Eclipse – Navigation – One Trace
Use Concept Trace Slicing

Supporting Applications : [SVN]

```
import org.eclipse.ui.progress.DeferredTreeContentManager;
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.action.IMenuListener;
import org.eclipse.ui.IActionBars;
import org.eclipse.swt.graphics.Font;
import org.eclipse.ui.IViewActionDelegate;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.jface.action.IStatusLineManager;
import org.eclipse.core.resources.IResource;
import org.eclipse.jface.resource.FontRegistry;
import org.eclipse.ui.model.WorkbenchContentProvider;
import org.eclipse.ui.model.WorkbenchLabelProvider;
import org.eclipse.ui.actions.SelectionListenerAction;
import org.eclipse.team.core.RepositoryProvider;
import org.eclipse.ui.help.IWorkbenchHelpSystem;
import org.eclipse.ui.IViewPart;
import org.eclipse.team.ui.TeamOperation;
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 org.eclipse.ui.themes.ITheme;
import java.util.Iterator;
import org.eclipse.core.runtime.IAdapterFactory;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.ui.themes.IThemeManager;
import org.eclipse.jface.viewers.SelectionChangedEvent;
import java.io.File;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.IToolBarManager;
import org.eclipse.core.runtime.IPath;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.core.runtime.jobs.ISchedulingRule;
import org.eclipse.jface.viewers.ILightweightLabelDecorator;
import java.text.Collator;
import org.eclipse.ui.ISelectionListener;
import org.eclipse.core.runtime.Platform;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.window.SameShellProvider;
import org.eclipse.core.resources.IProject;
import org.eclipse.ui.dialogs.PropertyDialogAction;
import org.eclipse.core.runtime.SubProgressMonitor;
import org.eclipse.ui.model.IWorkbenchAdapter;
import org.eclipse.swt.events.KeyAdapter;
import org.eclipse.swt.graphics.Color;
import org.eclipse.ui.ISelectionService;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.part.DrillDownAdapter;
import org.eclipse.ui.progress.IDeferredWorkbenchAdapter;
import org.eclipse.swt.widgets.Tree;
import org.eclipse.ui.PlatformUI;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.team.core.variants.IResourceVariant;
import org.eclipse.team.core.variants.CachedResourceVariant;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.dnd.DragSourceListener;
import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.eclipse.jface.resource.ColorRegistry;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.jface.viewers.ViewerSorter;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.ui.IViewSite;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.ui.console.IOConsole;
import org.eclipse.jface.action.Action;

public class AppAdaptable
implements IAdaptable, IWorkbenchAdapter  {

      public  AppAdaptable() {
            AppAdaptable appAdaptable = new AppAdaptable();
            AppCachedResourceVariant appCachedResourceVariant = new AppCachedResourceVariant(appAdaptable);
            AppResourceVariant appResourceVariant = new AppResourceVariant(appAdaptable);
      }

      public Object getAdapter(Class) {
            AdapterManager adapterManager = Platform.getAdapterManager();     // REPEATED!
```

```
            adapterManager.getAdapter(appAdaptable && object1);   // REPEATED!
      }

      public Object[] getChildren(appAdaptable && object) {
            AppAdaptable appAdaptable = new AppAdaptable();
            IPath ipath = Plugin.getStateLocation();  // REPEATED!
            ipath.append(String):(IPath)||(IPath):(IPath);  // REPEATED!
      }
}

public class AppLightweightLabelDecorator
implements ILightweightLabelDecorator  {

      public void decorate(iproject) {
            /*                    Cyclic Statements                  */
            RepositoryProvider.getProvider(iproject); // REPEATED!
            IProject iproject = IResource.getProject();     // REPEATED!
            int app_int = iproject.getType();   // REPEATED!
      }
}

public class AppWorkbenchContentProvider
extends WorkbenchContentProvider {

      public void inputChanged(appAdaptable && treeViewer && object && object1) {
            DeferredTreeContentManager deferredTreeContentManager = new DeferredTreeContentManager(appWorkbenchContentProvider
                                                                                                    && treeViewer);   // REPEATED!
      }

      public boolean hasChildren(appAdaptable && object1) {
            boolean app_boolean9 = deferredTreeContentManager.isDeferredAdapter(Object);
      }

      public Object[] getChildren(appAdaptable && object && object1) {
            Object[] objectArray1 = deferredTreeContentManager.getChildren(appAdaptable && object && object1);    // REPEATED!
      }
}

public class AppSelectionListener
implements ISelectionListener, IWorkbenchPart  {

      public void selectionChanged(iselection2) {
            IViewSite iviewSite = IViewPart.getViewSite();  // REPEATED!
            IActionBars iactionBars = iviewSite.getActionBars();  // REPEATED!
            IStatusLineManager istatusLineManager = iactionBars.getStatusLineManager();   // REPEATED!
            istatusLineManager.setMessage(string);    // REPEATED!
            /*                    Cyclic Statements                  */
            Object object2 = object1.getAdapter(Class);
            int app_int1 = iselection1.size();
            Object object1 = iselection.getFirstElement();
            boolean app_boolean2 = iselection.isEmpty();    // REPEATED!
