静岡大学情報学部情報社会学科 卒業研究

品詞に着目した識別子の命名方法の提案

鳥居　克哉（7071–1041）

2021年1月

指導教員：◯◯ ○○［12P、MS明朝］

卒業研究要旨

[16P、HGPゴジックM]

執筆にあたっては、フォントサイズを10Ｐとし、フォントはMS明朝体を用いる。
卒業研究要旨のページ番号は、ローマ数字で表示する。

目次[16P]

執筆にあたっては、フォントサイズを10Ｐとし、フォントはMS明朝体を用いる。

序論

インターネットが我々の生活にとって欠かせない存在になるにつれて、ソフトウェアは社会の様々な場面で活用されるようになった。それに伴い、ソフトウェアに対する要求は増大し、ソフトウェアは大規模化や複雑化が進んでいる。そして、ソフトウェアの大規模化や複雑化に伴い、ソフトウェア保守にかかるコストが増大するという問題が発生している。

一般に、ソフトウェア開発は複数のソフトウェア開発者間でソースコードを共有する。それゆえ、ソースコードの可読性はソフトウェア保守のコストに影響を与える因子の一つである。可読性もまた、様々な因子によって成り立っている。その因子の一つに識別子が挙げられる。

ほとんどのプログラミング言語において、識別子はソフトウェア開発者が自由に命名をすることができる。識別子の命名は時に、ソフトウェア開発者の気まぐれによって行われることもある。それゆえ、ソフトウェア開発のプロジェクトごとに識別子の命名規則が定められることがある。命名規則では識別子の命名方法について様々な規則が定められるが、しばしば品詞に関わる規則が定められる。

そこで、本研究では識別子中の品詞と可読性の関係について調査をし、識別子の命名手法を提案する。

第1章では、研究の背景について述べる。第2章では調査方法について述べる。第3章では調査によって得られたデータの評価を行う。第4章では第3章を踏まえ、識別子の命名手法の提案を行う。

1．研究の背景

1.0　ソースコードにおける可読性

可読性とは、テキストがどれだけわかりやすいかという人間の判断のことである[6]。これより、ソースコードにおける可読性とは、ソースコードによって表現されたプログラムの内容が、人間にとってどれだけわかりやすいかという指標であると言える。

ソースコードの可読性に影響を与える因子は数多く存在する。例えば、○○の研究ではソースコード中のコメントは可読性の向上に貢献することを明らかにした[4]。また、○○の研究では適切な改行によって、ソースコードの可読性が向上することを明らかにした[5]。ソースコードにおけるコメントは、ソースコードへの注釈であり、プログラムの実行結果に影響を及ぼさない。また、Pythonのように、インデントによるコードブロック形成を行わないプログラミング言語において、ソースコード中の改行は、プログラムの実行結果に影響を及ぼさない。これより、プログラムの動作に影響を与えない要素によって、ソースコードの可読性は実現されることがわかる。

1.1　ソフトウェア保守と可読性の関係

ソフトウェアは、ソフトウェアの開発が企画されてから運用が終了するまでを、いくつかの工程に分けることができる。この工程全体をソフトウェアライフサイクルという。ソフトウェアライフサイクルは、「企画」、「要件定義」、「開発」、「運用」、「保守」という工程に分けることができる。そして、ソフトウェアの保守作業は、ソフトウェアライフサイクルのコストの大半を占める。BohemとBasiliは、調査によって、ソフトウェアの保守作業がソフトウェアライフサイクルのコストのうちおよそ70％を閉めることを明らかにした[1]。

ソフトウェアの保守作業はバグの修正や、プログラム外部的な振る舞いを保ちつつ、内部的構造を変化させる一連の作業である[7]「リファクタリング」などを行う工程である。このように、ソフトウェアの保守作業の多くがプログラムの変更を伴う作業であることから、ソフトウェアの保守作業にかかるコストの多くは、ソフトウェア開発者の人件費によるものであると考えられる。

ソフトウェアの保守作業のしやすさのことを保守性という。保守性が高く保たれているソフトウェアは、保守性が高く保たれていないソフトウェアに比べて、ソフトウェアの保守作業にかかる時間が少なくなると考えることができる。これより、ソフトウェアの保守性と高く保つことで、ソフトウェアの保守コストを削減することができると言える。ソフトウェアの保守性に影響を及ぼす要素には、さまざまなものが挙げられる。その要素の一つとして、ソフトウェアのソースコードの品質が考えられる。ソースコードとは、プログラミング言語によって記述された、プログラムを表現するテキストのことである。

ソースコードの品質にはさまざまな指標が存在する。ソースコードの品質は、ソースコードを解析するソフトウェアメトリクスによって計測することができる。ソースコードを解析するソフトウェアメトリクスの代表的なものとして、プログラムの制御フローの複雑さを計

