Identifying Idioms of Source Code Identifier in Java Context

Suntae Kim1, Rhan Jung1*

1Department of Computer Engineering, Kangwon National University, Kangwon-Province, South Korea, {stkim, jungran}@kangwon.ac.kr

Abstract. This paper presents an approach to identifying a domain word POS (Part of Speech) and idiom code identifiers written in Java programming language. To detect them, we extracted common identifiers from 14 Java API documents, and applied diverse filters. In addition, NLP (Natural Language Parser) has been used to detect common mistakes in the Java API documents. As a result, this paper identified 80 idioms from the documents.

Keywords: Idioms, Source Code Identifiers, Inconsistency

1 Introduction

A source code identifier (e.g., method or class identifiers) is one of the key means for comprehending a software system for software maintainer. Specially, it becomes more important when the system does not have enough documents describing the internal structure of the system. In naming code identifiers, developers tend to constantly use a single POS (Part-of-Speech) of a word, and common idioms that the only people possessing the domain knowledge of the application area can understand regardless. Identifying the domain terms and idioms is valuable not only for writing an understandable source code, but also for reviewing the validity of the source code identifiers.

There has been some work to improve source code readability (e.g., see [1][5]). Most studies are based on software metric, measuring extent of readability using indicators such as line length of a method, number of comments and keywords. Although the approach is helpful to characterize quality of the source code, software developers do not have any concrete hints or guidelines while naming source code elements.

In this paper, we present an approach to identifying a domain word POS and idiom code identifiers written in Java programming language. We define an idiom as a word or a combination of words that violates grammar rules of Java Naming Convention throughout diverse Java projects. In addition, the domain word POS indicates a specific word POS that only used in a specific domain. To identify idioms and a domain word POS, we use the Java API (Application Programming Interface)
Document as a input, and grammatically analyze all identifiers of classes, methods and attributes introduced in the API documents. Then, domain words that are usually discovered in diverse projects are detected, and their POS are mined through several filters. Also, we check violation of an identifier against the grammar rules, so that idioms are detected if the violation is commonly discovered throughout several java projects. In identifying the domain word POS and idioms, we have utilized popular 14 Java API documents including Java SE(Standard Edition) and various Apache projects, which can be considered as core libraries for Java based software systems.

2 Identifying Idioms of Source Code Identifiers

This section introduces how we automatically identify a domain word POS and idioms. Fig. 1 shows the overview of our approach. It first starts with analyzing 18 java or library API documents. Then, it tokenizes all identifiers of classes, methods and attributes and parses each identifiers with a NLP(Natural Language Processing) parser[6] so that the results are stored in the intermediary storage. Based on the results, domain words and its POS are identified, and idiom identifiers are also extracted via passing them through several filters.

Step1. Parsing API Documents

Domain words and Idioms are extracted fromJava or Common LibraryAPI(Application Programming Interface) documents. Obtaining code identifiers from API documents is easier and more reliable approach rather than examining source code, because code identifiers from API are officially suggestible to
users of the API. We collected class, method, and attribute identifiers from 14 API documents as shows in Table 1. Due to the space limit, we just listed the representative libraries.

<table>
<thead>
<tr>
<th>Library</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Development Kit</td>
<td>1.7.0</td>
<td>Java standard development kit</td>
</tr>
<tr>
<td>Apache Ant</td>
<td>1.8.3</td>
<td>Java library and command-line tool for building java project</td>
</tr>
<tr>
<td>Apache POI</td>
<td>3.9</td>
<td>APIs for manipulating various file formats based upon Microsoft</td>
</tr>
</tbody>
</table>