      }

      public void setFocus() {
            Control control = treeViewer.getControl();
            boolean app_boolean5 = control.setFocus();
      }


public void createPartControl(Composite) {
            AppragSourceListener appragSourceListener = new AppragSourceListener();
            MenuManager menuManager = new MenuManager(String)||(String,String)||();
            AppKeyAdapter appKeyAdapter = new AppKeyAdapter();
            AppAdaptable appAdaptable = new AppAdaptable();
            PlatformUI.getWorkbench();    // REPEATED!
            AppWorkbenchContentProvider appWorkbenchContentProvider = new AppWorkbenchContentProvider();
            ViewerSorter viewerSorter = new ViewerSorter(Collator)||();
            AppDoubleClickListener appDoubleClickListener = new AppDoubleClickListener();
            WorkbenchLabelProvider workbenchLabelProvider = new WorkbenchLabelProvider();
            TreeViewer treeViewer = new TreeViewer(Composite,int)||(Tree)||(Composite);
            treeViewer.setContentProvider(appWorkbenchContentProvider);
            treeViewer.setLabelProvider(workbenchLabelProvider);
            treeViewer.setInput(appAdaptable);
            treeViewer.setSorter(viewerSorter);
            DrillDownAdapter drillDownAdapter = new DrillDownAdapter(treeViewer);
            String string2 = ActionFactory.getId();   // REPEATED!
            Separator separator = new Separator(String)||();
            AppMenuListener appMenuListener = new AppMenuListener();
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener();
            /*                    Cyclic Statements                  */
            ISelection iselection1 = treeViewer.getSelection();   // REPEATED!
            itoolBarManager.update(boolean);
            iactionBars.setGlobalActionHandler(string2 && appSelectionListenerAction && appAction && propertyDialogAction);   // REPEATED!
            iworkbenchHelpSystem.setHelp(appSelectionListenerAction && control && appAction && tree); // REPEATED!
            control.setMenu(menu);
            int app_int1 = iselection1.size();
            treeViewer.addDragSupport(appragSourceListener);
            AppSelectionListenerAction appSelectionListenerAction = new AppSelectionListenerAction(shell);
            propertyDialogAction.setEnabled(boolean); // REPEATED!
            appAction.setDisabledImageDescriptor();   // REPEATED!
```

```
            IToolBarManager itoolBarManager = iactionBars.getToolBarManager();
            IWorkbenchWindow iworkbenchWindow = iworkbenchPartSite.getWorkbenchWindow();
            Shell shell = control.getShell();   // REPEATED!
            IViewSite iviewSite = IViewPart.getViewSite();  // REPEATED!
            menuManager.addMenuListener(appMenuListener);
            iworkbenchPartSite.setSelectionProvider(treeViewer);
            treeViewer.addSelectionChangedListener(appSelectionChangedListener && appSelectionListenerAction);    // REPEATED!
            appAction.setHoverImageDescriptor();      // REPEATED!
            Menu menu = menuManager.createContextMenu(control && tree);
            appSelectionListenerAction.selectionChanged(SelectionChangedEvent);
            iactionBars.updateActionBars();
            treeViewer.addDoubleClickListener(appDoubleClickListener);
            menuManager.setRemoveAllWhenShown(boolean);
            appAction.setToolTipText(String);   // REPEATED!
            IActionBars iactionBars = iworkbenchPartSite.getActionBars();     // REPEATED!
            itoolBarManager.add(separator && appAction);    // REPEATED!
            IWorkbenchPartSite iworkbenchPartSite = IWorkbenchPart.getSite(); // REPEATED!
            AppAction appAction = new AppAction(shell);
            ISelectionService iselectionService = iworkbenchWindow.getSelectionService();
            control.addKeyListener(appKeyAdapter);
            drillDownAdapter.addNavigationActions(itoolBarManager);
            iselectionService.addPostSelectionListener(ISelectionListener):()||(String,ISelectionListener):();
            iworkbenchPartSite.registerContextMenu(menuManager && treeViewer);
            Tree tree = treeViewer.getTree();   // REPEATED!
            Control control = treeViewer.getControl();      // REPEATED!
            PropertyDialogAction propertyDialogAction = new PropertyDialogAction(treeViewer && sameShellProvider);
            SameShellProvider sameShellProvider = new SameShellProvider(shell);
      }
}

public class AppViewActionDelegate
implements IViewActionDelegate  {

      public void selectionChanged(iselection2) {
      }

      public void init(IViewPart) {
            IWorkbenchPartSite iworkbenchPartSite = IWorkbenchPart.getSite();
            Shell shell1 = iworkbenchPartSite.getShell();
      }
}

public class AppDeferredWorkbenchAdapter
implements IDeferredWorkbenchAdapter  {

      public ImageDescriptor getImageDescriptor(appAdaptable) {
      }

      public String getLabel(appAdaptable && object1) {
      }

      public ISchedulingRule getRule(appAdaptable && object1) {
            AppSchedulingRule appSchedulingRule = new AppSchedulingRule(appAdaptable && object1);
      }
}

public class AppSelectionListenerAction
extends SelectionListenerAction {

      public boolean updateSelection(iselection) {
            AppSubProgressMonitor appSubProgressMonitor = new AppSubProgressMonitor();
            boolean app_boolean2 = iselection.isEmpty();
            Iterator iterator = iselection.iterator();
      }
}

public class AppSelectionChangedListener
implements ISelectionChangedListener  {

      public void selectionChanged(SelectionChangedEvent) {
            AppSubProgressMonitor appSubProgressMonitor = new AppSubProgressMonitor();
            appSubProgressMonitor.beginTask(String,int);
            ISelection iselection = SelectionChangedEvent.getSelection();     // REPEATED!
