人にやさしいプログラミングの哲学　講義要綱

CreW Project
http://www.crew.sfc.keio.ac.jp/

2003 年 5 月 20 日

目次

第 I 部

構造化プログラミング編

第 1 章

人にやさしいプログラムの書法

この章で学習すること

人にやさしいプログラムの書法でプログラムが書ける
- プログラムにインデントが付けられる
- プログラムに適切な変数名が付けられる
- プログラムに適切なコメントが付けられる

HCP チャートを使ってプログラムの構造を設計できる

ユーザが使うことを考慮したプログラムが書ける
- ユーザインターフェイスの基本構造を説明できる
- ユーザの不正な入力を考慮したプログラムが書ける

時間めやす

表 1.1: 時間めやす

1.1 節	10 分
1.2 節	20 分
1.3 節	20 分
1.4 節	10 分

1.1　人にやさしいソースコードとは

学習目標

このテキストの方針を理解する

考えてみよう

出題意図: 今まで書いたプログラムと、本テキストでこれから書いていくプログラムの違いを理解する。

コメントを書く
プロンプトを表示する
本処理を繰り返し、ユーザの入力値によって停止する
出力を見やすくする

詳しくは、竹田論文 (p.946-947) を参考にすると良い

セリフ: 「これから演習でプログラムを書いてもらうが、この（人にやさしい）形式でないと何度でもやり直しをさせるので注意すること」

1.2　人にやさしいソースコードの書法

学習目標

プログラムにインデントが付けられる
プログラムに適切な変数名が付けられる
プログラムに適切なコメントが付けられる

オマジナイ

”オマジナイ”を確認する。

package—, import—　本テキストのソースコードは、テキスト作成上の都合 (package 管理しているため) 全て package, import 文が記述されています。
Input クラスを配布して、作成するプログラムと同じフォルダに置くようにして下さい。

public static void main(String args[]), main()　この main メソッドの扱いなどはこの講座独自のルールであることを一応断っておくといいと思います。またメソッドという言葉が頻出していますが、第 4 章以降で詳しく説明するのでここではメソッドについて深く触れる必要はありません。
メソッド登場時に、static にする必要をなくすためです。

文字の出力

Java 言語で文字列をコンソールに出力するには、System.out.println() メソッドを使います。(() の中に表示したい文字列を、ダブルクォーテーション「”」で囲んで記述)

print(), ush()

print() は改行しない。(本当は、フラッシュもされないのが仕様だが、Windows 版 Java では何故かフラッシュされる) 同時に、ush()　も説明してください。

文字列

”文字列は、何文字かの文字をひとまとめにしたもので、プログラムでは意味のある文やフレーズをまとめて扱うことができるので便利。

表 1.2: 文字列の例

”abcdefghij”	・・・長さ 10 文字の文字列
””	・・・長さ 0 文字の文字列
”あ”	・・・長さ 1 文字の文字列

1.2.1 意味のまとまりを意識する

インデントを入れることで同じように並んでいる文と文の間には上下関係があることを気づかせます。またインデントによってソースコードに階層構造ができ、見渡しやすくなるというメリットを気づかせます。たとえば本の目次の例を挙げて、「本を手渡されてすぐにそれにどんなことが書いてあるかを知りたいときあなたはまずどこを読みますか」と問いかけ、本の目次がやっている内容の階層化と同じことがプログラムではインデントによって行われていることを説明します。

二つ目の改善点として、処理のまとまりごとに空行を入れブロックを作ることで、一つのプログラムの中にいくつかの意味のまとまりを作って読みやすくすることを紹介します。

1.2.2 変数に適切な名前をつける

この節ではプログラムを分かりやすくするために、変数やクラスに意味のある分かりやすい名前をつけることが重要であることを意識してもらいます。

例題プログラムのような変数名からそれが何を表すものなのか考えさせ、インデントや

パラグラフがあってもまだ「分かりにくい」ということをここでも痛感してもらいます。

どういう名前なら分かりやすいのか、またアルファベット一文字の変数名は良いのか悪いのかといった問題提起をし、「意味が分かりやすいかどうか」という基準で考えて名前をつけることが大事であると意識してもらいます。

変数名がついたところで最初に見せたプログラムが実は成績の平均点を四捨五入するプログラムであったことに気づいてもらいます。

1.2.3　コメントをつける

考えてみよう

出題意図: 「コメントには目的をつける」という話題の前ふり

質問

どんな成績管理を行うのか
合計を 5 で割るとはどういうこと?
average を 10 倍する..... とはどういうこと?

1.2.4　ひとにやさしいコメントの書法

プログラムの目的を明らかにするコメント

明らかにして欲しいこと

手段から、目的を推測するのは難しい
目的が分かれば、手段を読むのが容易である

コメントの種類

各コメントの「通用する」範囲を明らかにしてあげてください。

```
/**
 * Hoge クラス
 */

public class Hoge{
}
```

特に、初心者は、上記のようなプログラムを書きがちなので、注意。(コメントと通用する部分が 1 行あいていては通用範囲が分かりにくい)

1.3 プログラムの目的と階層構造

学習目標

プログラムが、目的の階層構造になっていることを理解する
HCP チャートを書く目的を説明できる
HCP チャートを反映したソースコードについて考える

1.3.1 目的の階層構造

挙げられた目的の階層構造は、前のソースコードと対応しています。

1.3.2 HCP チャートによるプログラムの設計

本テキストの第 I 部で行われるすべての演習課題（ミニプロ含む）は、HCP チャートを書いてから、プログラムを書くという方針ですすめる。

HCP チャートの左側に行くほど、大きな目的になり、右側に行くほど、手段に近くなる。(あまり手段すぎるものは書かなくて良い)

HCP チャートの左側のほうが抽象的、右側のほうが具体的。

HCP チャートによる設計において一番重要なのは、プログラムの目的をしっかり捕らえ、明確な日本語として記述して、それを階層構造に整理することです。記法は多少適当でかまいませんので、日本語とその構造をしっかり記述してください。

1.3.3 HCP チャートを反映したソースコードの記述

よい、HCP チャートが書ければ、よいコメントが書ける。

ただし、HCP チャートがそのままソースコードに反映できない場合もあるので注意。

1.4 人にやさしいユーザインターフェイス

学習目標

UI の基本 (入力=> 処理=> 表示) を理解し、プログラムが書ける
繰り返しを使う意味を理解できる
不正入力というものについての知識を得る

1.4.1　ユーザインターフェイスの基本

本テキストでは、UI を備えたプログラムを「-Application」というクラス名に、それ以外のプログラムを「-Sample」にしてあります。(説明しても良いかもしれません。)

「タイトル」、「プロンプト」、以外に「エンドタイトル」(終了しました) も重要です。(それがないといつ終ったか分からない)

1.4.2　繰り返し

while 文

ここでででている HCP チャートは while 文のものではなく、「繰り返し」のものだということに注意。(for 文も繰り返しである)

繰り返しの終了記号 (下向き矢印) は、繰り返しの一段抜け記号です。

1.4.3　不正な入力の処理

不正な入力の処理を直す問題ではない。(すでに直されている)

直されている部分は、48-52 行目

40 行目のメッセージが工夫されていることも触れる。

練習問題 3 との関連で、不正な入力の処理を行わなかった場合どういうエラーが出るか (ArithmeticException : devide by zero) がでる

1.5　練習問題

練習問題 1

タイトルコメントとして必要なことの確認問題です。

練習問題 2

「人にやさしいプログラム」の復習問題です。

練習問題 3

直すのが主旨ではありません。あくまで、エラーを体験するのが目標です。

参考までに、Input. の実装を載せておきます。

```
private static InputStream stream = System.in;
```

```
/**
 * コンソールから文字を読み込む
 * @return String
 */
public static String getString() {

  String returnString = null;

  try {

    InputStreamReader isr = new InputStreamReader(stream);
    BufferedReader br = new BufferedReader(isr);
    returnString = br.readLine();
    return returnString;

  } catch (IOException e) {
    e.printStackTrace();
    return null;
  }
}

/**
 * コンソールから数字を読み込む
 * @return int
 */
public static int getInt() {

  int returnInt = 0;

  returnInt = Integer.parseInt(getString());
  return returnInt;

}
```

Input クラスには他にも、与えられた変数が文字であるか数字であるかを判別し、数字であれば true、そうでなければ false を戻り値とする、isInteger メソッドが実装されています。これは教材の例題プログラムには登場しませんが、ミニプロの際に配布する住所管理プログラムで使用されていますのでその際に説明をしてください。

なおミニプロ以前であっても、1.4.3 節の不正な入力の処理や第 3 章のエラー処理などで受講者からこのようなメソッドが欲しいという要望が出た場合は個別に対応してください。

第 2 章

変数としての抽象化 (1)

この章で学習すること

数字に意味を与え、変数（定数）として定義できる
変数と値の関係が説明できる
変数の宣言・代入・評価の過程を変数表（表モデル）を描いて説明できる
式が与えられたとき、どのように値が導かれるかを評価の概念を用いて説明できる

時間めやす

表 2.1: 時間めやす

2.1 節	40 分
2.2 節	40 分
2.3 節	20 分
2.4 節	20 分

2.1　変数と値

学習目標

プログラムにおけるデータをどう表現するかについて考えることができる
変数と値の関係を理解する
変数の宣言・代入・評価の過程を変数表（表モデル）を描いて説明できる
変数が評価される仕組みを説明できる

2.1.1　データをどう扱うか

データ構造の設計はアルゴリズムの設計と並び、初学者にとって難しいものです。

特に、この章の冒頭で議論になっている、じゃんけんの手のデータ表現法は、現実世界のデータをプログラムのデータとして表現する、いわば「モデル化」の基礎となっています。

このじゃんけんのモデル化はあたりまえですが、初学者には難しいものです。実際に、練習問題 2 のような問題は、じゃんけんをやった学習者は、転移することで解けますが、じゃんけんをやらない学習者は解くことが出来ません。

2.1.2　変数と値

抽象的なものが変数、具体的な内容が値です。これから、プログラムを読む時に、変数を評価すると値になるという置き換えの思考が身につくことがこのセクションの目標でもあり、この章全体の目標でもあります。

（この章だけ）表モデルは、親切に「変数名」、「値」のキャプションがついている。

変数表を使ったプログラムをトレースする方法を学びます。これから先、常に使う方法なので、ここでしっかり身に付けておくことが必要になります。パイロットテストでは、慣れない方法なので、「表モデルで説明して」というと戸惑うことが多く見られました。今後もメソッドの話、スコープの話をする際にも必要になるので、なるべく丁寧に講義をしてください。簡単なプログラムであれば、表モデルを自力で書けるようになるのが目標です。

この先、ＴＡ（サブ講師）は、学習者がプログラムの理解に困っているようであれば、表モデルを書かせて説明させると良いでしょう。トレース表を書かせるのも手です。

2.1.3　プログラムの実行と変数の評価

変数の宣言という概念を学びます。表モデルでは、宣言が行われるとどのように表現するかということと、常に対応をとって説明してください。宣言が行われたら表モデルにどう記述したらよいのかが分かることが目標です。

次に代入を学びます。代入は直接値が代入される場合だけではなく、他の変数が変数に代入される場合は、いったん評価を行って、値にしてから代入されるということを学びます。この置き換えモデルが今後も基礎になります。この代入の過程も表モデルではどのように表現するかという点と同時に講義をしてください。

「プログラムの実行と表の変化」では、プログラムの実行とともに、表が変化していく様を提示します。この一つ一つをプログラムの「状態」といいます。プログラムは、状態が次々と変化していくことにより仕事を成し遂げるという話をすると良いでしょう。

2.2 式と評価

学習目標

式から値を導く過程を評価の概念を用いて説明できる
条件分岐を使ったプログラムが書けるようになる
条件式の評価の仕組みを説明できる

2.2.1 式と値

評価の概念を変数だけではなく、式に拡大します。プログラムがすべて評価という概念を適用することで説明できるということを実感してもらえれば成功です。ここでは、難しそうなプログラムでも自分の力で読めるようになりそうだと感じられることが目標です。

