Eclipse – Tree Viewer – One Trace
Use Full Trace (No Slicing)

Supporting Applications : [LDAP Browser]

```
import org.eclipse.ui.ISharedImages;
import org.eclipse.swt.graphics.FontData;
import org.eclipse.jface.text.templates.TemplateContextType;
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.action.IMenuListener;
import org.eclipse.ui.IActionBars;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.swt.graphics.Font;
import org.eclipse.ui.contexts.IContextActivation;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.core.commands.IHandler && actionHandler;
import org.eclipse.ui.IWorkbench;
import org.eclipse.jface.resource.FontRegistry;
import org.eclipse.core.expressions.Expression;
import org.eclipse.ui.editors.text.templates.ContributionTemplateStore;
import org.eclipse.core.runtime.Preferences;
import org.eclipse.ui.help.IWorkbenchHelpSystem;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.IWorkbenchPartSite;
import org.eclipse.jface.preference.PreferenceConverter;
import org.eclipse.swt.graphics.RGB;
import org.eclipse.core.runtime.Plugin;
import org.eclipse.jface.preference.IPreferenceStore;
import java.util.Iterator;
import org.eclipse.ui.IDecoratorManager;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.IToolBarManager;
import org.eclipse.core.runtime.IPath;
import org.eclipse.jface.commands.ActionHandler;
import org.eclipse.core.runtime.Preferences$IPropertyChangeListener;
import org.eclipse.ui.IPartListener2;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.ui.editors.text.templates.ContributionContextTypeRegistry;
import org.eclipse.ui.ISelectionListener;
import org.eclipse.jface.viewers.DecoratingLabelProvider;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.core.runtime.Path;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.core.commands.Command;
import org.eclipse.jface.viewers.IColorProvider;
import org.eclipse.ui.IWorkbenchPartReference;
import org.eclipse.swt.graphics.Color;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.ISelectionService;
import org.eclipse.swt.widgets.Tree;
import org.eclipse.ui.PlatformUI;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.ILabelDecorator;
import org.eclipse.ui.IEditorInput;
import org.eclipse.jface.text.templates.GlobalTemplateVariables$Cursor;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.layout.GridData;
import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.jface.resource.ColorRegistry;
import org.eclipse.jface.viewers.ViewerSorter;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.ui.IViewSite;
import org.eclipse.jface.viewers.IFontProvider;
import org.eclipse.jface.viewers.ITreeContentProvider;
import org.eclipse.jface.viewers.ITreeViewerListener;
import org.eclipse.jface.action.Action;

public class AppWorkbenchPart
implements IWorkbenchPart  {

      public Object getAdapter(Class) {
      }

      public void setFocus() {
            Control control = treeViewer.getControl();
            boolean app_boolean2 = control.setFocus();
      }
```

```
	public void dispose() {
		labelProvider.dispose();
		menuManager.dispose();
		appTreeContentProvider.dispose();
		IWorkbenchWindow iworkbenchWindow = IWorkbenchPart.getActiveWorkbenchWindow();
		ISelectionService iselectionService = iworkbenchWindow.getSelectionService();
		iselectionService.removeSelectionListener(appDoubleClickListener);
		IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();
		ipreferenceStore.removePropertyChangeListener(appreferences$IPropertyChangeListener && appPropertyChangeListener);
		control.dispose();
		Preferences preferences = Plugin.getPluginPreferences();
		preferences.removePropertyChangeListener(appreferences$IPropertyChangeListener && appPropertyChangeListener);
		IWorkbenchPartSite iworkbenchPartSite = iworkbenchPart.getSite(); // REPEATED!
		treeViewer.removeSelectionChangedListener(appAction && appSelectionChangedListener);	// REPEATED!
		treeViewer.removeTreeListener(appDoubleClickListener);
		treeViewer.removeDoubleClickListener(appDoubleClickListener);
		appWorkbenchWindowActionDelegate.dispose();	// REPEATED!
		/*			Cyclic Statements			*/
		IWorkbenchPage iworkbenchPage = iworkbenchPartSite.getPage();
		iworkbenchPage.removePartListener(appDoubleClickListener);
		iworkbenchPartSite.setSelectionProvider(treeViewer);
	}

	public void createPartControl(Composite) {
		AppDoubleClickListener appDoubleClickListener = new AppDoubleClickListener();
		MenuManager menuManager = new MenuManager(String)||(String,String)||();
		ActionHandler actionHandler = new ActionHandler(appAction && appSelectionChangedListener);	// REPEATED!
		GridData gridData = new GridData(int)||(int,int,boolean,boolean)||(int,int,boolean,boolean,int,int)||(int,int)||();	// REPEATED!
