

Knowledge Generation in Java API Reference Documentation

Miss. Monali Metkar¹, Prof. S. M. Kamalapur²

¹Department of Computer Engineering, PG Student, KKWIEER, Nashik, University of Pune, India

²Department of Computer Engineering, Associate Professor, KKWIEER, Nashik, University of Pune, India

Abstract

Application Program Interface (API) allows programmers to use predefined functions instead of writing them from scratch. Description of API elements that is Methods, Classes, Constructors etc. is provided through API Reference Documentation. Hence API Reference Documentation acts as a guide to user or developer to use API's. This work focuses on Knowledge generation in the Java API Reference Documentation.

Keywords: API, API Reference Documentation, Knowledge Types.

1. INTRODUCTION

An Application Programming Interface (API) is a set of commands, functions, and protocols. It also specifies the interaction between the software components. In most procedural languages, an API specifies a set of functions or routines that accomplish a specific task or are allowed to interact with a specific software component. For example, consider following Field in Java:

```
static InputStream in
```

In above example, "in" is used for standard input stream.

Whenever user or developer is referring to an API and has planned to use it for specific purpose API Reference documentation works as a guide. API Reference Documentation is an important part of programming with APIs and it complements the API by providing information about the API. So, it plays a crucial role in how developers learn and use an API, and developers will expect the information about API elements they should find therein. By considering the above example, if new developer wishes to use "in" field in Java Program, he can refer to API Reference Documentation of Java and he will find the description of "in" field in Field Summary as:

The "standard" input stream.

In above example Java Documentation is considered and Java APIs are documented through Javadocs which is a set of web pages such that one for each package or type in the API.

To enhance the quality of API reference documentation and the efficiency with which the relevant information it contains can be accessed, it's necessary to first understand its contents by analyzing it. Therefore, to reason about the quality and value of Java API reference documentation, focus should be about what knowledge it contains. Because Knowledge refers to retrieve useful information from data and then use this knowledge for specific purpose. By analyzing the contents of Java API Reference Documentation Knowledge is generated and this knowledge can be categorized further.

Previous work focused separately on Studies of Knowledge Categorization and of API reference Documentation and Knowledge retrieval was done based on Experience, Observations and Analysis.

So proposed system focuses on classification of knowledge types in API reference documentation.

Section II focuses on Literature Review. Section III gives Implementation Details with Block Diagram, Concept with Example and Algorithms are highlighted in. Data Set and Results obtained are discussed in Section IV of Results. The paper ends with concluding remarks.

2. LITERATURE REVIEW

The previous work mainly focused on the Knowledge Categorization and API Reference Documentation Separately.

2.1 API Reference Documentation

Study of documentation needs for a domain-specific API, using surveys and interviews of developers was done by Nykaza et al.[6] This study identified, among other requirements and the importance of an overview section in API documentation.

Jeong et al. [10] conducted a lab study with eight participants to assess the documentation of a specific service-oriented architecture. This study identified 18 guidelines they believe would lead to increased documentation quality for the system under study, including "explaining starting points" for using the API.

Robillard and DeLine [9] identified the obstacles faced by developers when trying to learn new APIs through surveys and interviews with Microsoft developers. The study showed that many obstacles were related to aspects of the documentation, but did not include the systematic analysis of API documentation content.

Similarly, Shi et al. [8] studied API documentation evolution. The authors apply data mining techniques over the source repository of five open-source APIs. Their study provides various quantitative measures of which parts of the API documentation are most frequently revised, and how often API documentation is changed consistently with the corresponding elements.

2.2 Knowledge Categorization based on Manual Methods

Researchers have applied Knowledge from one field to other field, they also studied which are the different questions raised in Software Project Development.

Mylopoulos et al.[5] discussed how knowledge representation techniques from the field of Artificial Intelligence can be applied to software engineering. The authors presented a categorization of different knowledge types, presumably derived from their experience.

Requirement and Design are the important stages in Software Project Development. Herbsleb and Kuwana[4] classified questions asked in design meetings to study the kinds of knowledge that may benefit from explicit capture at the requirements and design stages based on their general experience.

Hou et al.[2] studied 300 questions related to two specific Swing widgets (JButton and JTree) posted on the Swing forum. They then mapped the questions to the different design features of the widgets. Their classification focuses more on the target of the question and less on discovering the different types of knowledge provided to and sought by API users.

More recently, Ko et al.[1] observed 17 developers at Microsoft for a 90 minutes session each, studying their information needs as they perform their software engineering tasks. From the observation data the authors collected 334 specific information needs, which they abstracted into 21 general information needs.