今後メソッドにも評価を適用するので、「ここを評価すると？」などの質問を順を追って受講者に行うのが有効なようです。

2.2.2 条件式の評価

プログラムで条件分岐を使えるようになることが基本ですが、今まで習ってきた評価という概念を用いて、説明することを忘れないように。「条件文も式だから、評価される。その値は true か false。」

一番最後の、「2.2.2.3 条件式の評価」から説明したほうが、おそらく説明しやすい。

HCP チャートは、条件分岐のもの。Java では、if 文のほかに switch 文が使える。

2.3 変数の型

学習目標

変数には型があることを理解する
型の変換を用いたプログラムが書けるようになる

2.3.1 変数の型

前節で boolean 型をやる前にやってもよい。(型変換とまとめたいのでテキストではこの位置)

なぜ型が必要かという議論は、3 章で詳しく議論する内容なので、今回は「型があれば値になにが入るか分かりやすい」という程度の結論にとどめてください。

2.3.2　型の変換

テキストの趣旨から言いうとあまり重要ではないが、テクニックとして必要。場合によっては、他書籍の詳しい説明、問題を解かせると良い。

例題は break の説明をしっかりやってください。

あと、タイトルコメントに表示例が書かれています。議論すると表示例があるほうが分かりやすいということになるだろうから、表示例が書けるときはこのように書くように指導する。

2.4　プログラムの意味と変数

学習目標

定数を用いる目的を説明できる
定数と変数の違いについて説明できる

2.4.1　定数

プログラムの意味を明らかにするためには、定数を利用する。プログラムの該当部分を説明する。

特に数字は、これからインラインコーディングしないように（3 章の最初の例でも説明できる。）

表示テキスト (”アプリが始まりました”) なども、定数にできるという話をしても良い。(ただし、やってみると分かるが、あまり表示テキストを定数化すると読みにくくなる) じゃんけんで言えば、グー、チョキ、パーなどのテキストは定数化すると嬉しいという議論ができる。(英語バージョンはどうする？などの理由で。)

2.5　練習問題

練習問題 1

表モデル、評価、演算子が理解できているかどうかの確認問題です。

練習問題 2

現実問題のモデル化の問題です。じゃんけんアプリケーションを参考に、「転移」させると出来ます。

練習問題 3*

変数の使用、ユーザインターフェース、定数というこの章までで学んできたことをすべて確認する問題です。

ちなみに、問題に*がついているのは、発展問題です。

第 3 章

変数としての抽象化 (2)

この章で学習すること

配列を使ったプログラムが書ける
情報を管理する簡単なアルゴリズムを考えて、実装できる
静的エラーと動的エラーの違いを説明できる

時間めやす

表 3.1: 時間めやす

3.1 節	30 分
3.2 節	50 分
3.3 節	40 分

3.1　同種の変数をまとめて扱う

学習目標

配列と値の関係を配列の表モデルを用いて説明できる
配列を用いる利点について説明できる

3.1.1　配列

2 章で値と変数の間に具体と抽象という関係を当てはめたのを発展させ、変数を寄せ集めて抽象化したものとして配列を位置づけます。配列の扱い方と、たくさんの変数をまとめて扱うことを考え、配列を導入することによるメリットを理解することが本節の目標です。

例題では、SCORE_SIZE 定数の導入のメリットについて議論すると復習になります。

3.1.2　配列の表モデル

変数の評価と同様、配列も評価によって値になることという置き換えの思考が身につくことがこの節の目標です。前節で登場した配列の表モデルを用い、自分で評価の仕組みが説明できるようになることが目標です。

3.1.3　配列を使ったプログラムの利点

配列の扱いに伴って出てくる繰り返し処理として for 文を学び、while 文の復習もあわせて行います。変数をトレースし、2 通りの繰り返し処理の違いを比較します。

for 文で出てくるイテレーターの i は、意味のわからない変数名ですが、i だけは、プログラマ共通の意味として、イテレータの略記号として使えるといってしまってよいでしょう。

3.2　配列を利用したアルゴリズム

学習目標

配列の要素の追加のアルゴリズムを HCP チャートを用いて考え、実装できる
配列の要素の検索のアルゴリズムを HCP チャートを用いて考え、実装できる
配列の要素の削除のアルゴリズムを HCP チャートを用いて考え、実装できる

配列を用いたアプリケーションを通して、要素の追加・検索・削除のアルゴリズムを理解することがこの節の目標です。

3.2.1　コマンド入力を受け付けるアプリケーション

状態遷移図の説明を少し入れても構いません。ここでは、このような図を書くと、プログラムの様子が容易に分かるという議論をして下さい。

この例題から、名前と成績の対応ができるようになります。

3.2.2　配列への要素の追加

もし、アルゴリズムの説明がすんなり行くようならば、HCP チャートとソースコードの関係に注目してください。「データを入力する」という HCP チャートの階層は、ソースコードに反映されていません。これは、意味的な階層と、ソースコードのブロックが必ずしも対応しないことを意味しています。本テキストの HCP は、初学者にわかりやすいように、かなりソースコードを意識したものになっていますが、本来 HCP チャートは、実装に依存しない、構造が書かれるべきなのです。

3.2.3 配列にある要素の検索

検索と削除のリストは、この先の 3.2.5 成績管理アプリケーションから抜粋しているので、注意。

この項では、配列の要素の検索の手順を理解することが目標です。検索はどのように行うのかまず HCP チャートで必要な処理を階層化してみます。それに従い、検索して見つかるとはどういうことなのか、見つからないとはどういうことなのかを考えながら理解していきます。

3.2.4 配列からの要素の削除

配列の要素の削除の手順を理解することが目標です。削除するためには削除すべき対象がどこにあるのかまず検索が必要になることや要素を削除した後、それ以後の要素を詰める必要があることなどを説明すると良いでしょう。

さらに、前節の HCP チャートと比較し、目的が同じだから前節の検索のアルゴリズムが再利用できることを説明します。

ちなみに、メソッドを使えば、本当にコードの再利用が可能ですが、検索アルゴリズムが index を返す必要があります。(この話は、余裕があったら 5 章で)

考えてみよう

出題意図: 配列の仕組みが分かっているかどうか

実行させてみると良いと思います。ご想像のとおり、最後の人のお化けがでます。

表モデルを書いて、実行例を考えると分かります。

デバッグプリントの項で、移動している様をプリントするような例が出ていますので、あわせて使うとよい。

3.2.5 成績管理アプリケーション

ソースのみです。

3.3 静的エラーと動的エラー

学習目標

動的エラーと静的エラーの違いを説明できる
デバッグプリントを用いてエラーを解決できるようになる
コンパイルエラーの利点を説明できる

3.3.1　二種類のエラー

ここで今まで経験してきたエラーには静的エラーと動的エラーの二種類があり、両者の違いを理解してもらいます。どちらが対処しやすいものであるかどうかを議論し、2 つのエラーの違いの確認も行います。

強調すべき所は、

> 良いプログラマは、エラーを出さないのではなく、エラーが起きたときに解決できる
> エラーが起きたときに解決できるようなソースコードを書く

ということです。

3.3.2　バグを解決するために

プログラミングを行っていくうえでのバグに対する基本的な対処方法を自分なりに身につけてもらうことがこの節の目標です。

配列の ArrayIndexOutOfBoundsException を例に、以前に経験している NumberFormatException などとあわせ、これらは実行時に起こるエラー動的エラーであったことを確認します。

デバッグの方法について、デバッグプリントという方法を学びます。

3.3.3　静的エラーとコンパイラの役割

> コンパイルエラーはうれしい

ということが分かれば成功です。

あわせて、

> コンパイルは少しずつ通す

ということに触れましょう。(初学者は、えてして、一気にコンパイルを通すので、大量のコンパイルエラーに悩まされることになります。)

考えてみよう

出題意図: 2 つのエラーの区別ができるかの確認、コンパイルエラーのうれしさ

動的エラーは、見つけるまでも大変だという議論ができるとよい。

さらにいえば、例外もうれしい。(例外がでない動的エラーは最悪です)

考えてみよう

出題意図: 型チェックのうれしさを知る

変数に意図しない値が絶対に入らないことを保証できるのでうれしいという議論で十分です。(前の考えてみようを利用してください)

ちなみに、ここでいっているのは、「変数の型」のことであり、いわゆるデータタイプの型ではありません。例えば、変数に型がない smalltalk などでは、ポリモーフィズムを促進できるというメリットがありますが、smalltalk にも型はあるので。念のため。

3.4 練習問題

練習問題 1

バグの発見、解決の練習問題です。

登録人数が 0 人の時に、一覧を実行すると、おかしいメッセージが出力されます。

練習問題 2

配列を使ったプログラミングの確認問題です。

じゃんけん同様、成績管理アプリケーションを参考にして、「転移」させるとできます。

早く出来てしまった人に対しては、2 重投稿が出来ないようにするなどのアドバイスを。

第 4 章

手続きとしての抽象化 (1)

この章で学習すること

手続きを使ったプログラムを記述できる
手続きを使ったプログラムの実行の過程を説明できる
変数の有効範囲を説明できる
引数を用いて手続きを抽象化できる
実引数と仮引数の結合の様子を図を用いて説明できる

時間めやす

表 4.1: 時間めやす

4.1.1 節	10 分
4.1.2 節	10 分
4.1.3 節	10 分
4.1.4 節	20 分
4.2.1 節	20 分
4.2.2 節	40 分

4.1　手続き

学習目標

引数なしのメソッドを用いてプログラムを書き直すことができる
メソッドの呼び出しと実行の関係を変数の表モデルを用いて説明できる
変数のスコープについて変数の表モデルを用いて説明できる

4.1.1　メソッド

メソッドの概念を導入します。メソッドの概念は、すべて足し算アプリケーションを例に説明し、「引数無し」=>「引数あり」=>「戻り値あり」の順序で段階的に説明します。この章では、そのうち、引数ありまでを学習します。

まず、はじめに、引数無しメソッドから導入します。共通の処理という分かりやすいところから導入すると受講者も抵抗がないようです。

後半では重複コードだけではなく、プログラムの意味に注目しようというふうにもっていくので、この時点で重複コードだけを強調しすぎるのは避けてください。

4.1.2　メソッドの書法

書法はさらっと説明してください。

4.1.3　メソッドと HCP チャート

手続きは、二重丸で表します。

4.1.4　変数のスコープ

メソッドにも表モデルを導入する部分です。メソッドの呼び出しによって、新たな表が作られるという点がポイントです。受講者はメソッドに慣れていないので、なるべく丁寧にサンプルのソースを読む時間を与えてください。

変数のスコープは、1 つの表の中であるという理解で説明を行います。パイロットテストを行った結果、授業中は分かった気になっていても、やはり自分でプログラムを書いていると混乱してくることが多いようです。また変数名にも大きく影響され、同じ変数名を使ったりするややこしいものだと理解の甘さが目立ちます。

変数のスコープはかなりのつまずきポイントですので、丁寧に説明し、混乱があれば必ず表モデルで説明を貫き通すことを行ってください。

表モデルでの理解でも十分ですが、実はこの説明はうそで、実際にはプログラムのブロック内有効ですので、講師の方は押さえて置いて、よく出来る学習者にだけ伝えてください。

4.2　引数による手続きの抽象化

学習目標

引数を用いて汎用的なメソッドを書くことができる
実引数と仮引数の対応を理解する

引数を持つメソッドの実行の様子を表モデルを用いて説明できる

4.2.1 汎用的な手続き

このセクションでは抽象化という概念と絡めて、メソッドを説明します。引数を導入することで、より手続きが抽象化できるという話にもって行きます。

引数ありの場合、仮引数と実引数が問題になりますが、実引数が値、仮引数が変数という今まで学んできた概念で説明すれば、特に問題はないようです。

この章ではあまりつっこんだ議論はできないので、どうしても聞いているだけになりがちです。自分で何をメソッドにしたら良いかということを 5 章で議論しますが、その辺までは、どうしても知識先行になってしまいがちで、受講者の反応も薄いものになってしまいます。基本的には書法や表モデルとの絡みがこれからの基礎知識として身についていればよいと思います。

4.2.2 プログラムの実行と引数

メソッドがあるプログラムの実行時に表モデルがどのようになるかをトレースできるようになるのが重要で、これができれば、引数のあるメソッドは理解できたと思ってよいと思います。