測するMeCabeメトリクス(再クロマティック数)などが挙げられる[8]。

ソースコードの品質に影響を与える要素は他にも存在し、先にあげたもの以外に、ソースコードの可読性(Readability)が考えられる。　一般にソフトウェアの保守作業はソースコードを読解する作業が伴う。ソースコードの可読性が高いほど、ソフトウェア開発者がソースコードの読解に費やす時間を削減することができると考えられる。つまり、可読性の向上によって、ソフトウェアの保守コストの削減が期待できると考えられる。また、可読性は、ソフトウェアの品質に影響を与えることがわかっている。Butler他は、ソースコードの可読性がソフトウェアの品質に影響を与えることを明らかにした[2]。これより、ソースコードの可読性は、ソフトウェア開発において重要な指標であると言える。

1.2　識別子

識別子とは、「プログラム言語における変数名や関数名を表す名前」[10]である。識別子は、関数や、変数のようなプログラムを構成する要素を一位に識別するためのものである。ほとんどのプログラミング言語において、識別子名はソフトウェア開発者が自由に決定することができる。識別子名には単一の単語や、複数単語をつなぎ合わせたものが使用されることがある。一般に、識別子が指すプログラムの構成要素を表現するように識別子を命名することが推奨される。

1.3　識別子名における単語の連結の形式

一般的なプログラミング言語では、識別子名に空白文字(スペース等)を含めることはできない。それゆえ、識別子の命名に複数の単語を用いる場合は何らかの形式に従って単語を連結する必要がある。Table1は複数の単語を連結する際の代表的な形式である。

形式名	形式	例
アッパーキャメルケース	全ての単語の先頭を大文字にして連結する	UpperCamelCase
ロワーキャメルケース	最初以外の単語の先頭を大文字にして、連結する	lowerCamelCase
アッパースネークケース	全ての文字を大文字にして、アンダースコアで連結する	UPPER_SNAKE_CASE
ロワースネークケース	全ての文字を小文字にして、アンダースコアで連結する	lower_snake_case
アッパーケバブケース	全ての文字を大文字にして、ハイフンで連結する	UPPER-KEBAB-CASE
ロワーケバブケース	全ての文字を小文字にして、ハイフンで連結する	Lower-kebab-case

Table1: 単語の連結方法

1.4　命名規則

1.2で述べたように、識別子名はソフトウェア開発者が自由に決定することができる。し

かし、その自由さが問題となることがある。Sneedはソフトウェア開発者の気まぐれで好きなスポーツ選手などの名前が、識別子につけられることを指摘している[12]。このような問題を防ぐために、プロジェクトによっては「命名規則」が導入されることがある。

命名規則とは、どのように識別子を命名するかを定めたものである。命名規則では識別子名に用いる単語を指定したり、複数単語を連結する際の形式、識別子の文字数、識別子の品詞など様々なものがあり、命名規則によってその指定の内容は異なる。

1.5 javaの言語機能

本項では、javaの言語機能について述べる。

1.5.1 javaの識別子

javaのプログラムは、「Token」、「Comment」、「WhiteSpace」という3つの最小単位によって構成される。さらに、Tokenはプログラムの構文的特性から「Identifier」、「Keyword」、「Separator」、「Operator」の4つに分類することができる[13]。これより、識別子はTokenの一種であり、プログラムの最初単位のであるとわかる。

分類	詳細
Identifier	識別子
Keyword	ASCII文字から成る50種類の予約後
Separator	区切り文字 ( ) { } [ ] ; , ... @ ::
Operator	ASCII文字から成る38種類の演算子

Table2: javaのTokenの分類

javaの識別子(Identifier)は、識別子が指すプログラム要素によって分類することができる。本研究では、識別子が指す対象によって「クラス識別子」、「メソッド識別子」、「変数識別子」の3つに識別子を分類する。Table3はその分類である。

javaの識別子名には、大文字小文字のアルファベット数字、ドル記号、アンダースコアを使用することができる[13]。これよりjavaは、開発者が識別子名を自由に決定することができるプログラミング言語であると言える。

分類	識別子が対象とする要素
クラス識別子	クラス
メソッド識別子	メソッド

変数識別子	フィールド変数、ローカル変数

Table3: 本論文におけるjavaの識別子の分類

1.5.2　クラス

オブジェクトはデータおよび、データを操作する手続きをパッケージ化したものである。オブジェクト指向言語では、オブジェクトを扱うが、オブジェクトはクラスをインスタンス化することで生成する。オブジェクトの実装はそのクラスによって定義される。クラスはオブジェクトの内部データとその表現を明記し、オブジェクトが実行するオペレーションを定義する[14]。これより、クラスはオブジェクトを生成するための雛形のようなものだと考えることができる。

javaには通常クラス(NormalClass)と列挙子クラス(EnumClass)が存在する。本研究では通常クラスをクラスと呼ぶことにする。また、javaではクラスの宣言にに修飾子(Modifier)を含めることができる。修飾子とは、識別子の性質を決める役割を持ったキーワードである。

javaのクラスはDeclaration1のように宣言される。

1.5.3　メソッド

メソッド(Method)とは、オブジェクトによってパッケージ化された、データを操作する手続きのことである。メンバー関数とも言われることがある。javaにおいて、メソッドの宣言はDeclaration2のように行われる。クラスの宣言と同様に、メソッドの宣言にも修飾子を含めることができる。

