PBL におけるソースコード引き継ぎ支援 ～日本語と英語を交えた識別子命名によるソースコード可読性向上の試み～

熊谷 優斗　伊藤 恵　奥野 拓

さまざまな情報系大学でシステム開発 PBL が行われている．PBL の中には複数年継続して行われ，プロジェクトメンバの入れ替わりが発生するものがある．本研究では，引き継ぎを行う PBL において，引き継いだ学生がソースコードを読むことに時間をかけてしまっている問題を解決することを目的とする．そのためのアプローチとして，日本語と英語を交えた識別子を用いることを提案し，その命名の支援を行うツールの開発を行う．本論文では，まず日本語のみを用いた識別子の可読性を評価するための予備実験を行ったため，その結果と考察を論じる．

System development PBL is performed at many information system university. Some PBLs continue for several years and their project members may be changed. In such PBLs, students who handover the project spend a lot of time to read source code in the project. The purpose of this study is to support students to handover the project. As approach of this purpose, we suggest using the identifier mixed Japanese with English, and develop a tool to support naming. In this paper, we did a preliminary experiment to evaluate the readability of the identifier only using Japanese, so we discuss results and consideration of it.

1　はじめに

1.1　背景

近年，実践的なソフトウェア開発演習として，PBL(Project Based Leaning) が注目され，多くの大学で導入されつつある．本学で行われている PBL の中には，複数年継続して行われ，プロジェクトのメンバーの入れ替わりが発生するもの（以下，継続的 PBL）がある．また，ソフトウェアの大規模化や，開発プロジェクトの複雑化により，ソフトウェアの保守作業にかかる時間的コストが増大している．更に，保守作業において最も時間のかかる作業は，既存のソースコードを読み解き，理解することであると言われている．継続的 PBL の場においても，保守作業は 1 つの問題として挙げられる．継続的 PBL に参加したことがある学生に対して，アンケート調査を行ったところ，14 名中 10 名が引き継ぎに苦労したと回答した．また，具体的にどのような点で苦労したか調査したところ，10 名の全員がソースコードを読み解くことであったと回答した．この結果から，継続的 PBL に参加している学生はコードリーディングに苦労していることが分かった．

ソースコードの可読性を高めるための手法として，識別子に日本語を用いることがある．一般に，日本人である我々にとって，英語よりも，母国語である日本語の方が，より表現豊かに扱えるはずである．すなわち，識別子の命名の際も，日本語を用いることによって，適切な命名を行うことができる．しかし，日本語のみを用いた識別子には，ソースコードエディタによる補完機能が効かないと言った問題や，getter や setter などで用いられるプリフィックスを適用できないと言った問題がある．

1.2　目的とアプローチ

本研究では，継続的 PBL において，前年度から引き継ぎを行う際に苦労している問題を解決することを目的とする．

Support to Handover Source Codes in PBLs -Try to Improve Readability of Source Code by The Identifier Mixed Japanese with English

Yuto Kumagai, 公立はこだて未来大学システム情報科学部, School of Systems Information Science, Future University Hakodate.

まずは，前年度書かれたソースコードを読み解く際に時間がかかっている課題を解決するため，適切な識別子の命名を行うことで，可読性の高いソースコードを書けるように支援することを目指す．そのための手段として，プログラミング言語によっては 2 バイト文字を用いた識別子の命名が可能である点に着目し，日本語を含めた識別子の適切な命名規則を模索する．また，日本語と英語を混ぜた識別子の命名によって、前項で述べた問題が解決されるかどうか検証を行う．その後，作成した命名規則に沿った識別子の命名付けを支援するツールの開発を行う．

1.3 先行研究

ソースコードの識別子に関する研究としては，平田ら [3] が日本語のみで書かれた識別子を用いることによる，可読性の変化を評価する研究を行った．この研究では，日本語識別子を用いることで，英語識別子よりも可読性が高まったことが証明された．

一方で，尾関ら [1] も日本語のみで書かれた識別子を用い，可読性の変化を評価したが，この研究では日本語識別子と英語識別子による可読性の変化を認めることはできなかった．これらの研究の違いを再確認する必要がある．

2 予備実験

平田らや尾関らが行った,「日本語識別子による可読性評価」の追試を行う．今回は学部 4 年生 4 名を対象に，用意したソースコードプログラムの難易度が適切であるか，評価手法は適切であるかといった観点を調査するための予備実験を行った．