あまり実際に表モデルを書く機会がないので、積極的に受講者に働きかけて、みんなで考えるような状況に持っていってください。

2 つ目の例題では、showAddResult() という新しいメソッドが追加されています。

最後に、配列の同期問題です。ここではうそを教えます。難しいので、もし理解できなければ、飛ばしても構いません。

4.3 練習問題

練習問題 1

引数なしメソッド作成の確認問題、簡単です。

練習問題 2

引数ありメソッド作成の確認問題、早く出来てしまう人には、他の部分もメソッド化する課題を与えてください。

練習問題 3

引数なしメソッド作成の確認問題、簡単です。早く出来てしまう人には、他の部分もメソッド化する課題を与えてください。

第 5 章

手続きとしての抽象化 (2)

この章で学習すること

戻り値のあるメソッドが書ける
戻り値のあるメソッドの実行の仕組みを説明できる
手続きが階層化されたプログラムが書ける
インスタンス変数を使ったプログラムが書ける

時間めやす

表 5.1: 時間めやす

5.1 節	30 分
5.2 節	60 分
5.3 節	30 分

5.1　手続きと値

学習目標

戻り値のあるメソッドが書ける
戻り値があるメソッドの実行の仕組みを表モデルを用いて説明できる
手続きも変数や式と同じように評価することができることを理解する

5.1.1　手続きの評価と戻り値

この章では、戻り値があるメソッドについて学びます。書法はもちろんですが、戻り値があるメソッドは評価すると return で書かれた変数が評価されたのと同じになるという

方法で理解します。

表モデルはこれまでと一緒ですが、return 文で戻り値に設定したことで、評価という今までも使ってきた概念を用いて戻り値のあるメソッドを使ったプログラムを説明することができます。

この考え方を理解していると、オブジェクト指向編で、new の評価で値（オブジェクト）が戻って来ることなどが容易に理解できます。

ちなみに、ここで getNumber() メソッドになると、いままで 2 つだったブロック (一つ目の数字を入力, 二つ目の数字を入力) が 1 つのブロックになっている（2 数を入力する) にできて、より簡潔に、読みやすくなっていると思います。

5.1.2　変数, 手続きと評価

このセクションでは、評価という概念を使ってプログラムを読んでいく場合、変数も手続きも差がなく、同じように評価の概念を使って把握できるということを学びます。このことは、今までしつこく評価の話をしてきたのでパイロットテストでも受講者はすんなり飲み込めたようでした。評価をすると「意味のある値を表す」という点で手続きも変数も同じだということを理解するのが目標です。この考え方で、プログラムを読み解く自信を付けることができます。

ここまでで、今まで使ってきた Input.getString() なども大体の意味がつかめるようになるので、こういった既存のメソッドを使って説明すると、より受講者は納得するという結果をパイロットテストから得ました。

考えてみよう

出題意図: 変数と手続きの関係について考えてもらうこと。

上の説明を踏まえ、基本的に同じだということに気づけばよい。違いは、手続きは計算して値を出すということだけです。

あとどうでもよい違いは、変数はメソッドの中に書き、有効範囲がメソッドの中なのに対して、メソッドの中にメソッドは書けない（すべてインスタンスメソッド）なこと。これは、インスタンス変数を学ぶと、同様になる。

5.2　プログラムの意味を明確にするための手続き

学習目標

何をメソッドにすべきか判断できる
メソッドを用いてプログラムの意味を明確化することができる
メソッドを用いて手続きに階層構造を持たせることができる

5.2.1 何をメソッドにすべきか

ここからは、自分でプログラムを書く場合に、どういったものをメソッドにするかという点を議論しながら学ぶということが目標です。受講者に考えてもらうために「考えてみよう」を多く用意しました。ある程度時間をとり、意見を発言してもらうのが効果的です。

順に重複コードから始まって、プログラムに意味に注目していこうという流れになります。

この話題に関しては、議論を中心に進めているので、その場では理解した気になっていても、自分でコードを書いてみると、良く分からないという印象も多くあるようでした。この議論が終わってからはソースコードレビューにおいても、より深いつっこみと議論ができると思うので、演習のレビューでも補強する形が望ましいと思います。

重複コードのメソッド化

考えてみよう

これは、単純にうれしいと思う。

プログラムを書くときに、カットアンドペーストをしてはいけない! （あとで、直すときに、なるべく少ない個所で済ませるため)

引数を使えば、データの違いまでも、重複コードとして扱える。

ただし、Java では、アルゴリズムの違いは抽象化できない。(メソッドが rst-class　ではないので)

意味を考えたメソッド化

考えてみよう

コメントが、そのままメソッドの名前になることが多いという議論をする。逆に、メソッドの名前が目的だと、コメントなしでプログラムに目的が反映できるとも言える。

うまく、メソッドを使うとコメントはいらなくなるが、基本的にまだコメントをつけること。と指導する。

意味と重複コードの関係

考えてみよう

基本的に、重複コードは目的が一緒の場合が多い。

しかし、重複コードでも、目的が違う場合、メソッド化すべきではない。（本当はいい例があればよい。難しいので、飛ばして構わない。）

5.2.2　意味を明確にするメソッド化

条件文のメソッド化

足し算アプリケーションに、isExitCode() メソッドが足されています。

考えてみよう

出題意図: メソッドの名前付けの議論をすること
レビューで指摘された isNoCharacter() 問題を議論することにしました。
終了コードを変えた場合どうなるかという議論をして下さい。

5.2.3　手続きの階層構造

手続きを呼ぶ手続き

このセクションでは、メソッドからメソッドを呼ぶことについてを議論します。「メソッドからメソッドを呼べる」ことを初学者は知らないので、まず、確認してください。

考えてみよう

出題意図: 次で議論するメソッドの階層化についての前フリ
例題では、showScoreList() から、getAverageStore() を呼ぶところだけが、階層構造になっています。

手続きの階層化

考えてみよう

目的の階層構造に関しては、HCP チャートで今までもやってきているので、HCP チャート主導で議論を進めるほうがスムーズです。
HCP チャートを見ながら、どのレベルで粒度をあわせて、メソッド化するかという議論をすると良いと思います。(このような話をすると、なんでもかんでもメソッド化する症候群がでてくるので、適切な粒度、レベルを議論するとよいと思います。)
あとここでもう一つ重要な話が、メソッドを書く順番です。基本的にコードは上から読むので、階層が上のメソッドから順番に並べていく、という読みやすさにこだわって書くという姿勢を指摘してください。(基本的にこの手の話は、ソースコードレビューで指摘することになります。)

5.3 共有される変数

学習目標

インスタンス変数を使ったプログラムが書ける
インスタンス変数とスコープの関係について説明できる

5.3.1 インスタンス変数

さらっとで構いません。

この話も「なんでもインスタンス変数症候群」がでるので、(この症状は非常にまずいです) 基本的に、「全体で使う変数だけをインスタンス変数にする」という話をして下さい。

何のために有効範囲があるのかという質問をして、範囲が明確だと読みやすいという議論をするとよいでしょう。

5.4 練習問題

練習問題 1

戻り値ありメソッド作成の確認問題、何をメソッド化するかという話も入ってきますが、getRorher() メソッドは必ず作るよう指導してください。インスタンス変数はもちろん禁止です。

練習問題 2

条件式のメソッド化問題、これも、インスタンス変数は禁止です。

練習問題 3

何をメソッド化するかの総合問題。配列と currentIndex だけインスタンス変数にしてもよい。

第 6 章

プログラムの効率

この章で学習すること

検索とソートの代表的なアルゴリズムについて説明できる
アルゴリズムの性能について比較検討し、説明できる
再帰を使ったプログラムが書ける

時間めやす

基本的に、出来る人だけ自習する。
再帰だけは、モデルを書かなければならないので、30 分程度必要。
この章は、時間が余った受講生がアルゴリズムを勉強するのための章です。飛ばして構いません。ただし、再帰の所だけは少しだけ触れる必要があります。(オブジェクト指向編で使うため)

6.1　検索アルゴリズムの効率

学習目標

検索の代表的なアルゴリズムを理解する
アルゴリズムによる検索の効率の違いを理解する

この節はオブジェクト指向哲学の 8 章と同じ内容です。

6.1.1　リニアサーチ

6.1.2　バイナリサーチ

6.1.3　効率の比較

6.2　並び替えアルゴリズムの効率

学習目標

並び替えの代表的なアルゴリズムを理解する
アルゴリズムによる並び替えの効率の違いを理解する

この節はオブジェクト指向哲学の 9 章と同じ内容です。

6.2.1　バブルソート

6.2.2　選択ソート

6.2.3　挿入ソート

6.3　手続きの再帰呼び出し

学習目標

再帰プログラムの実行の仕組みを理解する
再帰を用いた並び替えのアルゴリズムを理解する

6.3.1　プログラムの実行と再帰

考えてみよう

出題意図: 再帰プログラムの実行の仕組みを理解する
ポイントは、同じメソッドの表がたくさん出来ることです。(実際には、スタックに積まれていくので、表を下から書くように指導して、教えてしまっても良いかもしれません。)

6.3.2　マージソート

6.4　練習問題

練習問題 1

アルゴリズムの効率計測問題です。

実際に測って、(まずいアルゴリズムでは) 日が暮れるということを実感することは、学習者にとって非常にいい経験になります。

練習問題 2

再帰の練習問題です。
階乗プログラムの転移で簡単にできます。

第 II 部

オブジェクト指向プログラミング編

第 7 章

オブジェクトとしての抽象化 (1)

この章で学習すること

クラスを使うことの利点を説明できる
クラスとオブジェクトの違いを説明できる
インスタンスの関係をインスタンスの入れ子モデルで図に書くことができる

時間めやす

表 7.1: 時間めやす

7.1.1 節	30 分
7.1.2 節	30 分
7.1.3 節	30 分
7.1.4 節	30 分
7.1.5 節	10 分

7.1　クラスとインスタンス

7.1.1　同じ意味の変数はクラスに

学習目標

同じ意味の変数をまとめてクラスをつくる利点を理解する

ソースコードは、ミニプロの簡易バージョン。(ウインドウを使わないもの) オブジェクト指向編を始める前に、必ずミニプロをやっておくこと。

考えてみよう

出題意図: 変数のまとまりに名前をつけることでプログラムがわかりやすくなるということを理解する。

配列が増えることでだんだんソースコードがわかりにくくなっていくという議論が出来ればよい。

ここでは、電話番号も扱うようになった場合、変更のコストがかかることや、社員のパラメータを増やせば増やすほど配列に関するソースコードがごちゃごちゃしてくることなどに受講者に気づかせる。

ユーザインターフェースが変更されるという点に目が向く受講者が多いので、あまりその話が中心にならないようにする必要があります。

次の話の前フリなので、「社員」という概念をどう表すかという議論をすると良い。

7.1.2　クラスを使ったプログラムの記述

学習目標

クラスを使ったプログラムが書けるようになる

文法なのでさらっと

7.1.3　入れ子モデル

学習目標

インスタンスの構造を入れ子モデルをつかって書けるようになる
インスタンスが生成されるとはどのようなことがおきるのかを理解する
インスタンスが変数に代入されるとどのようなことがおきるのかを理解する
インスタンス変数に値が代入されるとどのようなことがおきるのかを理解する

この節では、オブジェクト指向編の最初のモデル、「入れ子モデル」を扱います。このモデルでは、インスタンスは変数の中に「入っている」という認識でオブジェクト同士の関係を理解します。(実はうそです)

まず、インスタンス生成の過程を、受講者に入れ子モデルで考えさせます。同時に基本的な入れ子モデルの記法についても認識を統一する必要があります。ここでは、new という処理を構造化編で学んだ式の評価の考え方をつかって説明します。また、変数とインスタンスの関係についてもここで確認する必要があります。変数はあくまでも入れ物であるというような認識を固めます。

7.1.4 複雑な構造を持ったオブジェクト

学習目標

インスタンスの入れ子構造を理解する
null とはどのようなことか説明できる

インスタンスが生成されるということがどのようなことか認識できたところで、インスタンス同士の関係に話を進めます。ここではプログラムの流れを追い、どのような代入が起きているのかを理解してもらいます。**代入が行われる際にはオブジェクトはコピーされる**というようにここでは教えます。参照がわかっていない段階では、同じオブジェクトが 2 つの変数の中に入っているということは教えられないからです。