		Appreferences$IPropertyChangeListener appreferences$IPropertyChangeListener = new Appreferences$IPropertyChangeListener();
		String string1 = appSelectionChangedListener.getActionDefinitionId();	// REPEATED!
		String string = ActionFactory.getId();	// REPEATED!
		AppPropertyChangeListener appPropertyChangeListener = new AppPropertyChangeListener();
		AppColorProvider appColorProvider = new AppColorProvider(appreferences$IPropertyChangeListener && appPropertyChangeListener);
		AppFontProvider appFontProvider = new AppFontProvider(appreferences$IPropertyChangeListener && appPropertyChangeListener);
		LabelProvider labelProvider = new LabelProvider(appreferences$IPropertyChangeListener && appPropertyChangeListener);
		AppViewerSorter appViewerSorter = new AppViewerSorter(appreferences$IPropertyChangeListener && appPropertyChangeListener);
		AppTreeContentProvider appTreeContentProvider = new AppTreeContentProvider(appreferences$IPropertyChangeListener &&
                                                                                   appPropertyChangeListener && appViewerSorter);
		Separator separator = new Separator(String)||();	// REPEATED!
		IWorkbench iworkbench = AbstractUIPlugin.getWorkbench();
		IWorkbenchHelpSystem iworkbenchHelpSystem = iworkbench.getHelpSystem();
		IDecoratorManager idecoratorManager = iworkbench.getDecoratorManager();
		idecoratorManager.getLabelDecorator(String):(ILabelDecorator)||():(ILabelDecorator);
		DecoratingLabelProvider decoratingLabelProvider = new DecoratingLabelProvider(appColorProvider && appFontProvider &&
                                                                                      labelProvider && idecoratorManager);
		Object object = iworkbench.getAdapter(Class);	// REPEATED!
		Command command = object.getCommand(string1);	// REPEATED!
		boolean app_boolean1 = command.setHandler(IHandler && actionHandler);	// REPEATED!
		Composite composite = new Composite(Composite,int)||();
		Tree tree = new Tree(composite);
		TreeViewer treeViewer = new TreeViewer(tree);
		treeViewer.setUseHashlookup(boolean);
		treeViewer.setSorter(appViewerSorter);
		treeViewer.setContentProvider(appTreeContentProvider);
		treeViewer.setLabelProvider(decoratingLabelProvider);
		tree.setLayoutData(gridData); // REPEATED!
		iworkbenchHelpSystem.setHelp(composite);
		AppMenuListener appMenuListener = new AppMenuListener();
		GridLayout gridLayout = new GridLayout(int,boolean)||();
		composite.setLayout(gridLayout);
		/*			Cyclic Statements			*/
		itoolBarManager.update(boolean);	// REPEATED!
		iactionBars.setGlobalActionHandler(string && appAction && appSelectionChangedListener);	// REPEATED!
		control.setMenu(menu);
		IMenuManager imenuManager = iactionBars.getMenuManager();
		treeViewer.setInput(iworkbenchPartSite && iviewSite);
		IToolBarManager itoolBarManager = iactionBars.getToolBarManager();
		IViewSite iviewSite = IViewPart.getViewSite();	// REPEATED!
		menuManager.addMenuListener(appMenuListener);
		iworkbenchPartSite.setSelectionProvider(treeViewer);
		Menu menu = menuManager.createContextMenu(tree && control);
		iactionBars.updateActionBars();
		menuManager.setRemoveAllWhenShown(boolean);
		itoolBarManager.add(separator && appAction && appSelectionChangedListener);	// REPEATED!
		IActionBars iactionBars = iworkbenchPartSite.getActionBars();	// REPEATED!
		IWorkbenchPartSite iworkbenchPartSite = iworkbenchPart.getSite(); // REPEATED!
		imenuManager.update(String):()||():();
		Control control = treeViewer.getControl();	// REPEATED!
	}
}

public class AppViewerSorter
extends ViewerSorter {

	public void sort(treeViewer) {
		IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();	// REPEATED!
		int app_int = ipreferenceStore.getInt(String);	// REPEATED!
