Shell shell = IShellProvider.getShell();
            ImageDescriptor.createFromURL();
            AppImageDescriptor appImageDescriptor = new AppImageDescriptor();
            ImageDescriptor.getMissingImageDescriptor();
            /*                      Cyclic Statements                 */
            String string4 = IAction.getToolTipText();
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(appTrayDialog && appAction);
            String string3 = appAction.getText();
            appAction.setText(string3);
            appAction.setDescription(String);
            appAction.setToolTipText(string4);
            appAction.setImageDescriptor();
            AppAction appAction = new AppAction(appSelectionChangedListener);
            appAction.setHoverImageDescriptor();
            appAction.setEnabled(boolean);
            appAction.setDisabledImageDescriptor();
      }
}

public class AppSafeRunnable
extends SafeRunnable {
}

public class AppImageDescriptor
extends ImageDescriptor {
}

public class AppransferDragSourceListener
implements TransferDragSourceListener  {

      public  AppransferDragSourceListener() {
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
      }
}

public class AppransferDropTargetListener
implements TransferDropTargetListener  {

      public  AppransferDropTargetListener() {
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button);
      }
}

public class AppViewerFilter
extends ViewerFilter {
}

public class AppAnnotationModelListener
implements IAnnotationModelListener  {
}

public class AppAnnotationModelListenerExtension
implements IAnnotationModelListenerExtension  {
}

public class AppCheckStateListener
implements ICheckStateListener  {
}

public class AppLightweightLabelDecorator
implements ILightweightLabelDecorator  {
}

public class AppLabelDecorator
implements ILabelDecorator  {
}

public class AppDoubleClickListener
implements IDoubleClickListener  {
}

public class AppOpenListener
implements IOpenListener  {
}

public class AppTreeViewerListener
implements ITreeViewerListener  {
}

public class SomeClass {

      public void someMethod() {
            AppTreeViewerListener appTreeViewerListener = new AppTreeViewerListener();
            ImageDescriptor.createFromURL();    // REPEATED!
            MenuManager menuManager = new MenuManager(String)||(String,String)||(); // REPEATED!
            menuManager.setRemoveAllWhenShown(boolean);
            PreferenceConverter.setDefault();
            AppOpenListener appOpenListener = new AppOpenListener();
            GroupMarker groupMarker = new GroupMarker(String);
            AppContributionItem appContributionItem = new AppContributionItem();
            AppViewerSorter appViewerSorter = new AppViewerSorter();
            AppTreeContentProvider appTreeContentProvider = new AppTreeContentProvider();
```