バスから乗客が降りることをあらわすために、null が登場します。null 自体はわかりにくい概念ですが、モデルであらわすと何も入っていないということが null だという認識で話を進めます。

クラスとインスタンスについてわかってきたところで最後に、アプリケーションクラスの種明かしをします。static に関しては最後までふれないので、最初に呼ばれる特別なメソッドであるという認識だけ確認します。

7.1.5 クラスを使ったプログラム

学習目標

クラスを使うとプログラムが実際にどのように変化するのかを理解する

例題のみです。

7.2 練習問題

7.2.1 練習問題 1

入れ子モデルの記法とインスタンス同士の関係を確認します。

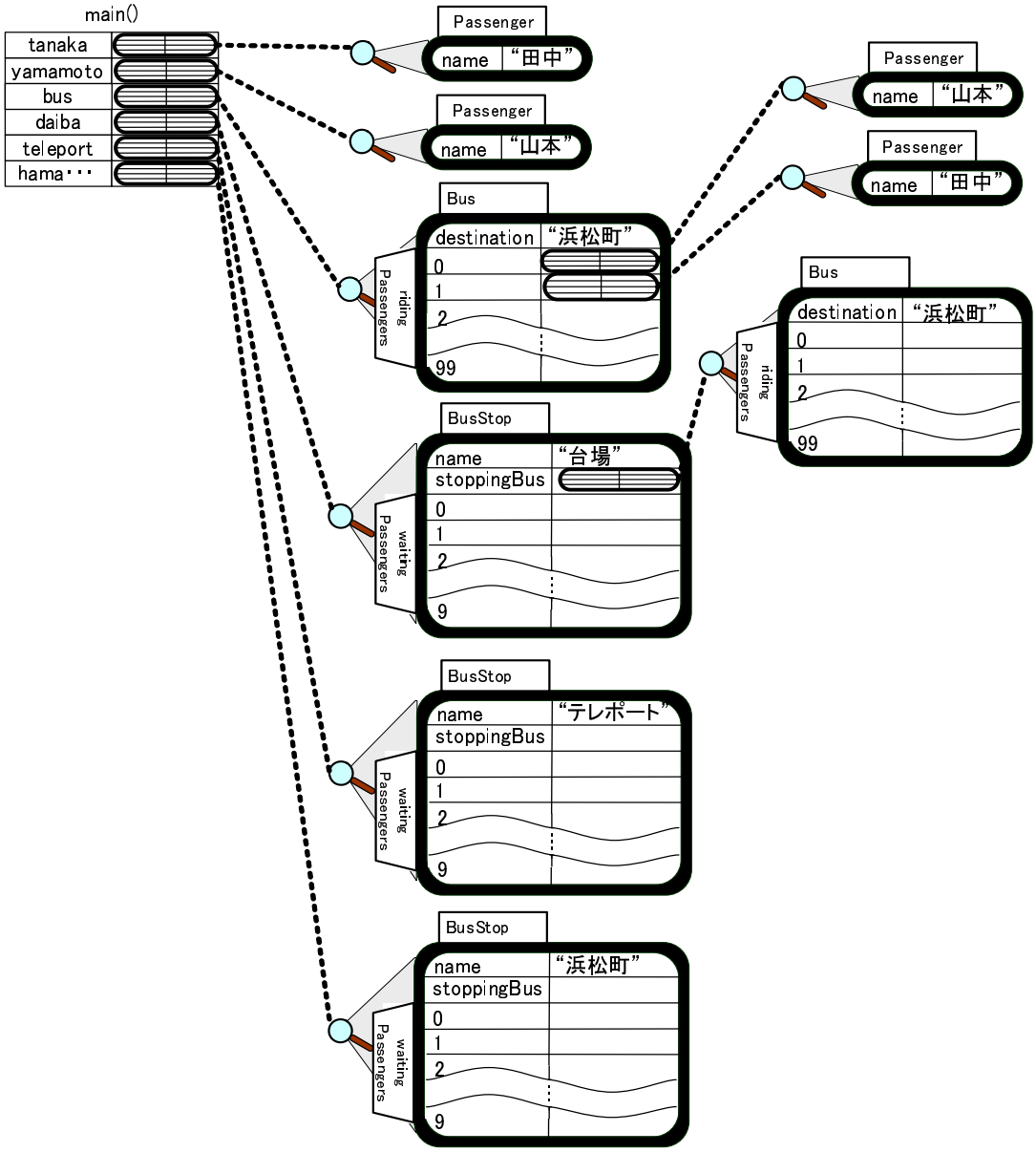

図 7.1: (8) 山本と田中がバスに乗る

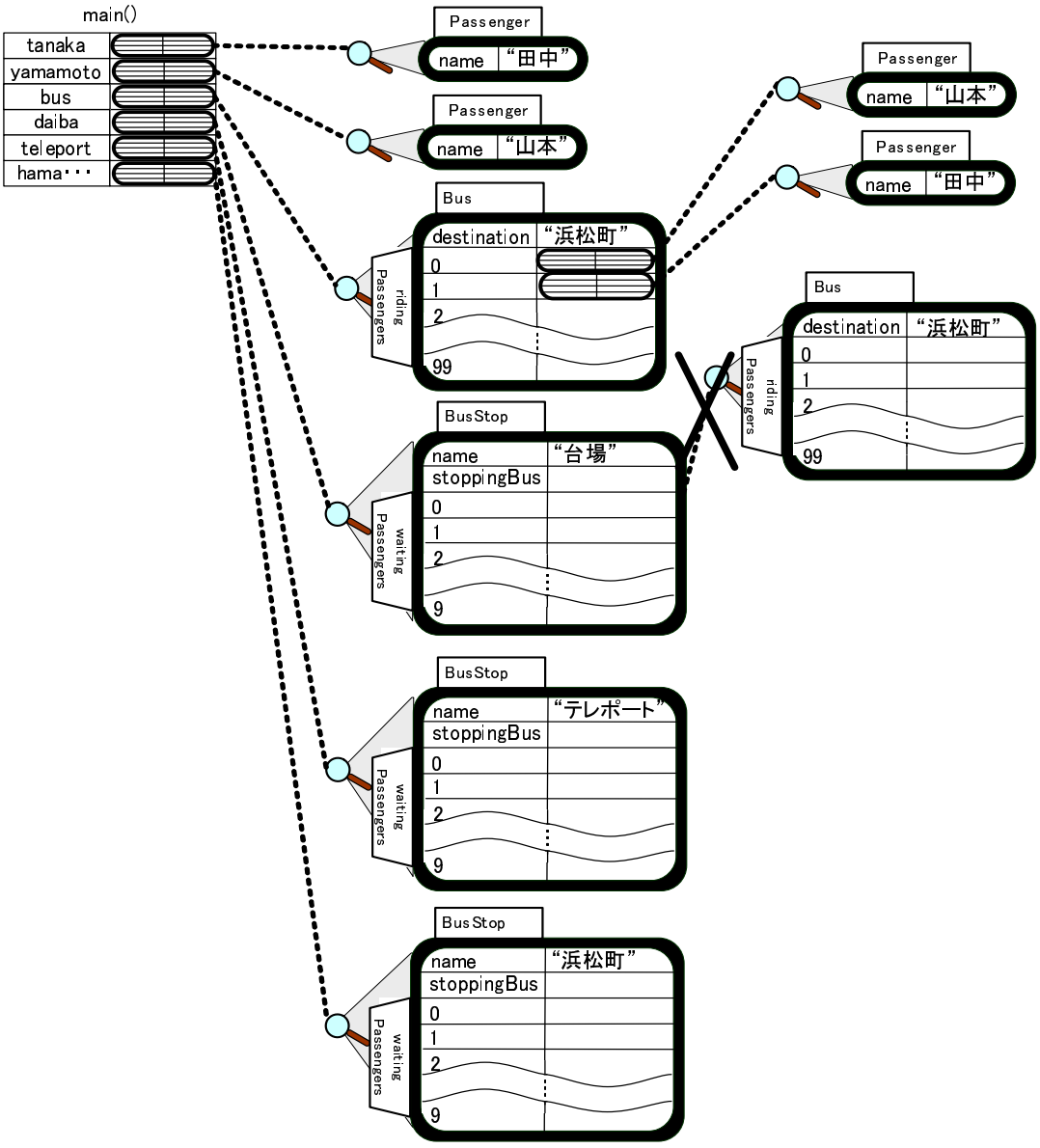

図 7.2: (9) バスが台場を発車

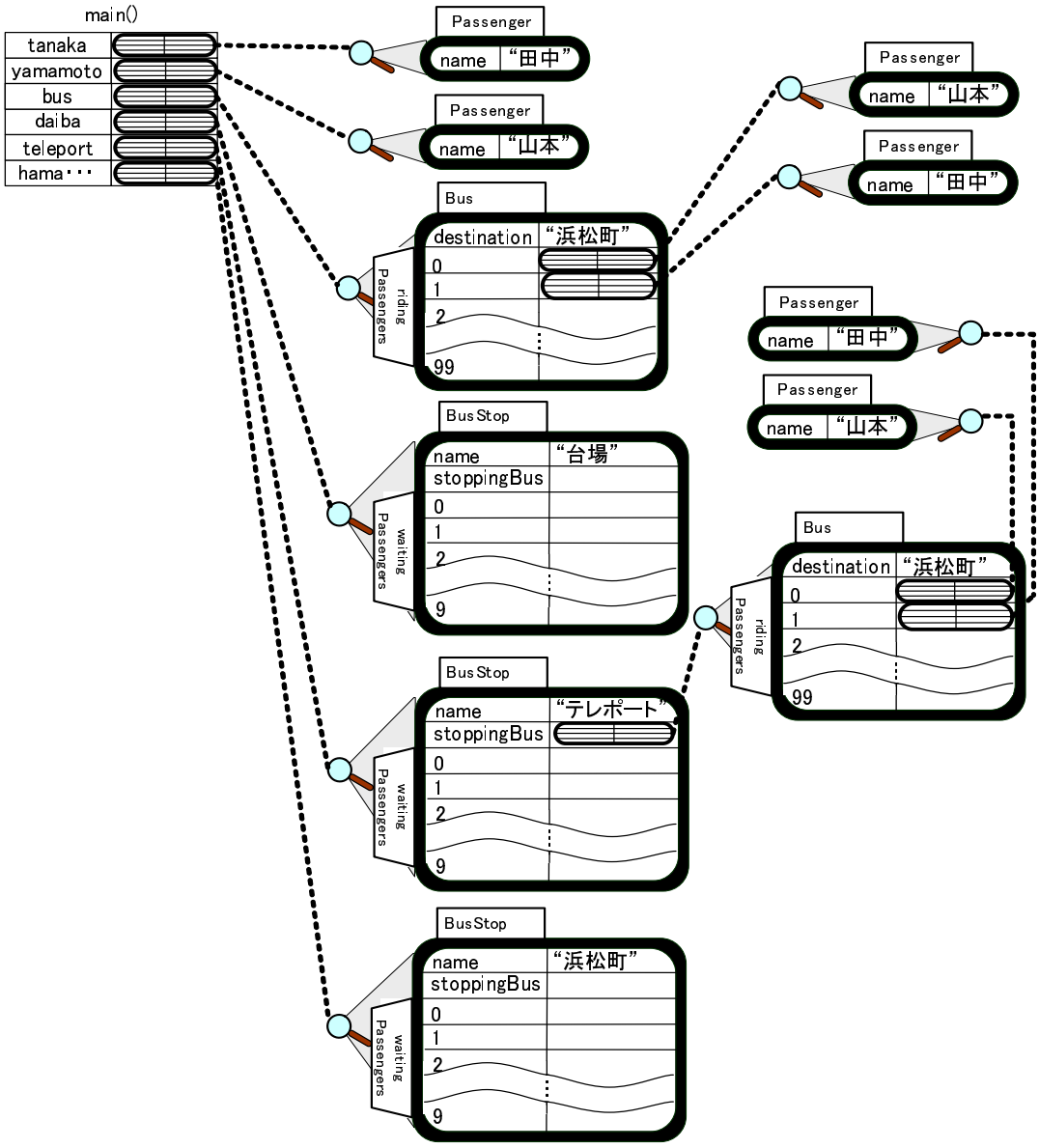

図 7.3: (10) バスがテレポートに停車

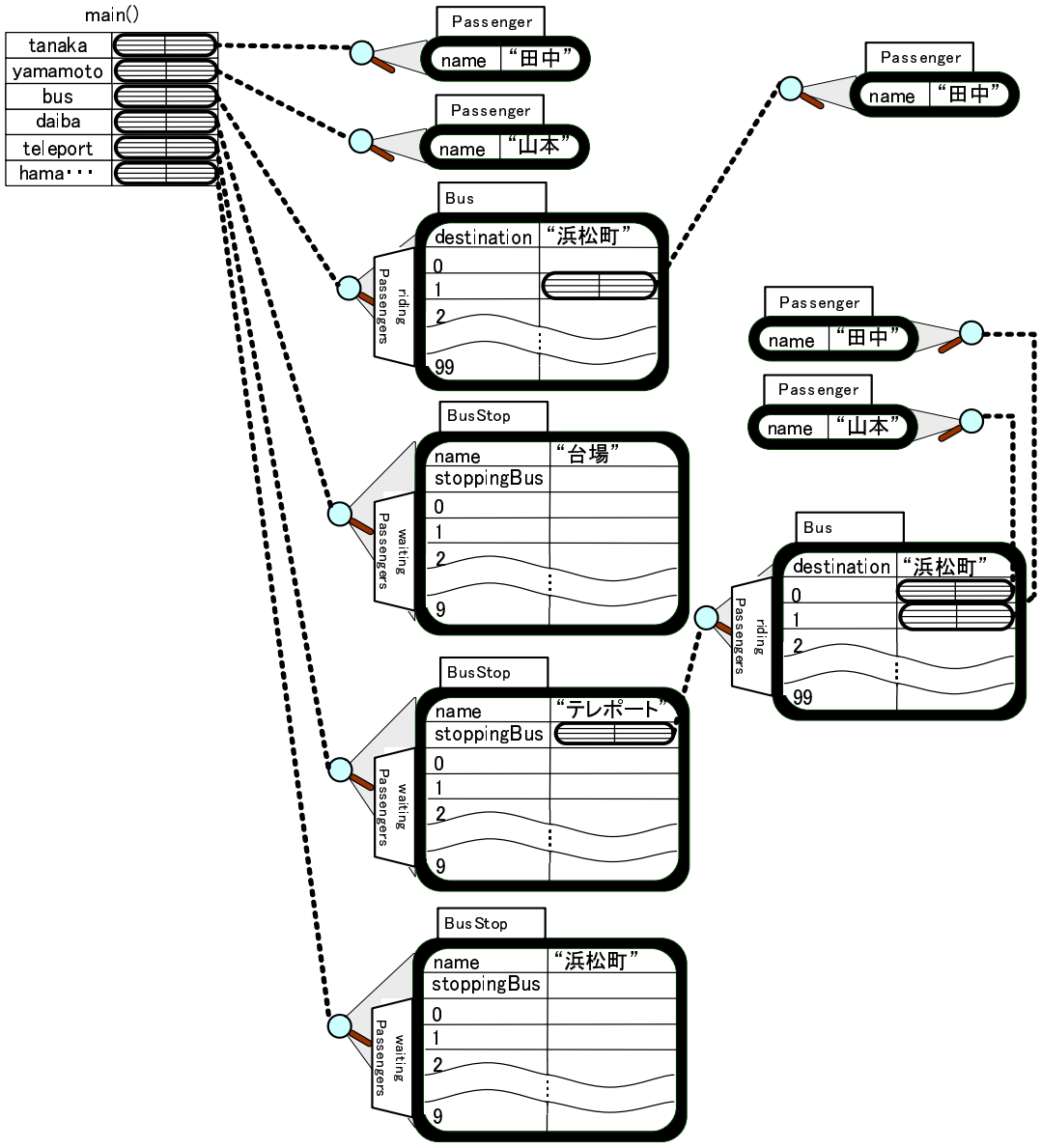

図 7.4: (11) 山本がバスを降りる

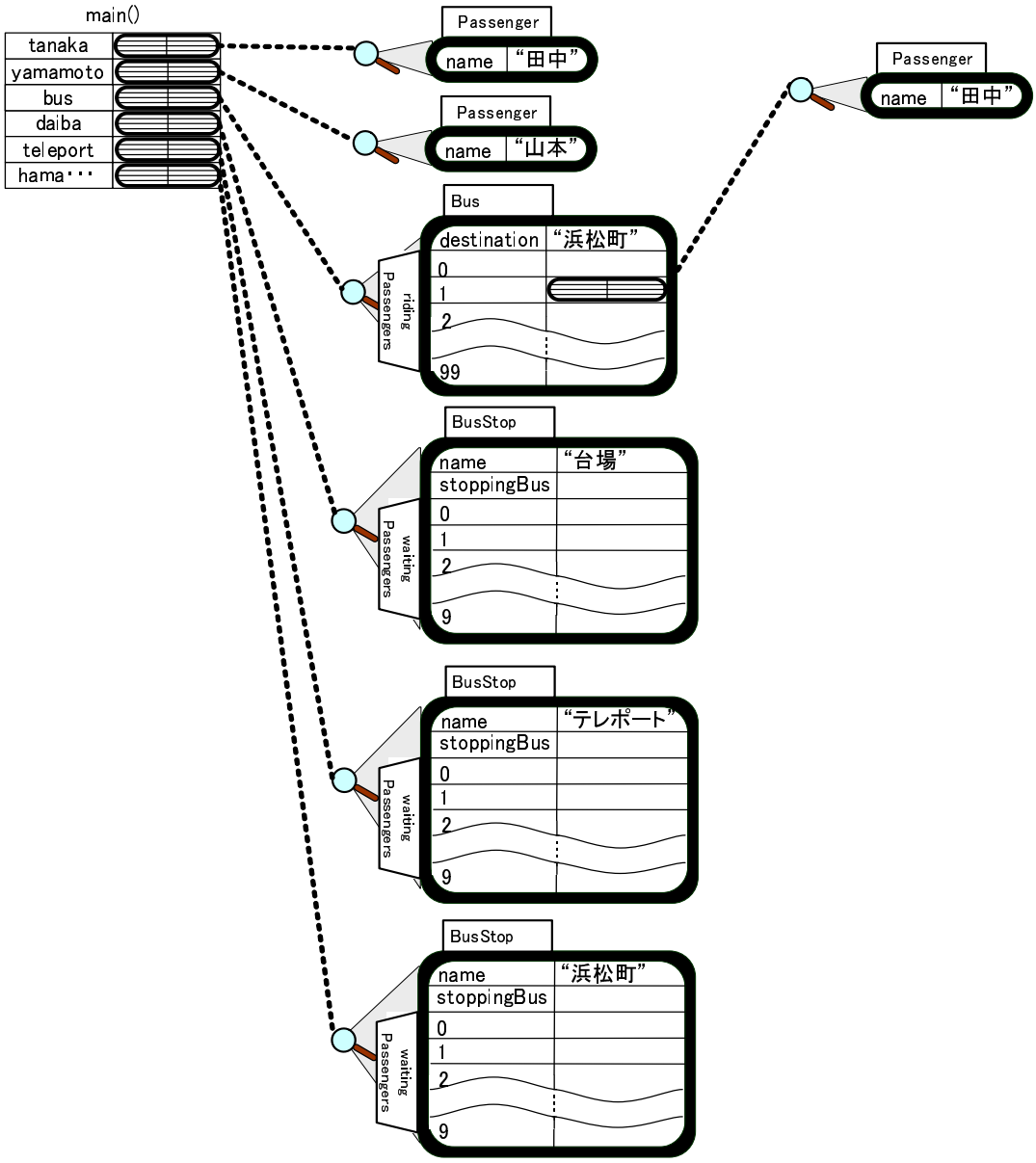

図 7.5: (12) バスがテレポートを発車

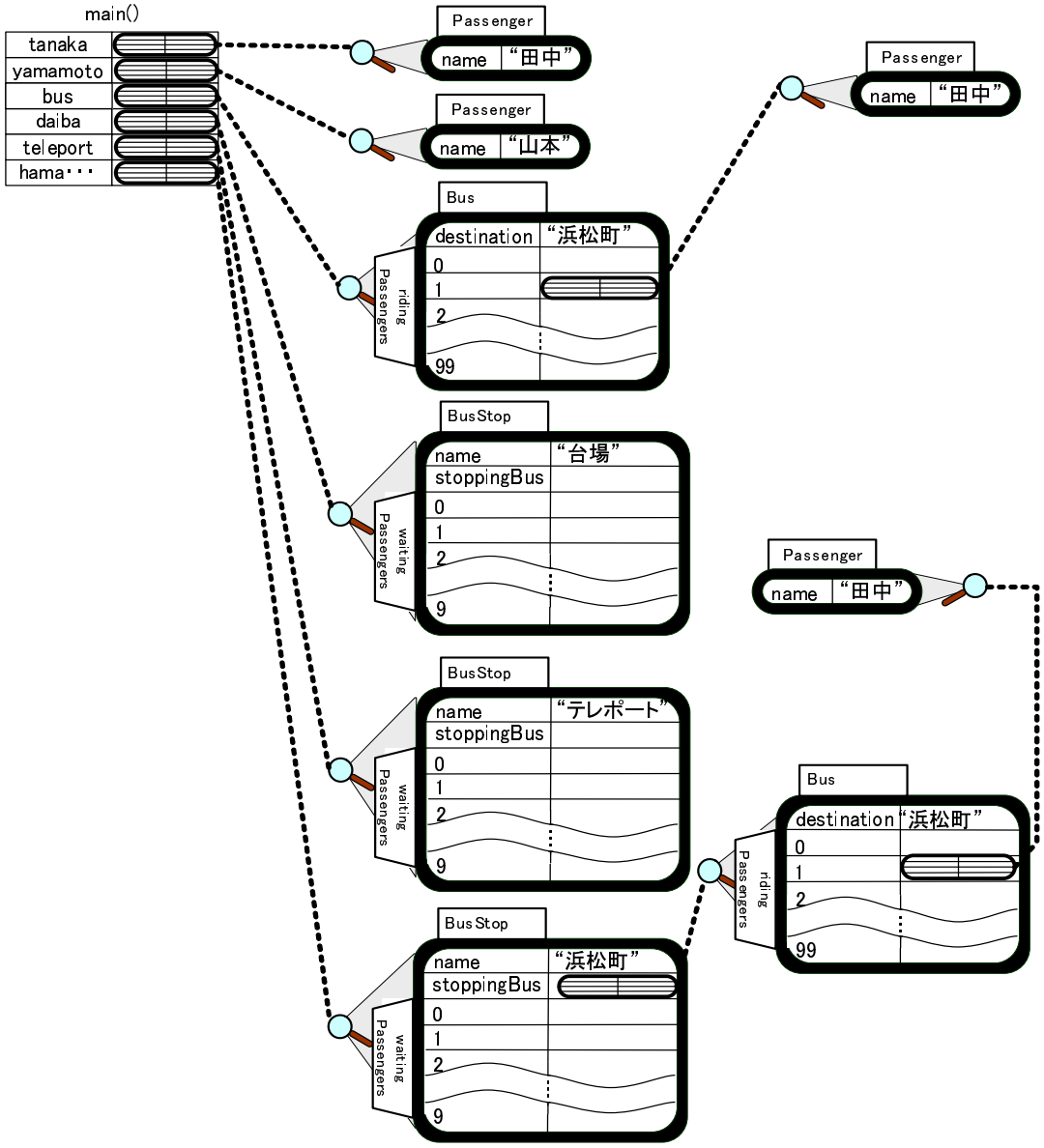

図 7.6: (13) バスが浜松町に停車

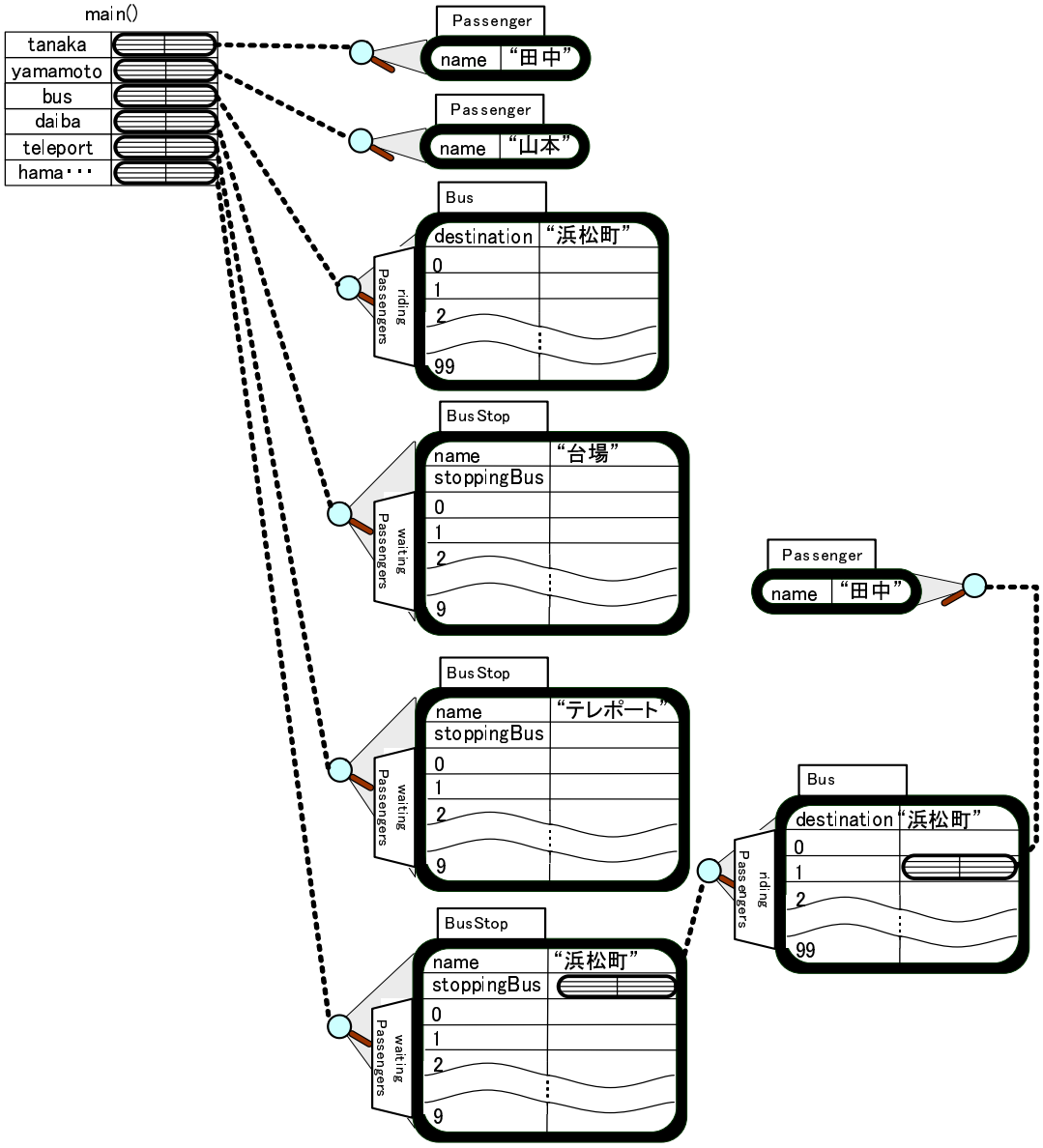

図 7.7: (14) 田中がバスを降りる

7.2.2 練習問題 2

成績管理アプリケーションにクラスを適用して、わかりやすくなるということを体験させます。変更の箇所は多いので、無駄なところで苦戦している人がいるならば置換や検索などをちゃんと使うように指示します。

7.2.3 練習問題 3

具体的には削除時に要素をつめていないことでエラーが出ます。わかりにくいようであれば、要素の削除後に検索をするというエラーのでる状況をヒントとして教えます。今後何度も出くわすであろうエラーへの対処方法を身につけさせます。

わからないときは削除と検索を行ったときにどのようにオブジェクトの状態が変わっているのかを入れ子モデルで書かせます。そうすることでなぜ問題がおきるのかが受講者にも明らかになります。words[i](Word オブジェクトの配列)、word(Word オブジェクトを格納するローカル変数)、word.spell(単語のスペル) などを受講者が混同するので、初心に帰ってそれぞれの変数を評価するとどうなるのかを考えさせます。

7.2.4 練習問題 4*

クラスの定義、インスタンスの生成などクラスを使うプログラムを書く能力を確認します

この問題は仕様は自由ということになっているので、パイロットテストの際に仕様に懲りすぎてしまう人がいました。あまり複雑な仕様にしないような指導が必要です。しかし、逆にちょっと複雑なことをやろうとすると、構造化編の理解が甘い部分が明らかになるので、受講者がつまっていたときは、丁寧に今まで習ったことを復習させるとよいでしょう。

7.2.5 練習問題 5*

時間の余る人には、構造化プログラムで学んだソートを実装してもらいます。新しい確認事項はありませんが、オブジェクトの扱いに慣れることがここでの目的です。よほど余裕がある場合にやらせるための問題です。

第 8 章

オブジェクトとしての抽象化 (2)

この章で学習すること

データと手続きの関係を説明できる
導出の考え方を説明できる

時間めやす

表 8.1: 時間めやす

8.1.1 節	60 分