2.1 題材とするソースコード

英語のみを用いた識別子（以下，英語識別子と呼ぶ）の代わりに日本語のみを用いた識別子（同様に日本語識別子と呼ぶ）を用いることで，ソースコードの可読性に変化が生まれるか比較して調査を行うため，同じ処理を行うソースコードに対し，英語識別子と日本語識別子を用いた 2 種類を用意した．例として、表 1 に英語識別子と日本語識別子の対応を示す．また，1 人の被験者に同じ処理を行うソースコードを連続で読ませてしまうと，後に読むソースコードの内容を理解した状態での実験となってしまうため，違う処理を行うソースコードをもう 1 つ用意した．それぞれ programA（英語識別子），プログラム A（日本語識別子），programB（英語識別子），プログラム B（日本語識別子）と呼ぶ．

ソースコードに関しては，Java を用いて作成した．Java は識別子に 2 バイト文字を扱うことをサポートしている．また，本学の学生（以下学生と呼ぶ）は学部によっては 2 年次の「情報処理演習 I」という講義で Java について学ぶことも考慮した．ソースコードを作成する際，情報処理演習 I の課題で用いるソースコードを参考にした．具体的には，programA は 2013 年度の課題で用いられるソースコード，programB は 2014 年度の課題で用いられるソースコードを参考にした．被験者によっては，今回実験で用いたソースコードの処理について多少の知識を事前に持っていた可能性があることを考慮に入れる必要がある．今回作成したソースコード内には，コメントを一切つけなかった．これは，ソースコードのみで可読性の評価を行うためである．また，それぞれのプログラムには，被験者がデバッグを簡単に行えるよう，テスト用のクラスを設けた．

付録 A に実験で用いた B のプログラムを示す．

2.2 実験手順

本実験は以下の流れで行った．

0. 予約語テストによる被験者のグループ分け
1. programA（プログラム A）のデバッグ実験
2. programA（プログラム A）についての確認テスト
3. プログラム B（programB）のデバッグ実験
4. プログラム B（programB）についての確認テスト
5. 実験後アンケート

各工程について次節以降に述べる．

2.3 被験者のグループ分けと予約語テスト

個人のプログラミングに対する習熟度や各ソースコードの難易度が実験結果に偏りを生んでしまうこ

表 1　B のプログラムにおける英語識別子と日本語識別子の対応

英語識別子	日本語識別子
frequencyMap	出現回数マップ
totalCount	総語数
Unigram	ユニグラム
initializeUnigram	ユニグラムを初期化
splitSentenceBySpace	センテンスを空白文字ごとに分割
addTotalCount	総語数を加算
countWordsFrequency	単語の出現回数を数える
isWordAppeardNotYet	単語がまだ出現していないか
registerNewWord	新しい単語を登録
countUpWordFrequency	単語の出現回数を加算
getTotalCount	総語数を取得
getCardinality	単語の種類を取得
getFrequency	出現回数を取得
add	加算
addTwoUnigramFrequency	二つのユニグラムの出現回数を加算
registerNewWord	新しい単語を登録
addTwoFrequency	二つの出現回数を加算
printWordTable	単語表を出力
createAllWordsSet	全ての単語の集合を作成
createSortedWordsArrayByWordsSet	単語集合からソート済みの単語配列を作成
createRowOfTable	行の文字列を作成

とを防ぐために，事前に被験者のプログラミング習熟度を図るテストを行い，その結果から被験者を 2 つのグループに分けた．グループの内訳については表 2 に示す．グループ分けは各グループの被験者のテストの平均点が近くなるように配慮しつつ行った．

また，プログラミング習熟度を図る方法として，尾関 [2] を参考に Java の予約語の知識を問うテストを行った．全部で 25 題出題し，それぞれの予約語に対する理解度を 1〜4 の点数で自己採点させた．採点の基準として，意味を十分に理解していれば 4 点，聞いたことすらなければ 1 点とした．