Kirk et al.[3] investigated the knowledge problems faced by them and their students when trying to develop applications by extending the JHotDraw framework.

Similarly to Ko et al.'s study, Sillito et al.[7] produced a catalog of 44 types of questions developers ask during software evolution tasks. The questions in the catalog do not focus exclusively on API usage, but rather relate to software evolution and maintenance tasks.

So, researchers focused on how different stages of Software Project Development and tools required can be analyzed in different ways and they classified the Questions raised in different phases into different categories based on their Experience, Observations. Knowledge Types was not generated automatically.

Here, authors referred and studied API Reference Documentation in different ways. So, Separate study of Knowledge Categorization and API Reference Documentation was done previously.

The proposed work focuses on generation of Knowledge Types from Java API Reference Documentation.

3.IMPLEMENTATION DETAILS

3.1 Block Diagram of the System

The following figure explains the Block Diagram of Proposed System:

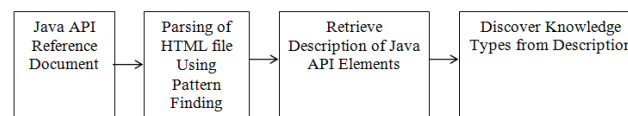


Figure 1: Block Diagram of the System

The System focuses on Java API Reference Documents that is Javadocs. Input to the system is API Reference Document of Java which is HTML Page.

This Java API Reference Document is then parsed by finding Pattern for the Tags having Description.

After finding the Patterns, the required Text is retrieved from the HTML page.

In the next step, Description of API elements is analyzed and then Knowledge Types will be generated.

Following are the Knowledge Types that are generated:

1. Functionality and Behavior: This Knowledge Type describes functionality and features of API. And also specifies what happens when the API is used.

e.g.: `protected boolean enabled`

Specifies whether action is enabled; the default is true.

2. Directives: It is related to accessibility that is what users are allowed or not allowed to do with the API element. Directives are clear contracts.

e.g.: `public class AccessException extends RemoteException`

An AccessException is thrown by certain methods of the `java.rmi.Naming` class (specifically `bind`, `rebind`, and `unbind`) and methods of the `java.rmi.activation.ActivationSystem` interface to indicate that the caller does not have permission to

perform the action requested by the method call. If the method was invoked from a non-local host, then an `AccessException` is thrown.

3. Control-Flow: How the API (or the framework) manages the flow of control is described by this knowledge type. For example by stating what events cause a certain callback to be triggered?

e.g.: `Set<String> getSupportedAnnotationTypes()`

If the processor class is annotated with `SupportedAnnotationTypes`, return an unmodifiable set with the same set of strings as the annotation.

4. Code Examples: Code examples are provided for how to use and combine elements to implement certain functionality or design outcomes.

e.g.: `public abstract class AbstractExecutorService extends Object implements ExecutorService`

Provides default implementations of `ExecutorService` execution methods. This class implements the `submit`, `invokeAny` and `invokeAll` methods using a `RunnableFuture` returned by `newTaskFor`, which defaults to the `FutureTask`.

class provided in this package. For example, the implementation of `submit(Runnable)` creates an associated `RunnableFuture` that is executed and returned. Subclasses may override the `newTaskFor` methods to return `RunnableFuture` implementations other than `FutureTask`.

Extension example. Here is a sketch of a class that customizes `ThreadPoolExecutor` to use a `CustomTask` class instead of the default `FutureTask`:

```
public class CustomThreadPoolExecutor extends ThreadPoolExecutor {
    static class CustomTask<V> implements RunnableFuture<V> { ...
    protected <V> RunnableFuture<V> newTaskFor(Callable<V> c) {
        return new CustomTask<V>(c);
    }
    protected <V> RunnableFuture<V> newTaskFor(Runnable r, V v) {
        return new CustomTask<V>(r, v);
    }
    // ... add constructors, etc.
}
```

5. **Environment:** Aspects related to the environment in which the API is used is described in this type, but not the API directly, e.g., compatibility issues, differences between versions, or licensing information.

e.g: public abstract class AbstractElementVisitor7<R,P>

```
extends AbstractElementVisitor6<R,P>
```

A skeletal visitor of program elements with default behavior appropriate for the RELEASE_7 source version.

6.External References: It includes any pointer to external documents, either in the form of hyperlinks, tagged "see also" reference, or mentions of other documents (such as standards or manuals).