1.5.4　変数

javaの変数には、ローカル変数とフィールド宣言が存在する。一般にローカル変数はメソッド内やforブロック内など、短いスコープで使用される。一方、フィールド変数はクラスのメンバとして宣言される。フィールド変数はクラスのメンバからアクセスされる。また、クラスの外からもアクセスされることがある。したがって、フィールド変数はローカル変数よりも大きなスコープで使用される変数であると言える。ローカル変数の宣言はDeclaration3、フィールド変数の宣言はDeclaration4のように行われる[13]。

```
[Modifier] class [クラス名(Identifier)] {
        [クラスの実装]
}

例: Carという名前のクラスを宣言する場合
class Car {
}
```

Declaration1:クラスの宣言

```
[Modifier] [戻り値の型(Type)] [メソッド名(Identifier)] ([一つ以上の引数]) {
        [メソッドの実装]
}

例: Carクラスに戻り値・引数を持たない"run"というメソッドを宣言する場合
class Car {
        void run() {
        }
}
```

Declaration2:クラスの宣言

```
[Modifier] [変数の型] [フィールド変数名[Identifier];

例: Carクラスに文字列を格納する"text"というフィールド変数を宣言する場合
class Car {
        String text;
}
```

Declaration4:フィールド変数の宣言

```
[変数の型] [ローカル変数名(Identifier)];

例: 文字列を格納する"text"という名前のローカル変数を宣言する場合
String text;
```

Declaration3:ローカル変数の宣言

1.5.5　アクセス修飾子

javaには「スコープ(Scope)」が存在する。スコープとは、ある識別子を参照できる範囲のことである。javaではクラス、メソッド、フィールド変数の宣言にアクセス修飾子を含めることができる。識別子の宣言にアクセス修飾子を含めることで、スコープを指定することができる。javaのアクセス修飾子は3種類存在する。

アクセス修飾子	詳細
public	全てのクラスからアクセスが可能
private	現在のクラスのみからアクセスが可能
protected	現在のクラスとサブクラスからのみアクセスが可能
なし(標準)	現在のパッケージからのみアクセスが可能

Table4: javaのアクセス修飾子

1.6　javaで推奨される識別子の命名

javaでは、開発者が識別子名を自由に決定することができるが、この自由さが問題となることがある[12]。このような問題を防ぐために命名規則が定められることがある。javaはOracle社によって提供されているプログラミング言語だが、オラクル(Oralcle)社が推奨する命名規則が存在する[13]。この命名規則では、単語の連結の形式と品詞について述べられていた。Table5は識別子の種類と単語の連結の形式と品詞についてまとめた表である。

識別子の種類	単語の連結の形式	品詞
クラス識別子	アッパーキャメルケース	名詞、名詞句
メソッド識別子	ロワーキャメルケース	動詞、動詞句
変数識別子	ロワーキャメルケース	名詞、名詞句

Table5: javaで推奨される命名規則

1.7　識別子と可読性の関係

識別子と可読性の関係について、様々な研究が行われている。例えば、Johannes，Janet，Danielは、識別子名に単語を使用した場合、文字を使用した場合に比べてソースコードの理解速度が有意に速くなることを明らかにした[3]。これより、識別子によって可読性を向上させることが可能であることがわかる。

ソースコードは、その大半が識別子によって構成されている。DeissenboeckとPizkaは調査

によって、ecliplse3.1.1 のソースコードの全文字数のうち71.5%が識別子によるものであることを明らかにした[9]。Source1はjunit4のソースコード[11]の一部を抜粋したものである。このソースコードはjavaによって記述されている。Table6はSouce1中に出現する全て識別子を分類した表である。Source1は14行のソースコードだが、6個の識別子が含まれており、識別子がソースコードの多くを占めていることがわかる。これより、識別子が可読性に大きな影響を持っていると考えられる。

```
public class Assert {
    /**
     * Protect constructor since it is a static only class
     */
    protected Assert() {
    }

    /**
     * Asserts that a condition is true. If it isn't it throws
     * an AssertionFailedError with the given message.
     */
    static public void assertTrue(String message, boolean condition) {
        if (!condition) {
            fail(message);
        }
```

Source1: junit4から抜粋したソースコード

識別子の種類	識別子
クラス識別子	Assert
メソッド識別子	assertTrue, fail
変数識別子	message, condition
その他の識別子	Assert

Table6: Source1中に出現する識別子

1.8　品詞と可読性の関係

品詞(Part-Of-Speech)は、自然言語における語の文法的な分類である。そして、品詞間には

文法的な関係が存在する[15]。

しばしば、命名規則では品詞に関する規則が定められることがある。1.6ではjavaにおける命名規則の中に品詞があることを確認した。プログラミング言語は、自然言語とは異なる構文を持った言語である。

