

JTube: A Short Introduction

Patrick Nepper

December 14, 2005

Abstract

Structure and documentation are two major factors for successful software development. Retaining control over them is difficult, because there are various forms of structural information and documentation. Java source code is still full of neglected sources of such information. Whether it is relations between methods of the same class or dependencies between source code and external documentation, we lack lightweight tools to extract and visualize the respective data. In this paper we present *JTube*, a tool that assists the developer with making use of this hidden information. *JTube* is designed to be easily usable for Java developers: it comes as an Eclipse plugin and makes use of common practices in writing Java source code, namely grouping methods, using submethods and referencing external documentation.

1 JTube

1.1 Overview

JTube is an Eclipse plug-in that supports the developer in using Java comments to structure and document code and visualizes the inner structure of Java classes by combining different sources of structural information (submethod dependencies, Java comments). JTube is designed to be a lightweight component that integrates seamlessly with the Eclipse SDK.

Our goal was to solve the problems described in Sect. ?? in a “hands-on” fashion, making use of current development tools while protecting the developer from having to adopt a new process. The result is JTube, an Eclipse plug-in that was implemented as a prototype to present the desired features in a Java environment:

1. **Grouping Methods.** JTube supports the developer’s desire to group methods using Java comments and makes this grouping browsable. This way, JTube exploits the structural information contained in Java comments and makes it visible.
2. **Recognizing Submethods.** JTube recognizes that developers create private methods that only function as submethods for other methods and makes this relationship visible.
3. **Adding Meta Information.** JTube helps the developer to reference arbitrary information (e.g. other code segments or external documentation) that is related to a certain line of code making use of the extensive capabilities of Hyena, a tool for RDF-based data linkage [?]. This helps to combine fast-accessible and easy-to-maintain inline documentation with in-depth external documentation.

1.2 Features

1.2.1 Grouping Methods

In recent discussions (cf. Sect. ??) about possible extensions to the current set of Javadoc tags, a tag named “@category” has been suggested to be used for “logically grouping classes, methods, fields” [?]. Although our contribution was not inspired by this suggestion, we adopt the naming for uniformity reasons. JTube supports method grouping in a straightforward manner. By using the

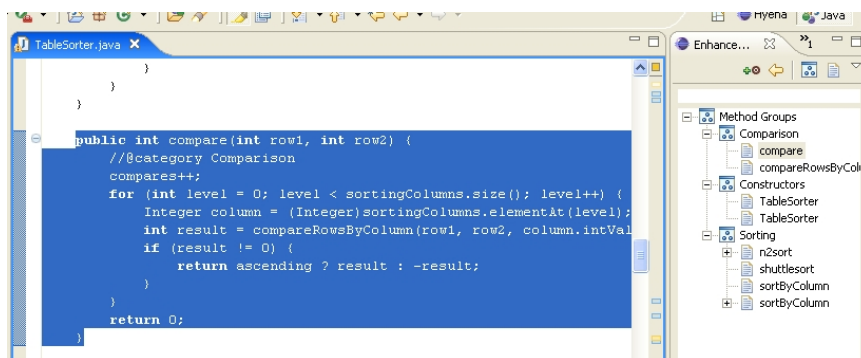


Figure 1: JTube’s Enhanced Outline View visualizes individual method group assignment

Javadoc @category-tag in a method-internal Java comment, a user can mark a method as belonging to one or more groups (separated by commas). Thus, the user can assign methods to groups with respect to their responsibilities (e.g. comparing objects or sorting data).

JTube makes these group assignments visible by providing a specialized outline view (“Enhanced Outline View”): This view is in synch with the current Java editor and presents a group-centric view on the methods contained in the current Java class. The usability of the enhanced outline view is based on the basic principle for visual user interfaces—the *Visual Information Seeking Mantra* as it was described by Shneiderman in [?]:

1. Overview first
2. Zoom and filter
3. Then details-on-demand

JTube’s enhanced outline view presents the user with an *overview* of all groups that are available in the current file. For each method group JTube will present a subtree that contains all the methods that belong to the same group (cf. Fig. 1), which allows the user to *zoom in* on a group of interest. JTube furthermore offers a text field that allows the user to *filter* the displayed list of methods based on (parts of) their name. If the user wants to go into *detail*, he can then easily navigate to methods of the same group by clicking on them.