            int app_int1 = iselection.size();
            Object object1 = iselection.getFirstElement();
            propertyDialogAction.setEnabled(boolean); // REPEATED!
      }
}

public class AppSubProgressMonitor
extends SubProgressMonitor {

      public  AppSubProgressMonitor() {
            ISelection iselection = SelectionChangedEvent.getSelection();
            boolean app_boolean2 = iselection.isEmpty();
      }

      public void setTaskName(String) {
      }
}
```

```
public class AppProgressMonitor
implements IProgressMonitor  {

      public void beginTask(String,int) {
            propertyDialogAction.setEnabled(boolean); // REPEATED!
      }
}

public class AppSelectionChangedEvent
extends SelectionChangedEvent {

      public ISelection getSelection() {
            AppTeamOperation appTeamOperation = new AppTeamOperation(appAdaptable && appCachedResourceVariant && appResourceVariant);
      }
}

public class AppAction
extends Action {
}

public class AppPropertyChangeListener
implements IPropertyChangeListener  {

      public  AppPropertyChangeListener() {
            ImageDescriptor.createFromURL();
            IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();
            /*                    Cyclic Statements                    */
            int app_int2 = ipreferenceStore.getInt(String);
            ipreferenceStore.addPropertyChangeListener(ioConsole && appPropertyChangeListener);
            boolean app_boolean = ipreferenceStore.getBoolean(String);
      }
}

public class AppAdapterFactory
implements IAdapterFactory  {

      public  AppAdapterFactory() {
            AppDeferredWorkbenchAdapter appDeferredWorkbenchAdapter = new AppDeferredWorkbenchAdapter();
            AppAdaptable appAdaptable = new AppAdaptable();
      }
}

public class AppResourceVariant
implements IResourceVariant  {
}

public class AppCachedResourceVariant
extends CachedResourceVariant {
}

public class AppTeamOperation
extends TeamOperation {
}

public class AppMenuListener
implements IMenuListener  {
}

public class AppKeyAdapter
extends KeyAdapter {
}

public class AppDoubleClickListener
implements IDoubleClickListener  {
}

public class AppragSourceListener
implements DragSourceListener  {
}

public class AppSchedulingRule
implements ISchedulingRule  {
}

public class SomeClass {

      public void someMethod() {
            IOConsole ioConsole = new IOConsole(String,String,ImageDescriptor,String,boolean)||
                                          (String,ImageDescriptor)||(String,String,ImageDescriptor,boolean);
            appSelectionListenerAction.selectionChanged(SelectionChangedEvent);     // REPEATED!
            boolean app_boolean6 = appSelectionListenerAction.updateSelection(iselection2);
            int app_int1 = iselection1.size();
            String string3 = appAdaptable.getLabel(appAdaptable && object1);  // REPEATED!
            appSubProgressMonitor.beginTask(String,int);    // REPEATED!
            boolean app_boolean8 = appWorkbenchContentProvider.hasChildren(Object);
            Platform.getAdapterManager(); // REPEATED!
            PlatformUI.getWorkbench();    // REPEATED!
            FontRegistry fontRegistry = itheme.getFontRegistry(); // REPEATED!
            itheme.addPropertyChangeListener(IPropertyChangeListener);
```

```
        Font font = fontRegistry.get(String);     // REPEATED!
        RGB rGB = new RGB(float,float,float)||(int,int,int);  // REPEATED!
        AppAdapterFactory appAdapterFactory = new AppAdapterFactory();
        boolean app_boolean4 = appSchedulingRule.isConflicting(ischedulingRule && appSchedulingRule);
        /*                    Cyclic Statements                  */
        String string = ipreferenceStore.getString(String);   // REPEATED!
        AppPropertyChangeListener appPropertyChangeListener = new AppPropertyChangeListener(ipreferenceStore);
        PreferenceConverter.setDefault(rGB && ipreferenceStore);    // REPEATED!
        ipreferenceStore.addPropertyChangeListener(ioConsole && appPropertyChangeListener);
        IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();    // REPEATED!
        Platform.getAuthorizationInfo(string);    // REPEATED!
        boolean app_boolean7 = Plugin.isDebugging();
        ipreferenceStore.setDefault(String,String):()||(String,boolean):()||(String,float):()||(String,double):()||
                                                                    (String,long):()||(String,int):();  // REPEATED!
        boolean app_boolean = ipreferenceStore.getBoolean(String);  // REPEATED!
    }
}
```