8.1.2 節	60 分

8.1　データ抽象

8.1.1　導出されるデータ

学習目標

導出の考え方を通して、複雑なデータ構造やそれにアクセスするためのアルゴリズムを意識しなくてすむことがプログラムをわかりやすくすることにつながるという意識を持つ
インスタンスメソッドを使ったプログラムが書ける

考えてみよう

出題意図: データを計算してもとめることで、アプリケーションの管理が楽になる場合があることを理解する

毎年 1 月 1 日になるたびに、全ての社員の年齢を 1 増やす必要がある

考えてみよう

出題意図: オブジェクトが持つデータとは変数として持っているものだけではないことを理解する

インスタンス変数として取得することができるデータと、手続きを通して取得できるデータは、両方ともオブジェクトが持つデータということができる

導出を行わないとプログラムの管理コストが高くなるということに注目します。より大規模なプログラムであったらどのようなことになるのかという話をすると、より認識が確かなものになります。

導出を使うとこの場合は管理コストが減るという点を理解させる一方、いつでもそれが正しいわけではないという話をします。導出を導入すると計算コストが問題になる場合の例を挙げます。

導出の利点を理解したところで、アプリケーションのクラスに導出メソッドがあるのでは結局変更の範囲が広がってしまうという点に注目します。この流れで、インスタンスに関するメソッドは、インスタンスメソッドにするという考え方と、その書法を理解してもらいます。

さらに、導出しない属性にも getter をつけることにします。ここでの getter のモチベーションとしては、現在は導出しない属性も将来は導出する可能性があるので getter をつけておくという意識を持たせます。

インスタンスメソッドの導入が完了したところで、データとは何かについてもう一度考えさせます。ここで重要なことはデータが、属性としてもたれているか、メソッドとして提供されているかは、意識する必要がないということです。内部を意識する必要がないとプログラムが書きやすいということはこの章を通じてのテーマなので、繰り返し教えます。

5 章で議論した、変数と手続きの関係とつながっているので、もう一度復習すると良いでしょう。

8.1.2　複雑さの隠蔽

学習目標

配列を管理するクラスを使ったプログラムを書ける
複雑なものを隠蔽することで目的に近いプログラムが書けることを理解する

考えてみよう

出題意図: リストクラスを導入することで、より目的に近いプログラムが書けることを確認する。

「社員を削除する」という目的に近いプログラムが書ける

考えてみよう

出題意図: データ構造とアルゴリズムを隠蔽することで、変更に強いプログラムが書けることを理解する。

リストのデータ構造を変更してもアプリケーションのクラスには変更が起きない。オブジェクト指向の考え方を使いデータ構造とアルゴリズムを隠蔽すると、変更の影響する範囲をせばめることができるから。

この節の目的は、複雑な処理をクラスとして分割し、内部の複雑な処理に関しては意識しないで済むようにするという考え方を学ぶことです。例として配列を管理するクラスを社員名簿アプリケーションに導入します。

複雑なものは一度に扱うことができないというのはこのカリキュラム全体を通しての一つの考え方です。ここでも配列の例を通して、その認識を再確認し、リストクラス導入のモチベーションを確認します。

次にリストクラスの仕様と、何を隠すのかについての話をします。具体的な”もの”ではないクラスが登場するのは初めてなので、具体的なイメージを共有するために、詳しく説明する必要があります。

最後にこの章のまとめとして、複雑なものを隠し、内部を意識しなくてよくなると、使う側のプログラムではプログラムの意味だけに注目してプログラムが書けるという点に注目します。これは次章でのクラスの責任に関する議論への布石です。