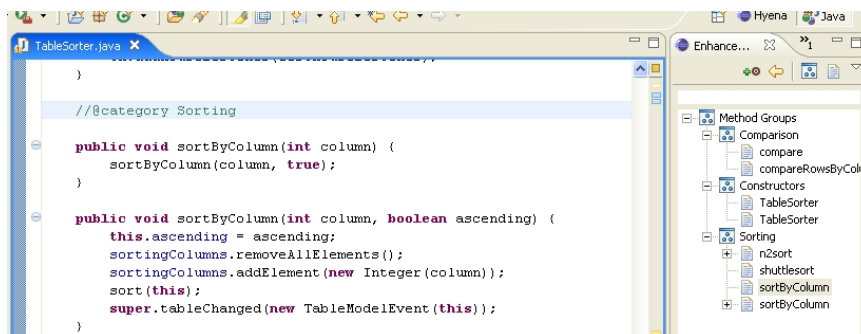


Figure 2: JTube’s Enhanced Outline View supports grouping multiple methods

In order to get even closer to how developers use Java comments to group methods, it is also possible to use the `@category`-tag in a comment that is placed between methods (i.e. in the class scope). This tells JTube to interpret such a group assignment as relevant to all the methods that follow this Java comment. Such a group assignment might span multiple methods until another group assignment, that is also placed between methods, might override the current assignment. JTube’s enhanced outline view offers a unified visualization of method groups—irrespective of whether the groups were assigned by using Java comments that were placed inside of methods or between methods (cf. Fig. 2). Furthermore, the developer can combine these two ways of grouping methods to mix general assignments spanning multiple methods with fine-grained group assignments of individual methods. For example, the developer can assign several methods to the group “Sorting” and one of these methods additionally to the group “Filtering”. This provides the necessary flexibility in making group assignments.

To increase the usability of JTube’s method grouping feature, the user is not forced to use the Javadoc `@category`-tag. Instead, JTube also analyzes

Java comments that start with “//–”, in order to allow for the use of Java comments such as

```
//----- Sorting
```

to structure Java code. This is helpful to visualize the internal structure even of old Java source files (see also the use case in Sect. ??). If JTube encounters a group assignment that defines no group name (e.g. “//@category”), JTube lists the methods belonging to this anonymous group as belonging to the special group “<anonymous group #X>”.

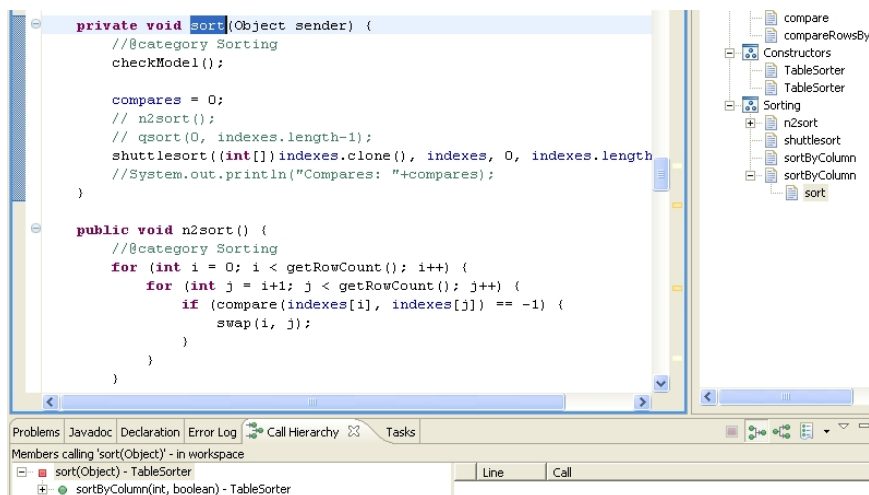


Figure 3: JTube visualizes submethods as leaves in the method-tree

1.2.2 Recognizing Submethods

Sometimes the source code of a single method extends over dozens of lines. In order to keep it simple, developers often refactor the source code and divide such a method into several “submethods”. As Kent Beck puts it, “*Lots of little methods named by what they are intended to do, not how they do it, make understanding the high-level structure of a computation easy.*” [?]. His “*Composite Methods Pattern*” explains the principle of how to extract such submethods from large methods. In Java, submethods are usually declared as private and are only referenced by their parent method. The standard outline view, however, presents us with a flat outline of all methods contained in the class—the dependencies between methods cannot be displayed. JTube provides an enhanced outline view that allows the user to browse methods and their depending submethods. By default, submethods are not displayed in the outline. This keeps the outline view simple and helps understanding the basic structure of the class (cf. Fig. 1). If the user wants to access the submethods for a specific method, he simply expands the respective subtree in the enhanced outline view (cf. Fig. 3). By integrating this feature with the preceding one (method grouping), the developer has a very intuitive outline view at hand that combines two very basic ways of structuring source code. To preserve simplicity, JTube

makes sure that submethods, that are members of the same method group as their parent method, are listed only once in the group listing.