命名規則がソースコードの可読性を向上させる目的で導入されることと、識別子の命名が品詞に基づいて行われることから、品詞にはソースコードの可読性を向上させる効果があると考えられる。

1.9 オープンソース・ソフトウェアと可読性の関係

オープンソース・ソフトウェア(Open Source ・ Software)とは、誰もが検査、修正、拡張できるソースコードを持つソフトウェアのことである[16]。一般に、オープンソース・ソフトウェアの開発では、誰もがソフトウェアを検査、修正、拡張できるという特徴から、不特定多数のソフトウェア開発者が関わることが多い。例えば、javaのテストライブラリであるjunitは150人のソフトウェア開発社によってメンテナンスされている[11]。これより、オープンソース・ソフトウェアのソースコードは、不特定多数のソフトウェア開発社に共有されていることがわかる。不特定多数の、オープンソース・ソフトウェアではソースコードの可読性が重要であると考えられる。

オープンソース・ソフトウェアの開発では、ソフトウェアの品質を維持するため、ソースコードの変更を統合する前に、変更内容を検証する「レビュー」という開発プロセスが存在する。このレビューは、ソースコードの変更者ではない、ソフトウェア開発者によって行われる。それゆえ、オープンソース・ソフトウェアではレビューという開発プロセスによってソースコードの可読性が維持されていると考えられる。

2． 調査手法

本節では、調査手法について述べる。識別子中の品詞間の関係を調べるために、javaのオープンソース・ソフトウェアから識別子を収集した。177のプロジェクトを対象に調査を行った。

2.1　調査対象

識別子から品詞間の関係を調べる上で、可読性が高いソースコードから識別子を抽出する必要があった。1.9で述べたように、オープンソース・ソフトウェアのソースコードは可読性が高いことが期待される。それゆえ、数多くのオープンソース・ソフトウェアが集まるGithub上のjavaプロジェクトを調査対象とした。調査対象としたオープンソース・ソフトウェアのプロジェクトの数は177である。

2.2　識別子の抽出方法

ソースコードから識別子を抽出するにあたって、AST(Abstract Syntax Tree)解析を用いた。ASTとは、javaオブジェクト表現としてjavaソースコードと対話することを可能にしたオブジェクト表現である[17]。javaのAST解析を行う代表的なライブラリにはeclipse JDTと、JavaParserが挙げられる[17][18]。本研究ではJavaParserを用いた。

ASTはソースコードを木構造として表現しており、この木構造はファイル全体を表すルート(root)と、複数のノード(node)を持っている。ルートから木構造をたどることで、ソースコードの任意の場所へとアクセスすることができる。Figure1はJavaParserを用いて、Source2に対しAST解析を行った結果を、ツリーダイアグラムとして表現した図(一部省略)である。Figure1を見ると、CompilationUnitをルートに持つ木構造になっていることがわかる。

本研究ではこのAST解析を行うことで、クラス識別子、メソッド識別子、変数識別子を抽出した。

2.3　識別子名の分割

本研究では品詞の関係を調査するために、識別子から単語を取得する必要がある。通常の英語であれば、単語は空白によって区切られている。しかし、1.3で述べたように、javaの識別子名は複数の単語が連結されている。それゆえ、識別子名を何らかの方法で単語に分割する必要がある。

javaには識別子の種類によって推奨される単語の連結の形式が存在する[13]。そこで、本研究では正規表現を用いて単語を分割した。Source3は本研究で、識別子中の単語を分割する際に使用した正規表現である。また、javaでは識別子名にアンダースコア

を使用可能であることから、アッパースネークケース・ロワースネークケースの識別子に対応ができるよう、アンダースコアが含まれる識別子名はアンダースコアで、識別子を分割した。

```
package com.github.javaparser;

import java.time.LocalDateTime;

public class TimePrinter {

  public static void main(String args[]){
    System.out.println(LocalDateTime.now());
  }
}
```

Source2: AST解析を行うソースコードの例

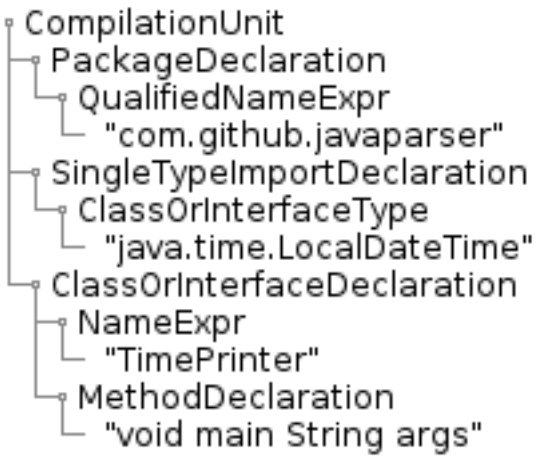


Figure1: Source2のAST解析によって生成された木構造

2.4 品詞の取得

識別子に含まれる単語の品詞の関係を調査する上で、識別子に含まれる単語の品詞を取得する必要がある。そこで、本研究ではpythonのNLTK(Natural Language Toolkit)を用いて品詞を取得した。NLTKは、自然言語を扱うための様々な関数が用意された、Pythonのモジュールである[18]。NLTKで品詞を取得する際、タグセットを指定することができる。今回はNLTKの品詞を取得する関数で標準とされていた、ペンツリーバンク(PennTreebank)のタグセットを用いた[19]。Table8はペンツリーバンクのタグセットの記号とその詳細である[20]。

```
([a-z]+)([A-Z][a-z]+)|([A-Z][a-z]+)
```