Table 1. Java/Library API Documents for the Code Dictionary

Then, an NLP parser assigns POS to each word composing of an identifier. To parse an identifier using an NLP parser, an identifier is tokenized first according to rules from the camel case[3]. Then, a blank is inserted between tokenized terms, and a period is inserted at the end of the last term especially for a method identifier to make a complete sentence. This is because a method identifier constitutes a verb phrase according to the Java Naming Convention[2], which is a complete sentence without a subject. The converted identifier is transferred into a NLP parser to analyze POS of each term. As an example, a method identifier `getWordSet()` is converted into ‘get word set.’. Then, the parser analyzes POS of the sentence, resulting in ‘[(VP (VB get) (NP(NN word)(NN set)(..)))]’, where VP, VB, NP and NN denote a verb phrase, a verb, a noun phrase and a noun respectively. The converted identifier is transferred into a NLP parser to analyze POS of each term. As an example, a method identifier `getWordSet()` is converted into ‘get word set.’. Then, the parser analyzes POS of the sentence, resulting in ‘[(VP (VB get) (NP(NN word)(NN set)(..)))]’, where VP, VB, NP and NN denote a verb phrase, a verb, a noun phrase and a noun respectively. The results are stored in each model as an intermediate storage.

**Step2. Detecting Domain Word POS**

Domain words and its POS cover computer domain specific words. The word ‘file’, as an example, is frequently used as a noun or a verb in a natural language, while it is generally used only as a noun indicating the way of storing data in the second storage in a computer domain. Detecting domain word and its POS consists of two sub-steps. The first step is that words discovered less than 100 times are filtered out among all candidate words. This is because domain words are used frequently rather than other non-domain words in source code identifiers. As the second step, if a POS in POSs of a word occupies 90% of total usage, the POS is considered as a dominated POS of the word and its POS is identified as a domain word. This is because a domain word are generally used as a single dominated POS.

Fig. 2 shows the steps with sample words. All words extracted from the API documents are 14,642 words, each word includes number of POS usages (see left part in the figure). In the figure, `nn, vb, adj and adv` denotes a noun, a verb, an adjective and an adverb respectively. After passing the first 100 Uses Filter, words having a dominant POS pass the second 90% Filter. With the two filters, we have obtained 183 domain words and its pos including `thread` and `file`. The domain words and its POS are used to increase precision of the NLP parser and inconsistent identifiers.
Step 3. Detecting Idiom Identifiers

An idiom identifier is a single or compound identifier that commonly violates grammatical rules of Java naming conventions. Due to its common usages, they are generally acceptable regardless of violation of the grammatical rules. The method `keySet()` and `length()` are a noun phrase or a noun respectively. However, the Java naming convention guides that the method should be a verb or a verb phrase. Building idiom identifiers are motivated by observation of the code. We discovered that a large number of identifiers does not observe the grammatical rules, even Java Development Kit[4] contains many identifiers violating the rules, and developers does not even think that is violation. Thus, we created idiom identifiers to reflect it to increase precision of inconsistent identifier detection.

Building idiom identifiers consists of two steps as shows in Fig. 3. It starts with results from parsing API documents, which contains 98,320 identifiers. They passes through the Frequency Filter which filters out identifiers that are not discovered throughout at least three API documents, and less than three usages for a class and an attribute, ten usages for a method. The identifiers that passed through the filter are 800. The second step filter is Phrase-POS Filter detecting identifiers that violate grammatical rules. As a result, we extracted 80 idiom identifiers including `abs()`, `available()`, `elements()`, `intValue()`, `length()` and `values()` as representative method idiom identifiers, `ALL`, `debug` and `verbose` as representative attribute idiom identifiers.

Fig. 2. Detecting Domain Words and Its POS

Fig. 3. Detecting Idiom Identifiers
3 Conclusion and Future Work

In this paper, we have presented an approach to identifying domain words and its POS, and idioms frequently discovered throughout diverse Java or Java based API documents. Based on the the domain words and idioms, we strongly believe that software developers can adopt the domain terms while developing their application of the application area, and software maintainers can comprehend the system in an easier manner. This research is also valuable as one of the steps to increase the quality of the source code identifiers. As a future work, we are planning to apply this approach to detect inconsistent identifiers of the Java based software systems.

References