8.2 練習問題

8.2.1 練習問題 1

メソッドによるデータ導出の練習問題です。いずれも生まれ年から導出できます。

8.2.2 練習問題 2

インスタンスメソッドの使い方の確認です。パイロットテストでは、「.」をつかってメソッドにアクセスすることをやろうとすると、今までできていたはずのメソッドの返り値

や引数がわからなくなる人があらわれました。復習する意味で、変数、メソッドを評価すると何が返ってくるのかを考えさせてください。

8.2.3　練習問題 3

導出を使ったデータ構造設計の練習問題です。星座を生データとして持たせるのではなく、生年月日を持たせるのが正解。分からない人がいたら、さらに誕生石を表示する場合を考えさせると良い。

8.2.4　練習問題 4

配列を管理するクラスを導入すると、アプリケーション全体でどのような変化が起こるのかを認識させる確認問題。これも今までの問題と同じで、複雑なことをやると今までやったことも混乱するという人が現れます。わからなくなったら、評価、入れ子モデル、という考え方を徹底させましょう。

8.2.5　練習問題 5*

内部の実装が変更されてもアプリケーションの変更が必要なくなるということを体験させる問題。時間が余った人のための問題です。

第 9 章

オブジェクトとしての抽象化 (3)

この章で学習すること

カプセル化の利点を説明できる
データ構造とアルゴリズムを結合させる意味を説明できる
スタックとキューを使ったプログラムが書ける
クラスの単体テストを行うことができる

時間めやす

表 9.1: 時間めやす

9.1.1 節	30 分
9.2.1 節	20 分
9.3.1 節	20 分
9.3.2 節	10 分
9.3.3 節	20 分
9.3.4 節	20 分

9.1　カプセル化

9.1.1　破られる紳士協定

学習目標

カプセル化の必要性を理解する
アクセス修飾子を使ったプログラムを書ける
setter をつかったプログラムが書ける

考えてみよう

出題意図: setter を作る意味を理解する

アクセス修飾子とあわせて使うことで、get はできるが set はできないというような細かいアクセス制限ができる
setXXX というメソッドの呼び出しがあることで、代入文よりもインスタンスのデータを設定しているということがわかりやすくなる
変更のログをとることができる

この節の目的は、前の章で学んだデータ抽象をより完全なものとするための手法としてカプセル化の考え方を学ぶことです。カプセル化はひとりでプログラムを書いている上ではあまり必要のない考え方なので、将来多人数でプログラムを書くことがあるという前提を意識させる必要があります。

まずカプセル化をしないとどのような問題点があるのかを受講者に認識させます。例題の意図は、size–はする必要が無いのに（使う人が知らなかったので）してしまったというものです。size はいじってはいけないというような暗黙のルールはバグの原因となることを理解することが重要です。

カプセル化を実現するために、アクセス修飾子を導入します。アクセス修飾子は現状で理解することができるのは public と private だけです。それ以外に関してはさらっと進めてください。

ここで this. を導入します。this. は参照がわかっていないと自分自身の参照という本当の意味を教えることはできません。ここでは「クラスのスコープの」という説明でごまかします。

最後に setter を導入します。setter の導入はとりあえず強制し、あとで理由を考えてもらいます。ログ出力が要求される可能性や、set と明示されていることで、代入文よりもプログラムがわかりやすくなることに注目するようインストラクションをします。