Source3: 識別子中の単語の分割に使用した正規表現

タグセットの記号	詳細
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Table8:PennTreebank tagset一覧

3． 調査結果

本節では、2の調査によって得られたデータについて述べる。調査によっておよそ500万個の識別子を取得することに成功した。Table7は調査によって得られたデータの概要である。

識別子の種類	抽出した識別子の数(個)	抽出した単語の数(個)
クラス識別子	365505	126757
メソッド識別子	2729915	7351610
変数識別子	2021672	3576170
合計	5117092	12195355

Table7:抽出した識別子の数

3.1 プロジェクトによる調査結果への影響の検証

TODO: 特定のリポジトリが結果の大半を占めてしまった。影響をなくすために何らかの手法を使ってデータを均すor影響がないことを確認

3.2 品詞の出現頻度

本節では、調査結果をもとに品詞の出現頻度と識別子との関係について述べる。

3.2.1 全体における品詞の出現頻度

得られたデータから、品詞の数について集計をした。使用頻度の少数点第2位に対し、四捨五入を行った。FIgure2は品詞の出現頻度について集計したデータをもとに作成したグラフである。

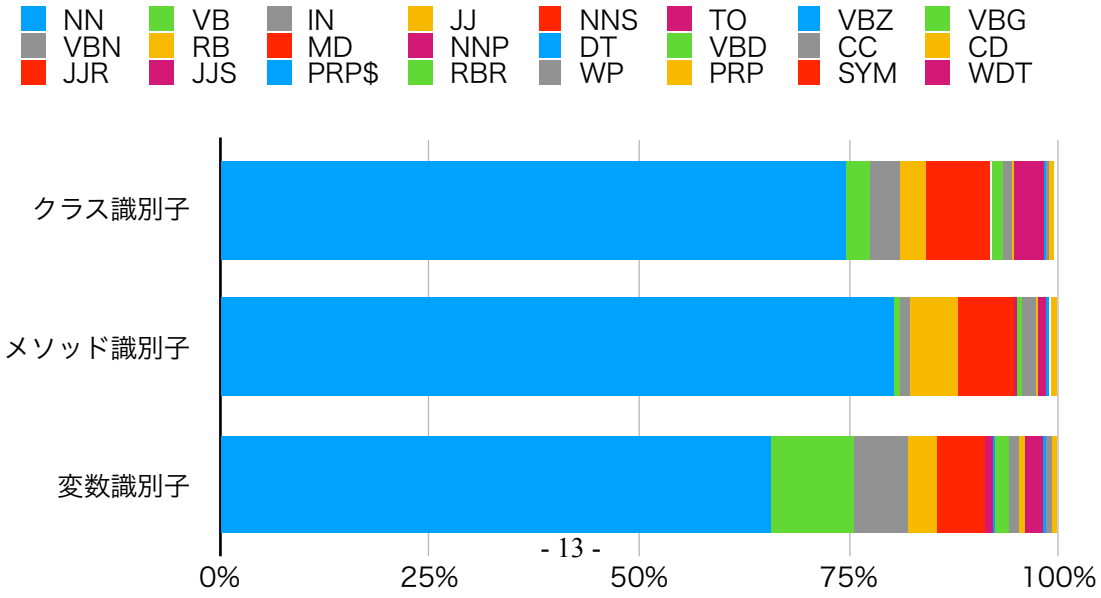


Figure2: 品詞の使用頻度

このグラフから、クラス識別子、メソッド識別子、変数識別子の全てにおいて、NNの出現頻度が極めて高いことがわかる。一方、VBは変数識別子において、出現頻度が高いことがわかる。javaでは、メソッド名が動詞・動詞句になるように命名することが推奨されていた[13]。しかし、調査結果からメソッド識別子におけるVB, VBD, VBG, VBN, VBP, VBZの出現頻度が低いことがわかった。<!--FIXME: 単語の位置を踏まえつつ議論する-->

3.2.1 単語の位置と品詞の出現頻度

3.3 マルコフモデルによる品詞間の関係

プログラミング言語は自然言語と異なった構文を持つ言語である。それゆえ、識別子名中における品詞の関係は、自然言語のそれとは異なると考えられる。そこで、識別子名中の品詞がマルコフ性を持つと仮定し、調査結果をもとにマルコフモデルを作成した。マルコフ性とは、現在の状態Xnが与えられた時、過去のいかなる 情報(X0，X1，… ，Xn−1)も、Xn+1を予測する際には 無関係であるという性質である[21]。