	}
}
```

```
public class AppTreeContentProvider
implements ITreeContentProvider  {

	public void dispose() {
	}

	public void inputChanged(treeViewer && iworkbenchPartSite && iviewSite) {
	}

	public Object[] getElements(iworkbenchPartSite && iviewSite) {
		Object[] objectArray1 = appTreeContentProvider.getChildren(iworkbenchPartSite && iviewSite);
	}

}

public class AppDoubleClickListener
implements IDoubleClickListener, IPartListener2, ISelectionListener, ITreeViewerListener  {

	public  AppDoubleClickListener() {
		treeViewer.addTreeListener(appDoubleClickListener);
		treeViewer.addDoubleClickListener(appDoubleClickListener);
		PlatformUI.getWorkbench();
		IWorkbenchWindow iworkbenchWindow = IWorkbenchPart.getActiveWorkbenchWindow();
		ISelectionService iselectionService = iworkbenchWindow.getSelectionService();
		iselectionService.addSelectionListener(appDoubleClickListener);
		IWorkbenchPartSite iworkbenchPartSite = iworkbenchPart.getSite();
		IWorkbenchPage iworkbenchPage = iworkbenchPartSite.getPage();
		iworkbenchPage.addPartListener(appDoubleClickListener);
	}

	public void partActivated(IWorkbenchPartReference) {
		PlatformUI.getWorkbench();    // REPEATED!
		Object object = IWorkbenchPart.getAdapter(Class);     // REPEATED!
		object.activateContext(String,Expression):(IContextActivation)||(String,Expression,boolean):(IContextActivation;
		IWorkbenchPart iworkbenchPart = IWorkbenchPartReference.getPart(boolean);     // REPEATED!
		String string1 = appSelectionChangedListener.getActionDefinitionId();   // REPEATED!
		Command command = object.getCommand(string1);   // REPEATED!
		ActionHandler actionHandler = new ActionHandler(appAction && appSelectionChangedListener);      // REPEATED!
		boolean app_boolean1 = command.setHandler(IHandler && actionHandler);   // REPEATED!
		String string = ActionFactory.getId();    // REPEATED!
		/*                    Cyclic Statements                  */
		iactionBars.updateActionBars();
		iactionBars.setGlobalActionHandler(string && appAction && appSelectionChangedListener);   // REPEATED!
	}

	public void partClosed(IWorkbenchPartReference) {
	}

	public void selectionChanged(iworkbenchPart) {
		Iterator iterator = IStructuredSelection.iterator();  // REPEATED!
		AppEditorInput appEditorInput = new AppEditorInput();
	}

	public void partDeactivated(IWorkbenchPartReference) {
		PlatformUI.getWorkbench();    // REPEATED!
		Object object = IWorkbenchPart.getAdapter(Class);     // REPEATED!
		object.deactivateContext();
		IWorkbenchPart iworkbenchPart = IWorkbenchPartReference.getPart(boolean);     // REPEATED!
		String string1 = appSelectionChangedListener.getActionDefinitionId();   // REPEATED!
		Command command = object.getCommand(string1);   // REPEATED!
		boolean app_boolean1 = command.setHandler(IHandler && actionHandler);   // REPEATED!
		String string = ActionFactory.getId();    // REPEATED!
		/*                    Cyclic Statements                  */
		iactionBars.updateActionBars();
		iactionBars.setGlobalActionHandler(string && appAction && appSelectionChangedListener);   // REPEATED!
	}

	public void partHidden(IWorkbenchPartReference) {
	}

	public void partOpened(IWorkbenchPartReference) {
	}

	public void partVisible(IWorkbenchPartReference) {
	}
}

public class AppAction
extends Action {

	public  AppAction() {
		IWorkbench iworkbench = PlatformUI.getWorkbench();
		ISharedImages isharedImages = iworkbench.getSharedImages();
		ImageDescriptor imageDescriptor = isharedImages.getImageDescriptor(String);
		String string4 = appAction.getText();
		appSelectionChangedListener.setText(String);
		appSelectionChangedListener.setToolTipText(String);
		appSelectionChangedListener.setEnabled(boolean);
		Path path = new Path(String,String[],int)||()||(String)||(String,String);
```

```
            treeViewer.addSelectionChangedListener(appAction && appSelectionChangedListener);
            /*                      Cyclic Statements                   */
            IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();
            ImageDescriptor.createFromURL();
            boolean app_boolean = ipreferenceStore.getBoolean(String);
            Plugin.find(path);
      }
}

public class AppWorkbenchWindowActionDelegate
implements IWorkbenchWindowActionDelegate  {
}

public class AppSelectionChangedListener
implements ISelectionChangedListener  {

      public  AppSelectionChangedListener() {
            IWorkbench iworkbench = PlatformUI.getWorkbench();
            ISharedImages isharedImages = iworkbench.getSharedImages();
            ImageDescriptor imageDescriptor = isharedImages.getImageDescriptor(String);
            Path path = new Path(String,String[],int)||()||(String)||(String,String);
            treeViewer.addSelectionChangedListener(appAction && appSelectionChangedListener);
            appSelectionChangedListener.setText(String);
            appSelectionChangedListener.setToolTipText(String);
            appSelectionChangedListener.setEnabled(boolean);
            /*                      Cyclic Statements                   */
            ImageDescriptor.createFromURL();