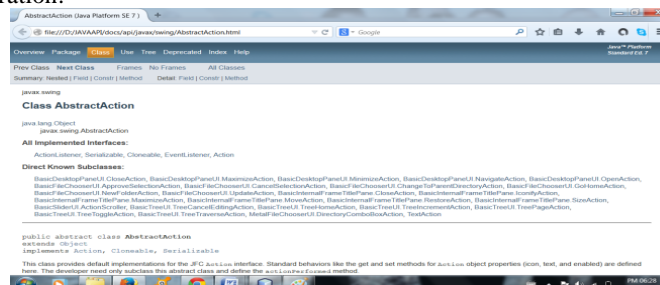
e.g: public interface DOMLocator

DOMLocator is an interface that describes a location (e.g. where an error occurred).

See also the Document Object Model (DOM) Level 3 Core Specification.

3.2 Concept in detail with example:

a) Consider following HTML file as Input: In this example, one of the class of Javadocs , named void AbstractAction is taken into consideration.



The screenshot shows a web browser displaying an API reference document. At the top, there's a navigation bar with a search icon and a 'Filter' button. Below this is a table with columns for 'Endpoint', 'Method', and 'Description'. The table lists several endpoints, including 'GET /api/v1/users', 'POST /api/v1/users', 'PUT /api/v1/users/{id}', 'DELETE /api/v1/users/{id}', 'GET /api/v1/users/{id}', 'POST /api/v1/users/{id}', 'PUT /api/v1/users/{id}', and 'DELETE /api/v1/users/{id}'. Each endpoint has a corresponding method and a brief description. The table is scrollable, and the bottom of the page shows a footer with 'Page 1 of 1' and a 'Back to Top' link.

Figure 2: Example of the API Reference Document

b) Parsing of HTML document is done using following technique:

In this technique, initially all HTML Tags are fetched from the HTML Page.

After fetching all the HTML Tags, the Tags having the required descriptions are observed.

And then the required Text is retrieved from the all HTML Tags.

For Example: To get the Description of Class , all HTML Tags are observed.

And then Pattern is detected as: Description of the Class is present under div tag having identity as <div class="block">. But here, there will be multiple div tags in one HTML page with same class="block".

So again, pattern is observed in all HTML pages of Java API Reference Documents as: Description of the Class is always present in the First tag having class="block".

And then Text is retrieved from this tag.

So after this First div tag, multiple div tags with class= "block" may be present.

c) After separating the tags having the description, next step is to generate Knowledge Types for the given API Reference Document. Here description of one API elements on the API Reference Document may fall under more than one Knowledge Types.

To generate Knowledge Types, identity of each Knowledge Type is observed.

For Example:

For generating the Functionality and Behavior Knowledge Type, Description of API Elements is considered as it is. Because Functionality and Behavior Knowledge Type describes functionality and features of API. And also specifies what happens when the API is used.

3.3 Algorithms

3.3.1 Parsing of HTML Files to fetch Description:

a) Initially, one of the Javadocs pages that is Java API Reference Document which is to be processed is taken as input.

b) Source code of the Javadocs is HTML tags and hence the actual input to the first step is HTML and JavaScript tags.

c) After taking HTML tags as input, the next step is to parse the HTML tags to fetch the tags having description.

d) So, to fetch the description of API element from the current page using Pattern Finding.

3.3.2 For Generation of Knowledge Types for the Description of API elements:

a) After fetching the description of API elements in second step, next step is to process this description.

b) To process the description of the API elements, the patterns of the Descriptions are observed, that is whether the descriptions are having some common words in them or they are starting with same words or having some common format.

c) So, after finding some common patterns in the descriptions, the Knowledge Types are generated.

That is description will be classified to the appropriate Knowledge Type. For Example, Description of all API elements will have common Knowledge Type as Functionality and Behaviour.

4. RESULTS

4.1 Data Set

The Data Set for the system are set of API Reference Documents. The jdk-7u51-apidocs.zip file contains the set of API Reference Documents for Java. The above said file can be obtained by using following link:

<http://www.oracle.com/technetwork/java/javase/documentation/java-se-7-doc-download-435117.html>

4.2 Results

Consider the API Reference Documentation in Figure 2 for Class AbstractAction.