2.4　デバッグ実験

可読性の評価を行うための手法として，平田ら [3] が行った「デバッグ実験」を模倣した．これは，予めソースコードにバグを潜ませておき，被験者がバグを解消するまでの時間を計測するというものである．ソースコードの内部に潜ませるバグは，被験者がソースコードを読まずに，エディタが示すエラー箇所のみを見てデバッグを行ってしまうことを防ぐため，ソースコードのビルド時にエラーを吐かないようなバグを潜ませた．また，デバッグ自体にかかる時間をできるだけ少なくするために，バグを潜ませる箇所は 1 箇所のみにし，1 行程度の加筆もしくは修正を行うことでバグを解消できるようにした．これは，本実験の本

表 2　グループ分け

	プログラム	被験者	点数
グループ A	programA プログラム B	被験者 1	85
		被験者 2	64
グループ B	プログラム A programB	被験者 3	81
		被験者 4	64

来の目的はソースコードを読み解く際にかかる時間を計測するためであり，バグを解消するのに複雑なアルゴリズムを考える必要があるなどして時間をかけることは避けたかったためである．デバッグ実験では，全体の時間が長くなりすぎてしまうことを防ぐため，30 分が経過した時点で次の行程に移るようにした．

2.5 確認テスト

デバッグを行ってもらうだけではプログラムの理解度調査には効果が薄いのではないかと考え，各プログラムのデバッグ終了後に，それぞれのプログラムについての理解度を調査するために，簡単な確認テストを実施した．確認テストの内容は，問 1 として，それぞれのプログラムは何を処理するものであるか問うもの，問 2 として，各プログラムの入力を変更した場合，どのような出力が帰ってくるか問うものを用意した．

2.6 実験後アンケート

実験終了後に被験者に対してアンケートに答えてもらった．アンケートでは,「ソースコード全体を通して，読みやすいと感じたのは英語識別子か日本語識別子か」「変数や関数それぞれの役割を理解しやすいのは英語識別子か日本語識別子か」といった点や，実験を通して不便に感じた点などを答えてもらった．

3 結果と考察

3.1 実験結果

それぞれの被験者がデバッグ実験，確認テストに要した時間を表 3 に示す．デバッグ実験では，多くのプログラムにおいて経過時間が 30 分を超過し，途中で打ち切る結果となった．確認テストでは，どちらのテストに対しても，事前テストで高い得点を記録した被験者が短い時間で回答を終えた．また，被験者全体を通して，英語識別子と日本語識別子の差に関わらず，B のプログラムの方がより短い時間で回答を終えた．また，アンケートの結果を表 4 に示す．全体を通して英語識別子で書かれたソースコードと日本語識別子で書かれたソースコードのどちらが読みやすかった，という質問に対しては，4 名中 3 名が英語識別子であると回答した．また，変数や関数 1 つ 1 つの意味を理解しやすいのはどちらか，という質問に対しては，4 名中 2 名がどちらとも言えないと回答した．

図 1　実験の様子

表 3　デバッグ実験及び確認テストに要した時間

		デバッグ		テスト	
		A	B	A	B
グループ A	被験者 1	25	30	7	3
	被験者 2	30	27	10	3
グループ B	被験者 3	30	22	6	2
	被験者 4	30	30	10	13

表 4　アンケート結果

アンケート内容	英語識別子	日本語識別子	どちらとも言えない
それぞれの識別子のどちらで書かれたソースコードが読みやすいと感じたか	3	0	1
変数や関数それぞれの役割について理解しやすいのはどちらか	1	1	2

3.2 実験考察

デバッグ実験では多くの被験者が 30 分を超過する結果となってしまったが，B のプログラムにおいては被験者 2 と被験者 3 が時間内にデバッグを終了することができた．また，確認テストでは A よりも B の

プログラムの方が平均して短い時間で回答を終えている．更に，B のプログラムは，被験者全員が学部 2 年次に一度学んだことのあるアルゴリズムを用いている．これらのことから，今回実験で用いた 2 つのプログラムの間に難易度の差があったと考えられる．また，プログラム自体の難易度も不適切であったと考えられる．幅広い学年の学生を被験者にするために，より普遍的なアルゴリズムを用い，ソースコードの行数も少ないものを用いるべきである．1 つ目のアンケートでは，誰 1 人として日本語識別子を用いた方が全体を通して読みやすいとは答えなかった．これは，数年間プログラミングを経験したことにある人物にとって，日本語で書かれた識別子と英語で書かれた予約語が混在するソースコードに違和感を感じた結果であると考えられる．また，日本語識別子によって，変数や関数が持つプログラム上の意図は伝わりやすくなると仮定していたが，2 つ目のアンケートの結果により，必ずしもそうとは限らないことが明らかになった．また，実験後に被験者に対してインタビューを行ったところ,「プログラムを読んでいく途中で日本語が入ると，それがコメントもしくは出力文であると勘違いを起こす」という意見が得られた．プログラミング経験のある人物にとっての日本語識別子の問題点を発見することができた．