3.3.1 調査結果に基づくマルコフモデルの作成

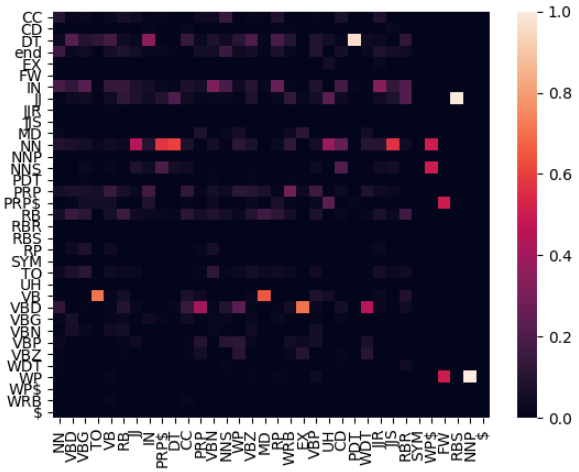
TODO:

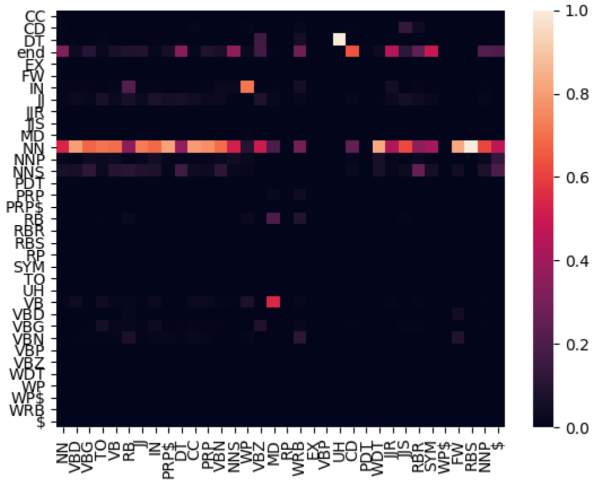
3.3.2 自然言語テキストをもとに作成したマルコフモデルとの比較

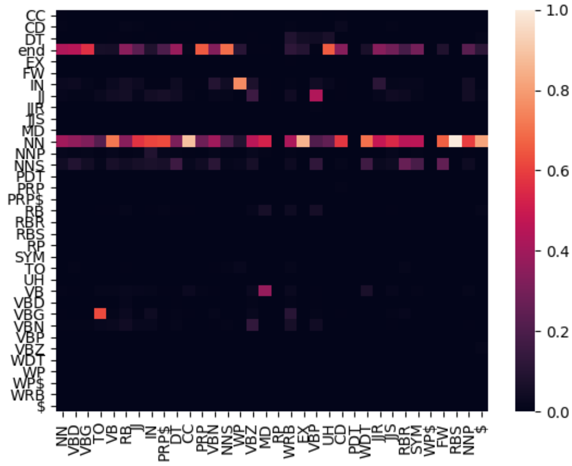
TODO;

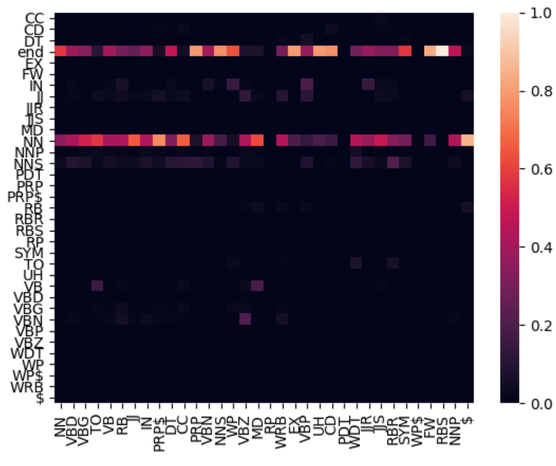
4． 命名手法の提案

本節では、品詞の関係に基づく識別子の命名手法の提案を行う。


CC
CD
DT
end
FX
FW
IN
J
JJ
JJS
MD
NN
NNP
NNS
PDT
PRP
PRP$
RB
RBR
RBS
RP
SYM
TO
UH
VB
VBD
VBG
VBN
VBP
VBZ
WDT
WP
WP$
WRB
$
NN
VBD
VBG
TO
VB
RB
J
IN
PRP$
DT
CC
PRP
VBN
NNS
WP
VBZ
MD
RP
WRB
FX
VBP
UH
CD
PDT
WDT
JJR
JJS
RBR
SYM
WP$
FW
RBS
NN$
$
1.0
0.8
0.6
0.4
0.2
0.0