```
            AppDoubleClickListener appDoubleClickListener = new AppDoubleClickListener();
            JFaceResources.getFontRegistry();   // REPEATED!
            PreferenceConverter.putValue(fontDataArray);
            AppTreeViewer appTreeViewer = new AppTreeViewer();
            appTreeViewer.setUseHashlookup(boolean);
            appTreeViewer.addDragSupport(int,Transfer[],DragSourceListener);
            appTreeViewer.addDropSupport(int,Transfer[],DropTargetListener);
            appTreeViewer.setComparer(IElementComparer);
            AppransferDropTargetListener appransferDropTargetListener = new AppransferDropTargetListener(appTreeViewer);
            AppransferDragSourceListener appransferDragSourceListener = new AppransferDragSourceListener(appTreeViewer);
            IDialogSettings idialogSettings1 = IDialogSettings.getSection(String);
            idialogSettings1.put(String,int):()||(String,long):()||(String,boolean):()||(String,float):()||(String,double):()||(String,String):()||(String,String[]):();
            AppActionA appActionA = new AppActionA();
            Separator separator = new Separator(String)||();      // REPEATED!
            Shell shell = IShellProvider.getShell();  // REPEATED!
            Action.findModifierString();
            AppMenuListener appMenuListener = new AppMenuListener();
            Assert.isTrue();  // REPEATED!
            AppSafeRunnable appSafeRunnable = new AppSafeRunnable();
            AppViewerFilter appViewerFilter = new AppViewerFilter();
            String string = PropertyChangeEvent.getProperty();
            IAction.setMenuCreator(IMenuCreator);     // REPEATED!
            /*                      Cyclic Statements                */
            String string6 = IPreferenceStore.getString(String);
            boolean app_boolean9 = IPreferenceStore.contains(String);   // REPEATED!
            appTreeViewer.addTreeListener(appTreeViewerListener);
            appTreeViewer.addOpenListener(appOpenListener);
            Iterator iterator = iselection.iterator();
            appTreeViewer.addFilter(appViewerFilter); // REPEATED!
            ISelection iselection = appTreeViewer.getSelection(); // REPEATED!
            IContributionItem icontributionItem = IContributionManager.find(String);
            appSelectionChangedListener.setDescription(String);   // REPEATED!
            actionContributionItem.fill(Menu,int):()||(ToolBar,int):()||(CoolBar,int):()||(Composite):();   // REPEATED!
            appTreeViewer.setLabelProvider(appLabelProvider && appDecoratingLabelProvider);
            int app_int = iselection.size();    // REPEATED!
            boolean app_boolean3 = appSelectionChangedListener.isEnabled();   // REPEATED!
            appTreeViewer.removeSelectionChangedListener(appAction && appSelectionChangedListener);   // REPEATED!
            appTreeViewer.setInput(Object);
            appSelectionChangedListener.setEnabled(boolean);      // REPEATED!
            appSelectionChangedListener.setDisabledImageDescriptor();   // REPEATED!
            IPreferenceStore.setValue(String,double):()||(String,String):()||(String,float):()||(String,int):()||(String,long):()||(String,boolean):();     // REPEATED!
            IPreferenceStore.addPropertyChangeListener(appLabelProvider);     // REPEATED!
            IPreferenceStore.setToDefault(String);
            AppDecoratingLabelProvider appDecoratingLabelProvider = new AppDecoratingLabelProvider(appLabelProvider);
            AppLabelProvider appLabelProvider = new AppLabelProvider(appTreeContentProvider && appLabelProvider);
            menuManager.addMenuListener(appMenuListener);   // REPEATED!
            appAction.setChecked(boolean);      // REPEATED!
            appTreeViewer.addSelectionChangedListener(appAction && appSelectionChangedListener);      // REPEATED!
            appSelectionChangedListener.setHoverImageDescriptor();      // REPEATED!
            IPreferenceStore.removePropertyChangeListener(appLabelProvider && ilabelProvider);
            .addListener(appLabelProvider);
            List list = iselection.toList();    // REPEATED!
            Menu menu = menuManager.createContextMenu(tree);
            boolean app_boolean1 = iselection.isEmpty();    // REPEATED!
            appTreeViewer.addPostSelectionChangedListener(appSelectionChangedListener);
            appSelectionChangedListener.setText(string3);   // REPEATED!
            IContributionManager.appendToGroup(appAction && separator && menuManager && actionContributionItem);  // REPEATED!
            appSelectionChangedListener.setActionDefinitionId(String);  // REPEATED!
            appTreeViewer.addDoubleClickListener(appDoubleClickListener);
            appSelectionChangedListener.setToolTipText(string4);  // REPEATED!
            ActionContributionItem actionContributionItem = new ActionContributionItem(appAction);    // REPEATED!
            menuManager.add(separator && appContributionItem && menuManager && groupMarker && appAction);   // REPEATED!
            appSelectionChangedListener.setImageDescriptor();     // REPEATED!
            AppAction appAction = new AppAction();
            appTreeViewer.removeTreeListener(appTreeViewerListener);
            IPreferenceStore.putValue(string6);
            appTreeViewer.setSorter(appViewerSorter);
            IMenuManager.removeMenuListener(appMenuListener);     // REPEATED!
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(appTrayDialog && appAction);
            Assert.isNotNull(appLabelProvider && appTreeContentProvider && appTreeViewer && point && point1 && button); // REPEATED!
            Object object = appTreeViewer.getInput(); // REPEATED!
            boolean app_boolean5 = IPreferenceStore.isDefault(String);  // REPEATED!
            Tree tree = appTreeViewer.getTree();      // REPEATED!
            IPreferenceStore.setDefault(String,String):()||(String,boolean):()||(String,float):()||(String,double):()||(String,long):()||(String,int):();
            Control control = appTreeViewer.getControl();   // REPEATED!
            boolean app_boolean4 = IPreferenceStore.getBoolean(String); // REPEATED!
            Object[] objectArray2 = iselection.toArray();   // REPEATED!
            appTreeViewer.setContentProvider(appTreeContentProvider && arrayContentProvider);
            String string5 = appLabelProvider.getText(Object);    // REPEATED!
        }
}
```

Description of False Negatives:

The following instructions are missing (3 false negatives) since they reside in org.eclipse.ui, not in JFace:
IToolBarManager manager = getViewSite().getActionBars().getToolBarManager();