9.2　オブジェクトの初期化

9.2.1　コンストラクタ

学習目標

オブジェクトにも初期化という概念があることを理解する
コンストラクタを使ったプログラムを書ける

考えてみよう

出題意図: コンストラクタを作る意味を理解する

> コンストラクタに初期化の処理を書くことで、生成されたインスタンスが安心して使えるものであることを保証することができる。
> setter,getter, アクセス修飾子とあわせてつかうことで、より柔軟なアクセス制限ができる

この節の目的は、コンストラクタを使わなければいけない場面を認識することです。また、変更できない変数というケースを通じてカプセル化の確認をします。

また同時に、オブジェクトを安心して使うために、コンストラクタを使うという話もします。getName したけど null が返ってくるとかは困るという話をしましょう。

9.3 意味のまとまりとしてのオブジェクト

9.3.1 アルゴリズムとデータ構造を結合する意義

学習目標

> アルゴリズムとデータ構造を結合することで意味のあるまとまりができることを理解する
> クラスが持つ責任とはどのようなものか理解する

考えてみよう

出題意図: 今までのクラスを例にとってクラスが持つ責任とはどのようなものか理解する

> Employee・・・社員が持つデータについて責任を持つ
> EmployeeList・・・社員の配列の管理について責任を持つ
> EmployeeDirectoryApplication・・・アプリケーションのユーザインターフェースについて責任を持つ

この節の目的は、データ抽象のまとめとしてアルゴリズムとデータ構造の関係について理解することです。両者が不可分なものであることを理解することが重要です。

アルゴリズム＋データ構造→意味のあるオブジェクトであると認識を受講者に持たせます、データ構造の例としては、これまであまりたいしたものはでてきていませんが、配列と現在のサイズをあらわす整数値がセットで配列のリストであるという例を挙げて、受講者のイメージを固める必要があります。

最後にクラスの責任についての話をします。データ抽象の考え方を適用し、他のクラスの内部を意識しなくてよくなると、どこに注目してプログラムを書けばよいのかというのが明らかになります。それがクラスの果たすべき責任であるということを受講者に理解させます。

9.3.2　クラスが持つ責任とテスト

学習目標

クラスの責任を単体でテストするという考え方を身につける

単体テストに関する部分は他には依存しません。時間がなければ飛ばしてください。

クラスの責任を明らかにし、他のクラスについて意識しなくてよくするということは、他のクラスが間違っていたときにどのようなことになるのかという話をします。一つ一つのクラスの正当性をテストすることで全体の正当性が保たれるのではないかという認識を持たせます。

また、何をどこまでテストするのかということに関してはここでは深くは触れません。簡単でもいいからテストをするという考え方を持つことがここでの目的です。

9.3.3　スタック

学習目標

スタックの概念とその実装を理解する

考えてみよう

出題意図: スタックのアルゴリズムについての理解を確認する

プリンタの用紙
食器棚のお皿

スタックに関する部分は他には依存しません。時間がなければ飛ばしてください。

この節と次のキューの節の目的は新しいデータ構造を通じて、クラス、インスタンス、データ抽象というここまで学んだ考え方を復習することです。また同時にそれぞれのデータ構造が情報システムの多くの場面で使われていることを 認識させます。

アルゴリズムに関してはスライド形式でさらっと説明してください

9.3.4 キュー

学習目標

キューの概念とその実装を理解する

考えてみよう

出題意図: スタックのアルゴリズムについての理解を確認する

エレベータの待ち行列

キューに関する部分は他には依存しません。時間がなければ飛ばしてください。
目的はスタックと同じです
アルゴリズムに関してはスライド形式でさらっと説明してください。

9.4 練習問題

9.4.1 練習問題 1

小規模なアプリケーションにカプセル化を導入することで、カプセル化を使ったプログラムを書けるようになることがねらいです。基本的にはアクセス修飾子をつけるだけですが、オブジェクトの使い方がわかっているかどうかがわかるので、わかっていない人がいたら評価、入れ子モデルなどを持ち出して再確認しましょう。

9.4.2 練習問題 2

テストの必要性を確認するのが目的です。また、あまり懲りすぎると無駄に時間がかかるので、あまり時間をかけさせないように指導してください。

9.4.3 練習問題 3

スタックのテストと同じです。

第 10 章

オブジェクトのネットワーク構造 (1)

この章で学習すること

参照型と基本データ型の違いを説明できる
インスタンスの関係をインスタンスの参照モデルで図に書くことができる

時間めやす

表 10.1: 時間めやす

10.1.1 節	40 分
10.1.2 節	20 分
10.2.1 節	30 分
10.2.2 節	10 分
10.2.3 節	20 分

10.1　参照

10.1.1　インスタンスの参照モデル

学習目標

入れ子モデルはインスタンス同士の関係を正しく表していないということを理解する
アクセス修飾子を使ったプログラムを書ける
setter をつかったプログラムが書ける

考えてみよう

出題意図: 部署を文字列で扱うことの問題点を理解する

全ての社員から、変更したい部署に所属している社員を検索し、それぞれの社員が持つ部署名を変更する必要がある。

考えてみよう

出題意図: 「オブジェクトがオブジェクトを持っている」という考え方は矛盾があるということを理解する

「二つの社員インスタンスの変数に、ひとつの同じ部署が入っている」という不思議絵のようなモデルができる。

ここの目的は、部署の例を通じて、参照の仕組みを理解することです。今までの入れ子モデルはまちがっていたということも認識します。

この章では社員名簿に部署の管理という新たな機能を追加します。社員名簿だからといって、社員の情報だけ管理していればよいというわけではないことを話し、ありうる機能追加であることを受講者に認識させます。また、機能としては、部署の追加、削除、名前変更を行うこととします。

まず今までの構造では部署の名前変更をするのに、どのような処理が必要なのかを受講者に考えてもらいます。結果として同じものをひとつのオブジェクトにしたいというモチベーションがわくようにインストラクションします。

部署クラスの導入へ向けてのモチベーションを共有したところで、入れ子モデルの矛盾について理解させます。みんなが同じものを持っているという状況は入れ子モデルでは説明できません。ここで、入れ子モデルが間違っていたことを明かします。

ここで参照の仕組みを教えます。参照は口で説明してもわかりづらいので、モデルの図を出して、参照モデルで言う黒丸がその変数に対応している値なのだということを認識させます。

最後に参照の仕組みをつかった社員名簿の変更を説明します。部署リストクラスの導入は一見必要なさそうに見えるので、なぜ全部署への参照が必要なのかという点をインストラクションします。

10.1.2　同じとはどういうことか

学習目標

参照が同じであることと値が同じであることは違うということを理解する

考えてみよう

出題意図: .equals() と==演算子の違いを確認する。

```
str1 == str2 ではない
str1.equals(str2) である
```

ここの目的は、参照が同じことと、値が同じことは違うということを理解することです。今まで避けてきた equals の説明をします。しかし、String で equals を扱うときの定数プールに関する説明は不要なので、そこには触れないようにします。

まず今まで扱った社員クラスの属性の中で、基本データ型の値が同じなのと、参照型の参照が同じなのではどうちがうかについて考えてもらいます。ここで、同じ入社年、部署を持つ社員二人の例を挙げ、片方を変更したときにもう一方はどうなるか考えてもらうとより理解がスムーズに進みます。

2 種類の「同じ」があるという認識ができたところで、==と.equals の説明をします。同時に基本データ型の存在についての説明もします。

10.2　オブジェクトの構造とナビゲーション

10.2.1　参照の方向

学習目標

参照には方向があり、それぞれ場合によって使い分ける必要があることを理解する。

考えてみよう

出題意図: 相互参照と単方向参照の利点と欠点を理解し、それらが場合によって使い分けるべきものであることを理解する。

単方向参照では検索などのときに多くの計算が必要になることがある
双方向参照は構築するのに参照を渡す手間がかかる
双方向参照を使いすぎると構造が複雑になり、わかりにくくなる

ここの目的は、参照には方向があり、必要に応じて単方向、双方向の参照を実装する必要性があることを理解することです。方向についての議論は、今までの参照モデルの議論が理解できていればそう難しくはないので、さらっと説明します。

10.2.2　ネットワーク構造の構築

学習目標

オブジェクトのネットワーク構造をつかったプログラムが書けるようになる。

ここの目的は、オブジェクトがネットワーク構造になっていることを理解し、その実装ができるようになることです。

この部分はネットワーク構造に必要な技術的な知識がほとんどなので、飛ばしても構いません。ただし、ミニプロ等のときに必要になるとき思うのでその際にはここを参照させるようにしてください。

保存、読み込みに関してはさらっとで構いません。ただし大元の参照が必要だという議論は難しいので、記録係のクラスがなかったときは何が保存できるのかを考え、必要性を認識させます。

10.2.3　オブジェクトの導出

学習目標

オブジェクトを導出するという考え方を通じて、構造を場合によって選択する必要があることを理解する。

この節は難しいので、飛ばしても構いません。ただし、次章のソースは導出バージョンになっているので気をつけること。

この節の目的は、参照の仕組みを踏まえて、オブジェクトの構造への理解をより深めることです。今まではただ値を出すだけだった導出の考え方を、オブジェクトの構造を導出するという、より大きな場面で適用します。

まず、部署と社員を相互参照にすることを考えます。これは次の部署リストからの全社員リストの導出への布石です。ここではすぐに相互参照という言葉を出すことはしませんが、参照の話しがすんなりと理解できていれば、相互参照はとくに問題にならないはずです。

現状の社員名簿プログラムは社員を追加するときに、アプリケーションが持つ全社員リストと、それぞれ社員が所属する部署の 2 箇所に社員を追加する必要があります。このような 2 箇所に追加するという暗黙のルールはわかりにくいというモチベーションから、社員リストを導出することを考えます。ただし、ここでは導出することの利点、欠点をインストラクションすることが重要です。規模が大きくなったときに、全社員のリストをいちいち導出するのは非現実的である可能性にも触れます。この話は、次章での設計は吟味することであるというトピックスへの布石です。

10.3 練習問題

10.3.1 練習問題 1

参照モデルをつかって参照の仕組みを理解しているか確認します。

10.3.2 練習問題 2

簡単なネットワーク構造を自分で実装することができるかを確認する問題です。

10.3.3 練習問題 3

前の問題とやることは同じなので、飛ばしても構いません。ただし若干こちらのほうが複雑になります

第 11 章

オブジェクトのネットワーク構造 (2)

クラスの再帰構造を発見することができる連結リストを使ったプログラムが書けるクラス図をもとに参照モデルの図を書くことができる

時間めやす

表 11.1: 時間めやす

11.1.1 節	5 分
11.1.2 節	10 分
11.1.3 節	10 分
11.1.4 節	10 分
11.2.1 節	30 分
11.2.2 節	35 分
11.3.1 節	20 分

11.1 クラス構造の図解

11.1.1 クラス図

学習目標

複雑なクラス構造を図に表すためにクラス図というものがあることを理解する。

ここでは複雑なクラス構造を表現するためにクラス図の導入を行います。ここでは紹介するだけです。

11.1.2　クラスの表現

学習目標

クラス図におけるクラスについての記法を覚える

クラスを表現する記法について学びます

11.1.3　関連の表現

学習目標

クラス図における関連についての記法を覚える

関連を表現する記法について学びます。多重度、ロールはどちらのクラスからみてかということがわかりにくいのでその点を強調する必要があります。

11.1.4　クラス図の曖昧さ

学習目標

クラス図と参照モデルの違いについて理解し、それらを使い分けられるようにする。

考えてみよう

出題意図: クラス図と参照モデルは片方だけあればいいというものではないことを理解する。

クラス図だけでは、実際にどのように（どのようなルールで）オブジェクトがつながっているのか分からない。
オブジェクト図では、全ての場合を書けない。（もしくはたくさん必要でわかりにくい）
結論:クラス図（制約無し）とオブジェクト図、片方だけではオブジェクトの世界を表せない。

ここでの目的は、ポーカーの例を使って多対多のクラス図は制約がなければ解釈できないということを理解することです。制約は難しいので、それには深く触れず、ここではインスタンスの参照モデルを書くことでそれを解決します。