CC CD DT end EX FW IN JJ JJR JJS MD NN NNP NNS PDT PRP PRP$ RB RBR RBS RP SYM TO UH VB VBD VBG VBN VBP VBZ WDT WP WP$ WRB $
NN VBD VBG TO VB RB JJ IN PRP$ DT CC PRP VBN NNS WP VBZ MD RP WRB EX VBP UH CD PDT WDT JJR JJS RBR SYM WP$ FW RBS NNP $
1.0 0.8 0.6 0.4 0.2 0.0


CC CD DT end EX FW IN JJ JJR JJS MD NN NNP NNS PDT PRP PRP$ RB RBR RBS RP SYM TO UH VB VBD VBG VBN VBP VBZ WDT WP WP$ WRB $
NN VBD VBG TO VB RB JJ IN PRP$ DT CC PRP VBN NNS WP VBZ MD RP WRB EX VBP UH CD PDT WDT JJR JJS RBR SYM WP$ FW RBS NNP $
1.0 0.8 0.6 0.4 0.2 0.0


CC CD DT end EX FW IN JJ JJR JJS MD NN NNP NNS PDT PRP PRP$ RB RBR RBS RP SYM TO UH VB VBD VBG VBN VBP VBZ WDT WP WP$ WRB $
NN VBD VBG TO VB RB JJ IN PRP$ DT CC PRP VBN NNS WP VBZ MD RP WRB EX VBP UH CD PDT WDT JJR JJS RBR SYM WP$ FW RBS NNP $
1.0 0.8 0.6 0.4 0.2 0.0

１．　　卒論の書式について

３行あけて４行目から本文を書く。
ページ番号は算用数字を使う。
執筆にあたっては、フォントサイズを10Ｐとし、フォントはMS明朝体を用いる。

1. **表紙について（紙媒体で提出の場合）**
   表紙は2部作成し、1部を「フラットファイル」に貼り付け、他の1部は論文冒頭（卒業研究要旨の前）に使う。
2. **ページ設定**
   余白上35mm、下30mm、左35mm、右30mmとする。1ページは40字35行。以下同様。
3. **脚注について[1]**
   下記の脚注1の手順に従う。脚注のフォントは9とする。

---

[1] 脚注は次の手順で挿入する。

1. 脚注番号を挿入したい位置をクリックする。

2. [挿入] メニューの [脚注] をクリックする。

3. [脚注] または [文末脚注] をクリックする。

4. [番号の付け方] で必要なオプションをクリックする。

5. [OK]をクリックする。

6. 脚注番号が挿入され、脚注ウィンドウ枠内にカーソルが移動する。

   脚注または文末脚注の内容を入力する。本文に戻り、入力を続ける。

結語

３行あけて４行目から本文を書く。
ページ番号は算用数字を使う。
執筆にあたっては、フォントサイズを10Ｐとし、フォントはMS明朝体を用いる。

Azure/azure-sdk-for-java	jfinal/jfinal	moquette-io/moquette	ipinfo/java
aws/aws-sdk-java	zxing/zxing	oldmanpushcart/greys-anatomy	vdurmont/emoji-java
aliyun/aliyun-openapi-java-sdk	google/gson	echoTheLiar/JavaCodeAcc	KikiLetGo/VirusBroadcast
bytedeco/javacpp-presets	searchbox-io/Jest	bytedeco/javacpp	joeyajames/Java
quarkusio/quarkus	rabbitmq/rabbitmq-java-client	biezhi/learn-java8	kevinsawicki/http-request
bcgit/bc-java	dromara/soul	0opslab/opslabJutil	javagrowing/JGrowing
google/guava	ScottOaks/JavaPerformanceTuning	egzosn/pay-java-parent	VerbalExpressions/JavaVerbalExpressions
SonarSource/sonar-java	scribejava/scribejava	qiyuangong/leetcode	biezhi/30-seconds-of-java8
radcortez/javaee7-angular	pubnub/java	ChinaLHR/JavaQuarkBBS	in28minutes/JavaWebApplicationStepByStep
alibaba/fastjson	json-iterator/java	crossoverJie/JCSprout	enhorse/java-interview
bjmashibing/java	EleTeam/Shop-for-JavaWeb	jwtk/jjwt	shakeelstha/java
kubernetes-client/java	phishman3579/java-algorithms-implementation	natural/java2python	SquareSquash/java
ramram43210/Java	appium/java-client	micromasterandroid/java	sendbird/SendBird-JavaScript
apache/servicecomb-java-chassis	exercism/java	HelloWorld521/Java	Snailclimb/JavaGuide
libgdx/libgdx	DreamCats/JavaBooks	mrniko/netty-socketio	Nirman-Rathod/Java
tensorflow/java	eclipse/paho.mqtt.java	NanoHttpd/nanohttpd	CarpenterLee/JavaLambdaInternals
querydsl/querydsl	java-decompiler/jd-gui	google/google-java-format	shekhargulati/strman-java
grpc/grpc-java	hub4j/github-api	shyiko/mysql-binlog-connector-java	nramnad/java
java-native-access/jna	JustinSDK/JavaSE6Tutorial	qiyaTech/javaCrawling	jenkins-docs/simple-java-maven-app