1.2.3 Adding Meta Information

A typical problem of source code documentation is that documentation that concerns cross-cutting topics is scattered over a multitude of different Java source files. Although it is possible to interrelate this information, e.g. by using Javadoc's `@see`-tag, it is hard to get an overview of all related documentation on a specific topic. In [?], Bjune and Hagemeister show that linking documentation to source code and vice versa *“can substantially aid the software development effort by maintaining traceability between work products”*. Their paper describes a showcase tool that can be used to create links between Visual C++ source code and Office documents. Our approach shares the idea of linking-up different sources of information while sticking to the tools the developer is used to, and adds a number of new ideas to facilitate synchronization for changes in the workspace and to improve collaboration with other developers. Through its integration with Hyena, JTube allows to link arbitrary information to Java resources and this information can then be manipulated, browsed and queried by the user in Hyena.

JTube allows the user to add references for lines of code to the Hyena database. The goal of this functionality is to make code positions available for reference within Hyena. This requires basically two things:

1. References for different lines of Java code must be unique.
2. References must be synchronizable with respect to code changes/deletions.

To fulfill these requirements JTube uses Eclipse markers (cf. Sect. ??) and the RDF persistency capabilities of Hyena. Using these technologies, JTube can make sure that the link between the Java source code and Hyena's symbolic reference always stays synchronized. In other words, if the user moves or deletes a file, JTube will automatically change/remove all of the file's references in the Hyena database. If another developer adds or removes a reference from/to Hyena, JTube will remove/add its link to the Hyena reference as soon as the user opens the respective file the next time. To ensure the usability of the above described functionality, JTube offers a very simple interface: JTube's enhanced outline view contributes two buttons to the view's toolbar. The first button allows the user to add a reference to the currently selected line of code to Hyena. JTube adds a decorator to the vertical toolbar to indicate the link to Hyena and to allow the user to navigate to the Hyena reference (cf. Fig. 4). JTube also contributes a command to Hyena (“Show in Java editor”) that can be used to let Eclipse open a Java editor with the currently selected Java source code reference. These two features combined provide bidirectional navigation between the Java editor and Hyena.

After the developer has added a reference to Hyena he can use Hyena to better manage his documentation: By linking up related pieces of documentation in Hyena the developer finally can see all related sources of information in one place. Using Hyena's querying features, it is even possible to create different dynamic views on these networks of documentation. The second button allows the user to force a complete update from the Hyena database. This is very

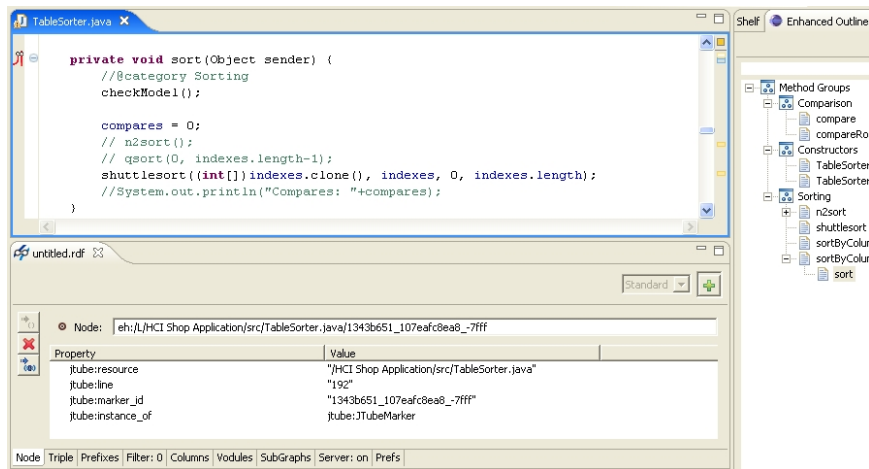


Figure 4: JTube markers link to data in Hyena

useful, when the developer checks out a project from a repository that already uses a Hyena RDF database, because Eclipse currently does not support sharing marker information with other developers. In order to synchronize the information contained in Hyena with the workspace, JTube will add markers to their respective Java resources for all references that were discovered in Hyena and that are currently not managed by JTube.