具体的にはこのクラス図は、カードが、常にプレーヤーか山のいずれかにしか参照をもたれないということを表現できていません。

11.2　クラスの再帰構造

11.2.1　階層構造の表現

学習目標

階層構造とはどのような構造であるかを理解する
クラス構造で階層構造を表現するために、再帰の考え方を使うことができる
クラスの再帰を使ったプログラムを書くことができる

考えてみよう

出題意図: 実世界のいろいろなとことに階層構造が使われていることを理解する

ファイルシステム
Windows のスタートメニュー
住所

ここの目的は、クラスの再帰構造を理解し、階層構造を再帰で表現することができるようになることです。社員名簿アプリケーションの部署と、ディレクトリ管理アプリケーションのディレクトリを例に再帰を学びます。

まず部署は本来階層構造で成り立っているということを理解させます。同時に階層構造にはどのようなものがあるかを考えさせ、例を挙げます。

再帰構造を部署クラスに適用することを考えます。クラス図では一つしか現れないクラスが、実際には無限に続く可能性のある、インスタンスの階層構造を表現しているということを実感させることが重要です。最上位の要素への参照を持つ話はさらっと進めてください。

最後にもう一度ディレクトリの例をつかって再帰について理解を深めます。この例のほうがわかりやすいので、部署のほうはさらっと進めここでしっかりとクラスの再帰構造とインスタンスの階層構造を対応付けるといいでしょう。

11.2.2　連結リスト

学習目標

クラスの再帰を使ったプログラムを書くことができる
連結リストのアルゴリズムを理解する

ここの目的は、連結リストのアルゴリズムを通じて再帰の確認を行うことです。また同時に、配列だけがデータ構造ではないということも認識させます。

連結リストは参照の練習です。余裕がなければ飛ばしください。

連結リストのアルゴリズムはスライドを使って説明してください。要素の削除を行うときの、要素が先頭、中間、最後尾のそれぞれにある場合のアルゴリズムの違いが難しいので、そこは詳しく説明する必要があります。

11.3　人にやさしいクラス構造の設計

11.3.1　現実世界に即したクラス構造の設計

学習目標

> 設計にひとつの明確な正解などなく、それぞれの場合において利点欠点を吟味しながら選択肢を選んでいく作業であることを理解する。

ここの目的は、設計とはどのような作業なのかを体験してもらうことです。実際にはモデリングの初歩ですが、概念、分析といった言葉は難しいので使いません。

ここの内容は難易度も高いので飛ばしても構いません。他の部分への依存はありません。

ここでは、上司という社員クラスのデータを導入することを考え、2 種類の設計を示します。重要なのはその選択肢を考えさせることと、選択肢の利点欠点を議論することです。

まとめとしてクラス構造の設計とは、何をクラスとするかの選択肢を挙げ、それぞれの選択肢の利点と欠点を考えて選択する作業であるという話しをします。この選択肢を挙げ、吟味するという考え方はカリキュラム全体でででてくる考え方なので特に強調してください。

最後にネットワーク構造と階層構造の対比をします。ここではネットワーク構造のほうが現実世界に近いが、複雑だという難点があり、階層構造はその逆だということを理解させることが重要です。

11.4　練習問題

11.4.1　練習問題 1

クラス図と参照モデルの変換を行い、片方があればいいというものではないということを確認します。

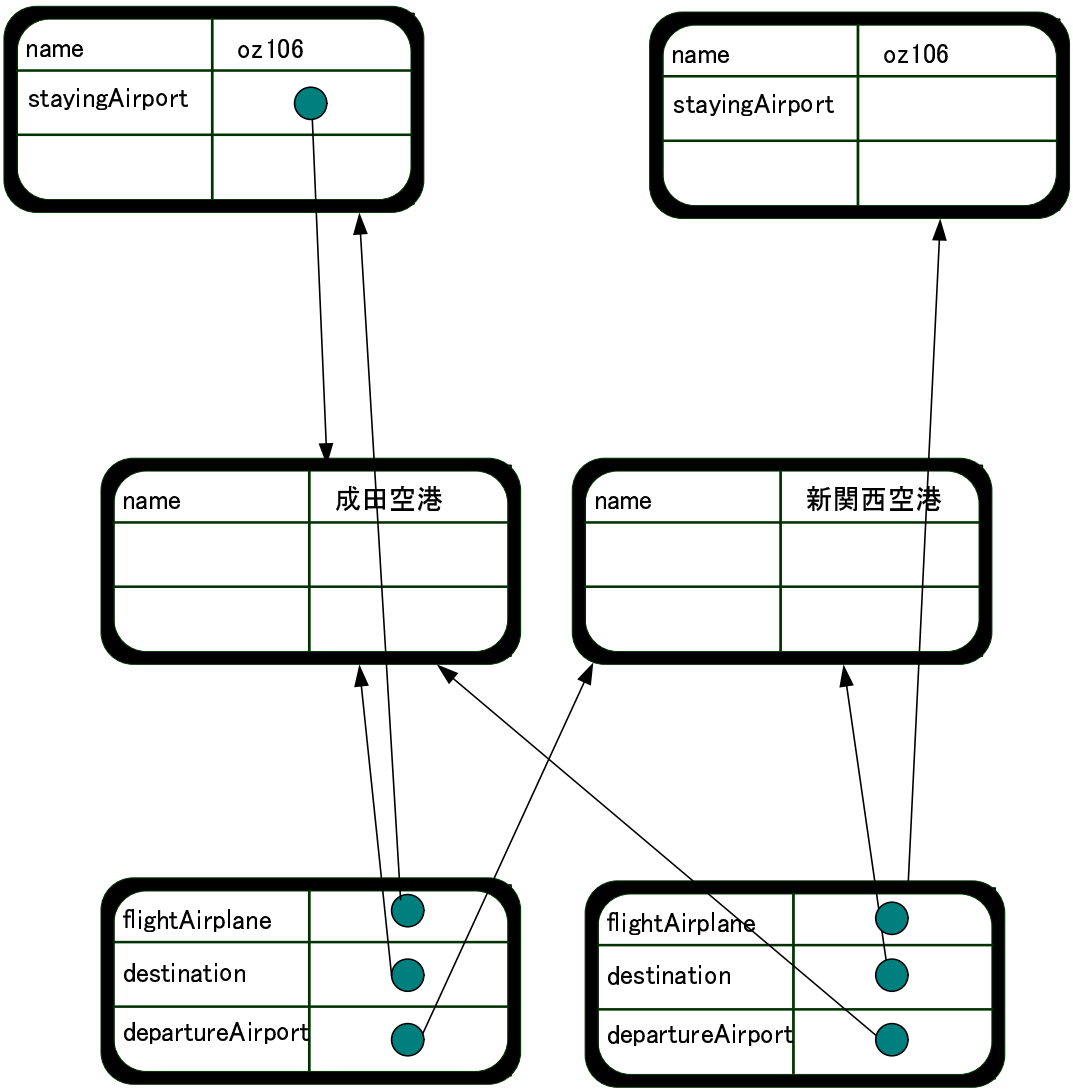

図 11.1: 空港管制塔システム参照モデル

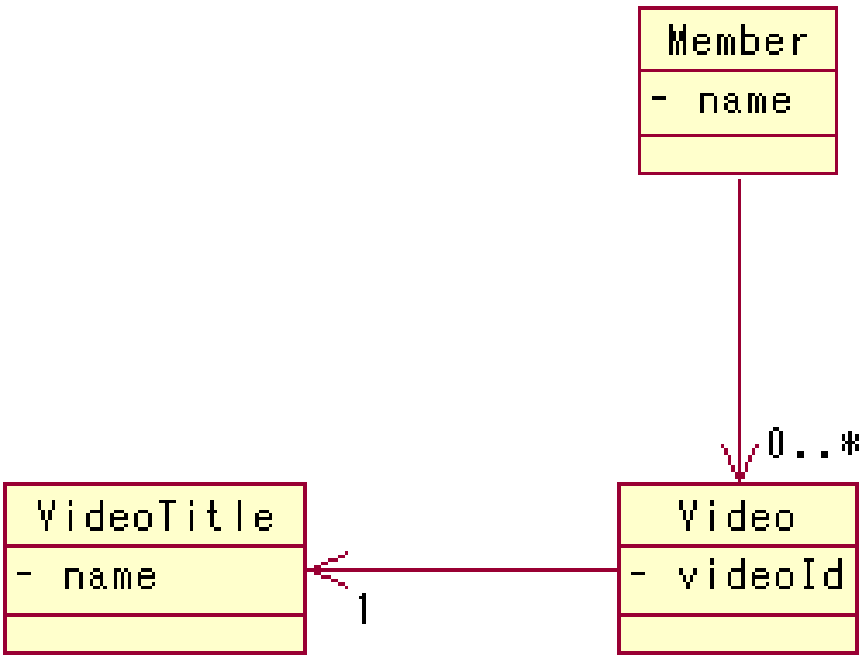

図 11.2: レンタルビデオ屋システムクラス図

11.4.2　練習問題 2

クラス図と参照モデルの変換を行い、片方があればいいというものではないということを確認します。

11.4.3　練習問題 3

ディレクトリの実装を通じて、再帰の考え方を確認します。

11.4.4　練習問題 4

連結リストの講義を飛ばした場合はここも飛ばしてください。連結リストの実装を通じて、再帰の考え方と連結リストのアルゴリズムを確認します。

第 12 章

継承を利用した抽象化

この章で学習すること

継承を使ったプログラムを書くことができる
継承を使ってクラスを抽象化することの利点を説明できる
JavaAPI をつかってプログラムを書くことができる

時間めやす

表 12.1: 時間めやす

12.1.1 節	40 分
12.1.2 節	10 分
12.2.1 節	30 分
12.2.2 節	30 分
12.2.3 節	10 分

12.1　クラスの抽象化

12.1.1　汎用的なリスト

学習目標

違うクラスのオブジェクトをひとつのものとして扱うためにクラスを抽象化するという考え方を理解する
継承をつかったプログラムが書ける

考えてみよう

出題意図: 違うクラスのオブジェクトをひとつのものとして扱うためにはそれらを抽象化したクラスが必要であることを理解する

メソッドの引数や戻り値の型がちがうために重複コードをまとめることができない

ここでは、Collection フレームワークを使うための複線として、自分で汎用 List クラスを設計するということを行います。

ここで、継承が出てきますが、変数やメソッドなどの、機能の継承については触れません。あくまで「同じ型として扱える」というのがこの章で扱う継承のメインで、空スーパークラスしか作成しません。

本カリキュラムでは、まだ本格的な継承の説明章が完成していません。ここで継承の簡単な説明をするのも手ですが、混乱すると思われるので、この章の練習問題が終ってからが良いでしょう。

12.1.2　オブジェクトと型

学習目標

抽象化してひとつのものとして扱っていた違うクラスのオブジェクトを再び具体的なオブジェクトとして扱うには型の変換が必要であることを理解する。
キャストをつかったプログラムが書ける

考えてみよう

出題意図: 型の変換が必要であることを理解する

変数の型と異なる値が代入ができないため

この節では、Collection フレームワークで使うための、オブジェクトのキャストについて議論します。基本的に使い方が分かればよいです。

12.2　コレクション API

12.2.1　JavaAPI とクラスライブラリ

学習目標

オブジェクト型とはどのようなものか理解する

JavaAPI の有用性を理解する

「API=Application Program Interface」アプリケーションプログラマーのために用意されているライブラリのこと。

ここから、import 文が必要なので注意。

12.2.2 コレクションフレームワーク

学習目標

コレクションフレームワークをつかったプログラムが書ける

ここでの継承の階層構造の説明はさらっと。

ArrayList だけ使えるように説明すればよい。連結リストをやった場合は LinkedList も触れる。

通常は、

```
List list = new ArrayList();
```

というプログラムを書くのが好ましいが、この段階では、継承を詳しくやらないので、

```
ArrayList list = new ArrayList();
```

というプログラムでよい。

12.2.3 JavaAPI ドキュメント

学習目標

API ドキュメントをつかって、必要な API を調べることができる

ArrayList, LinkedList などのドキュメントを読むと良い。あと、よく使っている、String クラスなども参考程度に触れる。

12.3 練習問題

練習問題 1

コレクション API を使う練習問題 (1)

練習問題 2

コレクション API を使う練習問題 (2)

LinkedList を使うので、11 章で連結リストを飛ばした場合は、やらなくてよい。