mongodb/mongo-java-driver	chenhaoxiang/Java	javaee-samples/javaee8-samples	MindorksOpenSource/from-java-to-kotlin
javaparser/javaparser	googleapis/google-cloud-java	android-async-http/android-async-http	MicrosoftDocs/pipelines-java
tronprotocol/java-tron	TheAlgorithms/Java	onblog/JavaMonitor	h2pl/Java-Tutorial
skylot/jadx	Vedenin/useful-java-links	alibaba/p3c	SuperMap/iClient-JavaScript
watson-developer-cloud/java-sdk	chanjarster/weixin-java-tools	notnoop/java-apns	17mon/java
iluwatar/java-design-patterns	byhieg/JavaTutorial	tipsy/javalin	mercyblitz/jsr
mybatis/mybatis-3	code4craft/webmagic	bottleleung/Java	udacity/Just-Java
shopizer-ecommerce/shopizer	qos-ch/slf4j	winterbe/java8-tutorial	
fabric8io/kubernetes-client	json-path/JsonPath	square/javapoet	
javaee-samples/javaee7-samples	apereo/java-cas-client	akshitagit/JAVA	
jboss-javassist/javassist	RichardWarburton/java-8-lambdas-exercises	AllAlgorithms/java	
GoogleCloudPlatform/java-docs-samples	redis/jedis	nanchen2251/RxJava2Examples	
graphql-java/graphql-java	in28minutes/JavaInterviewQuestionsAndAnswers	yasserg/crawler4j	
docker-java/docker-java	DiUS/java-faker	auth0/java-jwt	
realm/realm-java	lenve/JavaEETest	Blankj/awesome-java-leetcode	
DuGuQiuBai/Java	hamcrest/JavaHamcrest	otale/tale	
gitblit/gitblit	mrdear/JavaWEB	stleary/JSON-java	
aliyun/aliyun-oss-java-sdk	structurizr/java	amitshekhariitbhu/RxJava2-Android-Samples	
brianway/java-learning	googlemaps/google-maps-services-java	hmkcode/Java	
GoogleContainerTools/jib	TooTallNate/Java-WebSocket	kaushikgopal/RxJava-Android-Samples	
thaycacac/java	JeffLi1993/java-core-learning-example	AllenDowney/ThinkJavaCode	
square/retrofit	awangdev/LintCode	happyfish100/fastdfs-client-java	
zeromq/jeromq	ronmamo/reflections	csxiaoyaojianxian/JavaScriptStudy	

alibaba/arthas	java8/Java8InAction	socketio/socket.io-client-java	
msgpack/msgpack-java	crossbario/autobahn-java	Beerkay/JavaMultiThreading	
agoncal/agoncal-book-javaee7	prometheus/client_java	XU-ZHOU/Java	
mthli/Java	objectbox/objectbox-java	shekhargulati/java8-the-missing-tutorial	
OpenFeign/feign	bytedeco/javacv	gaopu/Java	
dcm4che/dcm4che	rybalkinsd/atom	WritingMinds/ffmpeg-android-java	
javamelody/javamelody	influxdata/influxdb-java	binarywang/weixin-java-mp-demo	
testcontainers/testcontainers-java	tobato/FastDFS_Client	yangfuhai/ASimpleCache	

参考文献

[1]<!--TODO: software defect reduction top 10 list-->
[2]<!--TODO: Relating Identifier Naming Flaws and Code Quality: an empirical study-->
[3]<!--TODO: shorter identifier names take longer to comprehend-->
[4]<!--FIXME: コメントと可読性の研究を持ってくる-->

[5]<!--FIXME: 改行と可読性の研究を持ってくる-->

[6]<!-- TODO: Learning a metric of source code readability -->
[7]<!-- TODO: Refactoring: Improving the Design of Existing Code-->
[8]<!-- TODO: ソフトウェアメトリクスとデータ分析の基礎-->

[9]<!-- TODO: Concise and consistent naming-->
[10]<!--TODO: -CS百科事典からの引用-->

[11]<!--https://github.com/junit-team/junit4/blob/main/src/main/java/junit/framework/Assert.java(2021/1/4 16:51閲覧)-->

[12]<!-- TODO: Object oriented cobol recycling sneed -->
[13]<!-- TODO: java language specification se8-->
[14]<!-- TODO: GOF本-->

[15]<!-- TODO: 英語の命令分の主語に対する統語論及び、語用論上の制限-->

[16]<!--TODO:htps://opensource.com/resources/what-open-source 2021/1/1 19:14閲覧-->

[17]<!--TODO: Java parser visited-->
[18]<!--TODO:https://projects.eclipse.org/projects/eclipse.jdt(2021/1/6 16:00閲覧)-->

[19]<!--TODO: nltk 公式(https://www.nltk.org/) 2021/1/3 21:19閲覧-->

[20]<!--TODO: THE PENN TREEBANK: AN OVERVIEW -->
[21]<!-- FIXME: マルコフ連鎖の解説を持ってくる-->