Furthermore, Hyena can be used to link arbitrary information to referenced code. Since Hyena is extensible and adaptable to a multitude of different fields of application, it will be possible to create tools for managing external sources of information almost without limitations. Everything that is representable as RDF can be imported to Hyena. For example, there is already an extension for an RDF-based Wiki. Linking JTube's method references to Wiki resources might lead to an integration of Javadoc and Wiki-based documentation.

1.3 Architecture

JTube is a simple Eclipse plug-in that contributes a view ("Enhanced Outline") to the Eclipse IDE. It uses some of Eclipse's core features to search files in the workspace and to evaluate Java source code. It provides an interface to Hyena, a tool for managing information with an RDF database.

1.3.1 Components

JTube consists of three main components:

1. **Data.** This component provides the basic functionality to access the relevant data from within the Java workspace and Hyena. It encapsulates all the details of the Eclipse and Hyena integration and offers the necessary interface for querying Java elements, Hyena source code references, etc.
2. **Model.** The data model is an abstraction of the actual Java source code and contains only the data relevant for the current JTube view. This

component is responsible for storing information on method groups and submethods.

3. **View.** This component contains the JTube Enhanced Outline View and all classes necessary to create this view and contribute it to the Eclipse workbench.
4. **Actions.** This component contains all actions that JTube contributes to Eclipse and that can be triggered by the user (e.g. adding references to Hyena).

1.3.2 Eclipse Integration

JTube has been designed to be a lightweight extension to the Eclipse SDK. Its Enhanced Outline View is contributed to the workbench by extending the Eclipse plugin extension point `org.eclipse.ui.views` and can be found in the Eclipse views collection under the category `JTube`. Since Hyena is also an Eclipse plugin, JTube also extends a Hyena extension point to integrate with Hyena (`de.hypergraphs.hyena.vodules`).

The JTube plugin makes use of several Eclipse core features:

- **File Search Engine.** To search the locations of methods that belong to the same group (cf. Sect. 1.2.1).
- **Java Search Engine.** To analyze if a method has the role of being a submethod (cf. Sect. 1.2.2).
- **Java Data Model.** To access Java methods as data objects.
- **Resource Markers.** To keep track of changes in source code positions and to enable synchronization with Hyena (cf. Sect. ??).
- **Resource Change Listener.** To be notified whenever a resource in the workspace is changed or deleted (cf. Sect. 1.4).

1.4 Synchronization

A major issue for the implementation of JTube was the synchronization between the Eclipse workspace and Hyena. Eclipse already offers everything needed to synchronize Java objects that are attached to specific lines of code in Java resources (so called “markers”). That means, whenever a user moves or changes a file or its contents, Eclipse makes sure that markers stay attached to their respective lines of code. If the user deletes a file all of its markers are deleted as well. As mentioned in [?] we only have to take care of “stale markers” that may still exist in Hyena, when they are changed or deleted by Eclipse. To achieve this we follow a two-fold synchronization process:

1. **Changes in the workspace.** In order to track changes in Java source files, we add a `ResourceChangeListener` to the Eclipse workspace that handles file move/rename operations. The listener updates Hyena by either removing marker information (file was removed) or by changing marker information (resource name/location has changed).

2. **Changes in Hyena.** If Hyena has been assigned new information from outside the current Eclipse workspace (e.g. by another software developer) it might reference non-existing markers or markers still existing in the workspace might have been removed from Hyena. To accommodate this issue, we regularly update the markers in the workspace with the information from the Hyena knowledge-base, either when a Java source file is opened or when the user explicitly orders a full update by clicking on the respective button.

This way, we provide the user with full flexibility in managing the links between JTube and Hyena: Both directions (JTube \rightarrow Hyena and Hyena \rightarrow JTube) support adding, changing and removing links. To make the synchronization method reliable, Hyena will always have the higher precedence. The basis for this clear decision is the fact that JTube's information relies on Eclipse markers that are only stored locally in the user's workspace. Hyena's information, however, can be shared with other developers and therefore deserves higher precedence in order to allow developers to collaborate (e.g. using a CVS repository).