4 おわりに

本稿では，日本語識別子による可読性実験の予備実験の概要と考察を論じた．予備実験によって，用意したプログラムの問題点，更に，日本語識別子の問題点を明らかにすることができた．今後は，本実験に向けて，プログラムの改善を行っていく．また，日本語識別子を用いることでは継続的 PBL の引き継ぎにおける苦労の緩和にはならないと判断し，別のアプローチを探ることを考える必要もある．

参 考 文 献

[1] 尾関哲, 佐藤邦弘, 太田健一, 宮脇冨士夫: プログラム理解における日本語使用の効果, 情報処理学会全国大会講演論文集, Vol. 51, No. ソフトウェア工学 (1995), pp. 169–170.

[2] 尾関哲, 佐藤邦弘, 太田健一, 宮脇冨士夫: プログラム理解における日本語使用の効果 (2), 情報処理学会全国大会講演論文集, Vol. 53, No. インタフェースサイエンス (1996), pp. 139–140.

[3] 平田篤志, 早川栄一, 並木美太郎, 高橋延匡: 識別子と内部コード系に着目した日本語によるプログラムの可読性の一評価, 情報処理学会研究報告ソフトウェア工学（*SE*）, Vol. 1995, No. 55(1995), pp. 1–8.

A 付録: 実験に用いたソースコード

```
package programB;

import java.util.*;

/**
 * @author b1013130 熊谷優斗
 */

public class Unigram {

  private HashMap<String, Integer> frequencyMap;
  private int totalCount;


  public Unigram() {
    frequencyMap = new HashMap<String, Integer>();
  }

  public Unigram(ArrayList<String> text){
    frequencyMap = new HashMap<String, Integer>();
    totalCount = 0;

    for(int i=0; i<text.size(); i++){
      initializeUnigram(text.get(i));
    }
  }

  private void initializeUnigram(String sentence) {
    String[] words = splitSentenceBySpace(sentence);

    addTotalCount(words);
    countWordsFrequency(words);
  }

  private String[] splitSentenceBySpace(String sentence){
    String[] splitedSentence = sentence.split(" ");
    return splitedSentence;
  }

  private void addTotalCount(String[] words){
    totalCount = totalCount + words.length;
  }

  private void countWordsFrequency(String[] words){
    for(int i=0; i<words.length; i++){
      if(isWordAppeardNotYet(words[i])){
        registerNewWord(words[i]);
      }else{
        countUpWordFrequency(words[i]);
      }
    }
  }

  private boolean isWordAppeardNotYet(String searchWord){
```

```
  if (getFrequency(searchWord) == 0){
    return true;
  }else{
    return false;
  }
}

private void registerNewWord(String word){
  frequencyMap.put(word, 1);
}

private void countUpWordFrequency(String word){
  int frequencySoFar = frequencyMap.get(word);
  frequencyMap.put(word, frequencySoFar+1);
}


public int getTotalCount() {

  return totalCount;
}


public int getCardinality() {

  return frequencyMap.keySet().size();
}


public int getFrequency(String searchWord) {
  if (frequencyMap.containsKey(searchWord)) {
    return frequencyMap.get(searchWord);
  }else{
    return 0;
  }
}

public static Unigram add(Unigram u1, Unigram u2) {
  Unigram newUnigram = new Unigram();

  newUnigram.addTwoUnigramFrequency(u1, newUnigram);
  newUnigram.addTwoUnigramFrequency(u2, newUnigram);

  newUnigram.totalCount = u1.totalCount + u2.totalCount;

  return newUnigram;
}

private void addTwoUnigramFrequency(Unigram oldUnigram, Unigram newUnigram){
  for (Map.Entry<String, Integer> Item : oldUnigram.frequencyMap.entrySet()) {
    String word = Item.getKey();
    int frequency = Item.getValue();
    if(newUnigram.isWordAppeardNotYet(word)){
      newUnigram.registerNewWord(word, frequency);
    }else{
      newUnigram.addTwoFrequency(word, frequency, oldUnigram.frequencyMap.get(
```

```
            word));
      }
    }
  }

  private void registerNewWord(String word, int frequency){
    frequencyMap.put(word, frequency);
  }

  private void addTwoFrequency(String word, int frequency1, int frequency2) {
    frequencyMap.put(word, frequency1+frequency2);
  }


  public static void printWordTable(ArrayList<Unigram> unigramAry) {
    HashSet<String> wordsSet = Unigram.createAllWordsSet(unigramAry);

    String[] sortedWordsArray = Unigram.createSortedWordsArrayByWordsSet(wordsSet);

    String row = "";
    for (String word : sortedWordsArray){
      row = Unigram.createRowOfTable(unigramAry,word);
      System.out.println(row);
    }
  }

  public static HashSet<String> createAllWordsSet(ArrayList<Unigram> UnigramAry){
    HashSet<String> wordsSet = new HashSet<String>();
    for(int i=0; i<UnigramAry.size(); i++){
      for (Map.Entry<String, Integer > Item : UnigramAry.get(i).frequencyMap.
          entrySet()) {
        String word = Item.getKey();
        wordsSet.add(word);
      }
    }
    return wordsSet;
  }

  public static String[] createSortedWordsArrayByWordsSet(HashSet<String> wordsSet
      ){
    String[] wordsArray = wordsSet.toArray(new String[0]);
    Arrays.sort(wordsArray);
    return wordsArray;
  }

  public static String createRowOfTable(ArrayList<Unigram> unigramAry, String word
      ){
      String row = word;
      for(int i=0; i<unigramAry.size(); i++){
        row = row  + "\t" + unigramAry.get(i).getFrequency(word);
      }
      return row;
    }
}
```