            boolean app_boolean = ipreferenceStore.getBoolean(String);
            IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();
            Plugin.find(path);
      }
}

public class AppEditorInput
implements IEditorInput  {
}

public class Appreferences$IPropertyChangeListener
implements Preferences$IPropertyChangeListener  {

      public  Appreferences$IPropertyChangeListener() {
            Preferences preferences = Plugin.getPluginPreferences();
            preferences.addPropertyChangeListener(appreferences$IPropertyChangeListener && appPropertyChangeListener);
            IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();
            ipreferenceStore.addPropertyChangeListener(appreferences$IPropertyChangeListener && appPropertyChangeListener);
      }
}

public class AppPropertyChangeListener
implements IPropertyChangeListener  {

      public  AppPropertyChangeListener() {
            Preferences preferences = Plugin.getPluginPreferences();
            preferences.addPropertyChangeListener(appreferences$IPropertyChangeListener && appPropertyChangeListener);
            IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();
            ipreferenceStore.addPropertyChangeListener(appreferences$IPropertyChangeListener && appPropertyChangeListener);
      }
}

public class AppAdaptable
implements IAdaptable  {
}

public class AppMenuListener
implements IMenuListener  {

      public  AppMenuListener() {
            IWorkbenchPartSite iworkbenchPartSite = iworkbenchPart.getSite();
            IWorkbenchWindow iworkbenchWindow1 = iworkbenchPartSite.getWorkbenchWindow();
            /*                      Cyclic Statements                   */
            AppWorkbenchWindowActionDelegate appWorkbenchWindowActionDelegate = new AppWorkbenchWindowActionDelegate(treeViewer &&
                                                                                appAction && appSelectionChangedListener && iworkbenchWindow1);
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(treeViewer &&
                                                                                                      appWorkbenchWindowActionDelegate);
            AppAction appAction = new AppAction(appreferences$IPropertyChangeListener && appPropertyChangeListener && treeViewer &&
                                                                                                      appWorkbenchWindowActionDelegate);
      }
}

public class AppColorProvider
implements IColorProvider  {
}

public class AppFontProvider
implements IFontProvider  {
}

public class SomeClass {

      public void someMethod() {
```

```
        GlobalTemplateVariables$Cursor globalTemplateVariables$Cursor = new GlobalTemplateVariables$Cursor(); // REPEATED!
        ContributionContextTypeRegistry contributionContextTypeRegistry = new ContributionContextTypeRegistry();    // REPEATED!
        AppAdaptable appAdaptable = new AppAdaptable();
        RGB rGB = new RGB(float,float,float)||(int,int,int);
        IPath ipath = Plugin.getStateLocation();  // REPEATED!
        ipath.append(String):(IPath)||(IPath):(IPath);  // REPEATED!
        Display.getDefault();
        /*                      Cyclic Statements                   */
        TemplateContextType templateContextType = contributionContextTypeRegistry.getContextType(String);     // REPEATED!
        FontRegistry fontRegistry = new FontRegistry(display);
        FontData[] fontDataArray = font.getFontData();
        RGB rGB1 = color.getRGB();
        IWorkbench iworkbench = AbstractUIPlugin.getWorkbench();    // REPEATED!
        Color color = display.getSystemColor(int);
        ColorRegistry colorRegistry = new ColorRegistry(display);
        PreferenceConverter.setDefault(rGB && ipreferenceStore && fontData && rGB1);
        contributionContextTypeRegistry.addContextType(TemplateContextType);    // REPEATED!
        IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();    // REPEATED!
        String string2 = FontData.getName();
        FontData fontData = new FontData(string2);
        ContributionTemplateStore contributionTemplateStore = new ContributionTemplateStore(contributionContextTypeRegistry &&
                                                                                             ipreferenceStore);     // REPEATED!
        ipreferenceStore.setDefault(String,String):()||(String,boolean):()||(String,float):()||(String,double):()||
                                                                                            (String,long):()||(String,int):();
        Display display = iworkbench.getDisplay();      // REPEATED!
        Font font = display.getSystemFont();
        templateContextType.addResolver(globalTemplateVariables$Cursor);  // REPEATED!
        int app_int1 = FontData.getHeight();
        contributionTemplateStore.load();   // REPEATED!
    }
}
```

Description of False Negatives:

- The class AppWorkbenchPart should extend ViewPart. So, we have one false negtive for the instruction **extends** ViewPart.
- The class AppTreeContentProvider should also implement the method getChildren(). So, we have another false negative for this instruction.