For API Reference Document in Figure 2, following Knowledge Types are generated:

Table 1: Results obtained for AbstractAction

Other Details of Class	Description	Knowledge Types that are generated
	Class Description: This class provides default implementations for the JFC Action interface. Standard behaviors like the get and set methods for Action object properties (icon, text, and enabled) are defined here. The developer need only subclass this abstract class and define the actionPerformed method. Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has been added to the java.beans package. Please see XMLEncoder.	1.External References Knowledge Type Detected 2.Environment Knowledge Type Detected 3.Functionality and Behavior Knowledge Type Detected

Fields under above class		
protected SwingPropertyChangeSupport changeSupport	If any PropertyChangeListeners have been registered, the changeSupport field describes them.	Control Flow Functionality and Behavior
protected boolean enabled	Specifies whether action is enabled; the default is true.	Functionality and Behavior
Methods in above class		
void addPropertyChangeListener(PropertyChangeListener listener)	Adds a PropertyChangeListener to the listener list.	Functionality and Behavior
protected Object clone()	Clones the abstract action..	Functionality and Behavior
protected void firePropertyChange(String propertyName, Object oldValue, Object newValue)	Supports reporting bound property changes.	Functionality and Behavior
Object[] getKeys()	Returns an array of Objects which are keys for which values have been set for this AbstractAction, or null if no keys have values set.)	Functionality and Behavior
PropertyChangeListener[] getPropertyChangeListeners()	Returns an array of all the PropertyChangeListeners added to this AbstractAction with addPropertyChangeListener().	Functionality and Behavior
Object getValue(String key)	Gets the Object associated with the specified key.	Functionality and Behavior
boolean isEnabled()	Returns true if the action is enabled.	Functionality and Behavior
void putValue(String key, Object newValue)	Sets the Value associated with the specified key.	Functionality and Behavior
void removePropertyChangeListener(PropertyChangeListener listener)	Removes a PropertyChangeListener from the listener list.	Functionality and Behavior
void setEnabled(boolean newValue)	Sets whether the Action is enabled.	Functionality and Behavior
Constructor in above class		
AbstractAction()	Creates an Action.	Functionality and Behavior
AbstractAction(String name)	Creates an Action with the specified name.	Functionality and Behavior
AbstractAction(String name, Icon icon)	Creates an Action with the specified name and small icon.	Functionality and Behavior

5. CONCLUSION

API's are used as interface for using predefined functions, packages, classes etc. Developers read API reference documentation to learn how to use the API and answer specific questions they have during development tasks. Thus API Reference Documentation provides guide to user for referring to API. API Reference Documentation contains description of API elements; this description will be analyzed for generating Knowledge. This system focuses on classification of description of API elements into different Knowledge Types for Java API Reference Documentation.

REFERENCES

- [1] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in Proceedings of the 29th International Conference on Software Engineering, 2007, pp. 344–353.
- [2] D. Hou, K. Wong, and J. H. Hoover, "What can programmer questions tell us about frameworks?" in Proceedings of the 13th International Workshop on Program Comprehension, 2005, pp. 87–96.

- [3] D. Kirk, M. Roper, and M. Wood, "Identifying and addressing problems in object-oriented framework reuse," *Empirical Software Engineering*, vol. 12, pp. 243–274, June 2007.
- [4] J. D. Herbsleb and E. Kuwana, "Preserving knowledge in design projects: what designers need to know," in *Proceedings of the Joint INTERACT '93 and CHI '93 Conferences on Human Factors in Computing Systems*, 1993, pp. 7–14.
- [5] J. Mylopoulos, A. Borgida, and E. Yu, "Representing software engineering knowledge," *Automated Software Engineering*, vol. 4, no. 3, pp. 291–317, 1997.
- [6] J. Nykaza, R. Messinger, F. Boehme, C. L. Norman, M. Mace, and M. Gordon, "What programmers really want: Results of a needs assessment for SDK documentation," in *Proceedings of the 20th Annual ACM SIGDOC International Conference on Computer Documentation*, 2002, pp. 133–141.
- [7] J. Sillito, G. C. Murphy, and K. D. Volder, "Asking and answering questions during a programming change task," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 434–451, July-August 2008.
- [8] L. Shi, H. Zhong, T. Xie, and M. Li, "An empirical study on evolution of API documentation," in *Proceedings of the Conference on Fundamental Approaches to Software Engineering*, 2011, pp. 416–431.
- [9] M. P. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.
- [10] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Improving documentation for eSOA APIs through user studies," in *Proc. 2nd Int'l Symp. on End-User Development*, ser. LNCS, vol. 5435. Springer, 2009, pp. 86–105.
- [11] Walid Maalej and Martin P. Robillard, "Patterns of Knowledge in API Reference Documentation," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, VOL. 39, NO. X, XXXXXXXX 2013.

AUTHORS

1. Miss. Monali Metkar pursuing PG degree in Computer Engineering from K.K.W.I. E. E. R , University of Pune, India.

2. Prof. S. M. Kamalpur working as Associate Professor in Department of Computer Engineering, KKWIEER, Nashik , University of Pune, India