リスト 1　programB

```
package プログラムB;

import java.util.*;

/**
 * @author b1013130 熊谷優斗
 */

public class ユニグラム {

  private HashMap<String, Integer> 出現回数マップ;
  private int 総語数;


  public ユニグラム() {
    出現回数マップ = new HashMap<String, Integer>();
  }

  public ユニグラム(ArrayList<String> テキスト){
    出現回数マップ = new HashMap<String, Integer>();
    総語数 = 0;

    for(int i=0; i<テキスト.size(); i++){
      ユニグラムを初期化(テキスト.get(i));
    }
  }

  private void ユニグラムを初期化(String センテンス) {
    String[] 単語一覧 = センテンスを空白文字ごとに分割(センテンス);

    総語数を加算(単語一覧);
    単語の出現回数を数える(単語一覧);
  }

  private String[] センテンスを空白文字ごとに分割(String センテンス){
    String[] 分割済みセンテンス = センテンス.split(" ");
    return 分割済みセンテンス;
  }

  private void 総語数を加算(String[] 単語一覧){
    総語数 = 総語数 + 単語一覧.length;
  }

  private void 単語の出現回数を数える(String[] 単語一覧){
    for(int i=0; i<単語一覧.length; i++){
      if(単語がまだ出現していないか(単語一覧[i])){
        新しい単語を登録(単語一覧[i]);
      }else{
        単語の出現回数を加算(単語一覧[i]);
      }
    }
  }

  private boolean 単語がまだ出現していないか(String 調査単語){
    if (出現回数を取得(調査単語) == 0){
      return true;
```

```
  }else{
    return false;
  }
}

private void 新しい単語を登録(String 単語){
  出現回数マップ.put(単語, 1);
}

private void 単語の出現回数を加算(String 単語){
  int これまでの出現回数 = 出現回数マップ.get(単語);
  出現回数マップ.put(単語, これまでの出現回数+1);
}

public int 総語数を取得() {
  return 総語数;
}

public int 単語の種類を取得() {
  return 出現回数マップ.keySet().size();
}

public int 出現回数を取得(String 調査単語) {
  if (出現回数マップ.containsKey(調査単語)) {
    return 出現回数マップ.get(調査単語);
  }else{
    return 0;
  }
}

public static ユニグラム 加算(ユニグラム ユニグラム1, ユニグラム ユニグラム2) {
  ユニグラム 新しいユニグラム = new ユニグラム();

  新しいユニグラム.二つのユニグラムの出現回数を加算(ユニグラム1, 新しいユニグラム);
  新しいユニグラム.二つのユニグラムの出現回数を加算(ユニグラム2, 新しいユニグラム);

  新しいユニグラム.総語数 = ユニグラム1.総語数 + ユニグラム2.総語数;

  return 新しいユニグラム;
}

private void 二つのユニグラムの出現回数を加算(ユニグラム 古いユニグラム, ユニグラム 新しいユニグラム){
  for (Map.Entry<String, Integer> Item : 古いユニグラム.出現回数マップ.entrySet()) {
    String 単語 = Item.getKey();
    int 出現回数 = Item.getValue();
    if(新しいユニグラム.単語がまだ出現していないか(単語)){
      新しいユニグラム.新しい単語を登録(単語, 出現回数);
    }else{
      新しいユニグラム.二つの出現回数を加算(単語, 出現回数, 古いユニグラム.出現回数マップ.get(単語));
    }
  }
```

```
  }

  private void 新しい単語を登録(String 単語, int 出現回数){
    出現回数マップ.put(単語, 出現回数);
  }

  private void 二つの出現回数を加算(String 単語, int 出現回数1, int 出現回数2) {
    出現回数マップ.put(単語, 出現回数1+出現回数2);
  }


  public static void 単語表を出力(ArrayList<ユニグラム> ユニグラム配列) {
    HashSet<String> 単語集合 = ユニグラム.全ての単語の集合を作成(ユニグラム配列);

    String[] ソート済みの単語配列 = ユニグラム.単語集合からソート済みの単語配列を作
        成(単語集合);

    String 行 = "";
    for (String 単語 : ソート済みの単語配列){
      行 = ユニグラム.行の文字列を作成(ユニグラム配列,単語);
      System.out.println(行);
    }
  }

  public static HashSet<String> 全ての単語の集合を作成(ArrayList<ユニグラム> ユニグ
      ラム配列){
    HashSet<String> 単語集合 = new HashSet<String>();
    for(int i=0; i<ユニグラム配列.size(); i++){
      for (Map.Entry<String, Integer > Item : ユニグラム配列.get(i).出現回数マップ.
          entrySet()) {
        String 単語 = Item.getKey();
        単語集合.add(単語);
      }
    }
    return 単語集合;
  }

  public static String[] 単語集合からソート済みの単語配列を作成(HashSet<String> 単
      語集合){
    String[] 単語配列 = 単語集合.toArray(new String[0]);
    Arrays.sort(単語配列);
    return 単語配列;
  }

  public static String 行の文字列を作成(ArrayList<ユニグラム> ユニグラム配列,
      String 単語){
      String 行 = 単語;
      for(int i=0; i<ユニグラム配列.size(); i++){
        行 = 行 + "\t" + ユニグラム配列.get(i).出現回数を取得(単語);
      }
      return 行;
    }
}
```

リスト 2　プログラム B

```
package programB;

import java.util.ArrayList;

public class TestUnigram {

  private static ArrayList<String> generateArrayText1() {
    ArrayList<String> text = new ArrayList<String>();
    text.add("the other day , I met a bear ,");
    text.add("a great big bear , a way up there . ");
    text.add("he looked at me , I looked at him ,");
    text.add("he sized up me , I sized up him .");
    return text;
  }

  private static ArrayList<String> generateArrayText2() {
    ArrayList<String> text = new ArrayList<String>();
    text.add("he says to me , \" why don't you run ? \"");
    text.add("\" cause I can see , you have no gun . \"");
    text.add("I say to him , \" that's a good idea . \"");
    text.add("\" now let's get going , get me out of here ! \"");
    return text;
  }

  public static void main(String[] args) {
    ArrayList<String> text1 = generateArrayText1();
    ArrayList<String> text2 = generateArrayText2();

    Unigram d1 = new Unigram(text1);
    Unigram d2 = new Unigram(text2);
    Unigram d3 = Unigram.add(d1, d2);

    System.out.println("#Frequency of each words");

    ArrayList<Unigram> list = new ArrayList<Unigram>();
    list.add(d1);
    list.add(d2);
    list.add(d3);
    System.out.println("word" + "\t" + "text1" + "\t" + "text2" + "\t" + "text1+
        text2");
    Unigram.printWordTable(list);
  }
}
```

リスト 3　B のプログラムのテストクラス