人にやさしいプログラミングの哲学

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

2003 年 5 月 8 日

目次

はじめに

はじめに

「人にやさしいプログラミングの哲学」は、初めてのプログラミングから、オブジェクト指向プログラミングの基礎までを学習するためのカリキュラムです。このカリキュラムは、単にプログラミング言語の文法や記法などの知識を得るのではなく、

> 実際にそれらの知識を使ったプログラムが書けるようになること
> 単に動くプログラムが書けるようになるだけでなく、使いやすく、メンテナンスのしやすいプログラムが書けるようになること
> 手続き指向やオブジェクト指向などの技法の背景にある「考え方」を捉えること

を目的としています。

プログラミングは文法などの「知識」だけを記憶しても実際に書けるようになりません。これは、例えば、野球のルールを知っていても、野球をプレーすることが出来ないのと同じ理由です。ルールはすぐに覚えることが出来ますが、実際に野球をプレーできるようになるためには、”いかに勝つか”といった視点で、基本的な戦術を学びながら、練習を積むことが必要です。このテキストでは、小規模ではありますが実際のアプリケーションを例題として用い、その開発過程をたどりながら、練習の方法を示していきますので、それに従ってプログラムを書く練習を積みましょう。

また、実際のプログラミング開発現場で求められるのは、単に動くプログラムを書けることだけでなく、使いやすく、メンテナンスのしやすいプログラムが書けることです。大規模なソフトウエアの開発現場では、複数の人が一緒に開発することが多いので、知識を共有するためには、プログラムの構造が整理された、人間が理解できるプログラムである必要があります。このテキストでは、そのようなプログラムのガイドラインを示していきますので、最初は真似をしながら、そしてそれを自分のプログラムに取り入れながら、自分のものとするように頑張ってください。

カリキュラムの取り組みかた

「考え方」を捉えよう

このカリキュラムの「哲学」という名前には、単にプログラミング言語の文法や記法などの知識を記憶するのではなく、背景にある「考え方」を身に付けるためのカリキュラムであるという意味がこめられています。

移り変わりが激しいのが IT 技術の特徴です。このテキストではプログラミング言語として Java を扱いますが、5 年後、10 年後も Java が最前線の技術として使われているとは言い切れません。しかし、背景にある「考え方」はそう簡単に変わるものではありません。例えば、オブジェクト指向手法は全く新しい手法ではなく、構造化手法の問題を改善するために考えられてきた手法です。その背景には、いかに人に理解できるプログラムを書くかという議論の積み重ねがあるのです。

こうした基本的な「考え方」は、新しい技術の習得を助けてくれます。このカリキュラムは、そうして身についた考え方を基礎として、技術を適切に適用、応用していくことが出来るように人材を育てることを目指しています。

それゆえ、このカリキュラムでは、新しい文法や記法などの知識が、「何故必要なのか?」「どのように使ったら効果的なのか?」という視点から議論をすすめます。是非よいプログラムを書くにはどうしたらよいか、という視点から、議論し、徐々に「考え方」や基本原理の理解を深めてください。

こうした理由から、テキストには”あえて”空欄を作ってあります。講義や議論を通して、空欄を自分なりに埋めていきましょう。最終的に自分が学習したプログラムの「考え方」のテキストが出来上がるはずです。

人にやさしいプログラムを目指そう

このテキストで学ぶソフトウエア作成技法の指針となっているのは、本テキストのタイトルともなっているように、「人にやさしいプログラムを書く」ということです。

「人にやさしいプログラム」の基本は、

> 意図した通りに動く（バグのない）プログラム
> 使う人にとって使いやすいプログラム
> 人に理解できるソースコードが書かれたプログラム

であることです。

この基本的な「考え方」はこのテキストの全てに貫かれた基本姿勢です。そのため、このカリキュラムでは、例え最初の小規模なプログラムでも、これらのことを意識して「人にやさしいプログラム」を示していきます。

テキストの構成

本テキストは 2 部構成になっています。

第一部　構造化プログラミング編　第一部では、人にやさしいプログラミングの基本を学習します。

プログラムの書法から、HCP チャートを用いた設計の技法、変数や手続きを用いた抽象化技法を使って、いかに人が理解できるプログラムを書くかという議論をします。

第二部　オブジェクト指向プログラミング編　さらに人にやさしいプログラミングを書く技法として、オブジェクト指向プログラミングに挑戦します。

オブジェクト指向技法の考え方自体はさほど難しくありませんが、重要なのは、どのようにオブジェクト指向の手法を使うかというところにあります。第二部では、オブジェクト指向手法を使って、いかに人が理解できるプログラムを書くかという議論をします。

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

1.1 人にやさしいソースコードとは

次のプログラム (リスト 1 とリスト 2) を比較してみましょう。

リスト 1: 足し算プログラム（一般的な教科書のスタイル）

```
 1: public class AddTwoNumberSample {
 2:
 3:   public static void main(String[] args) {
 4:     int a, b, c;
 5:     a = Input.getInt();
 6:     b = Input.getInt();
 7:     c = a + b;
 8:     System.out.println(c);
 9:   }
10:
11: }
```

リスト 2: 足し算プログラム（このテキストのスタイル）

```
 1: /**
 2:  * 足し算計算機アプリケーション
 3:  *
 4:  * ２つの数をキーボードから読み込み、足し算の結果を表示する
 5:  * ２数とも、０であったら、終了する
 6:  *
 7:  * @author Manabu Sugiura
 8:  * @version $Id: AddTwoNumberApplication.java,v 1.9 2003/05/20 12:24:47 duskin Exp $
 9:  */
10: public class AddTwoNumberApplication {
11:
12:   public static void main(String[] args) {
13:     AddTwoNumberApplication addTwoNumberApplication =
14:       new AddTwoNumberApplication();
15:     addTwoNumberApplication.main();
16:   }
17:
18:   void main() {
19:
20:     int firstNumber; //１番目に入力された整数
21:     int secondNumber; //２番目に入力された整数
22:     int result; //２つの整数の足し算の結果
23:
24:     //アプリケーションの説明をする
25:     System.out.println("２つの数の和を求めます。");
26:     System.out.println("（2 数に 0 を入力すると終了します）");
27:
28:     //１番目の数を入力する
29:     System.out.print("１つ目の整数を入力してください>>");
30:     System.out.flush();
31:     firstNumber = Input.getInt();
32:
33:     //２番目の数を入力する
34:     System.out.print("２つ目の整数を入力してください>>");
35:     System.out.flush();
36:     secondNumber = Input.getInt();
37:
38:     //加算を繰り返す
39:     while (firstNumber != 0 || secondNumber != 0) { //２数とも０なら終了コード
40:
41:       //計算する
42:       result = firstNumber + secondNumber;
43:
44:       //結果を表示する
45:       System.out.println("２つの数の和は" + result + "です。");
46:
47:       //１番目の数を入力する
48:       System.out.print("１つ目の整数を入力してください>>");
49:       System.out.flush();
50:       firstNumber = Input.getInt();
51:
52:       //２番目の数を入力する
53:       System.out.print("２つ目の整数を入力してください>>");
```

```
54:       System.out.flush();
55:       secondNumber = Input.getInt();
56:     }
57:
58:     //アプリケーションが終了したことを知らせる
59:     System.out.println("アプリケーションを終了しました。");
60:   }
61: }
```

どちらも同じことをするプログラムですが、見た目は大分異なりますね。

考えてみよう

気が付いたことを議論してみましょう。

1.2　人にやさしいソースコードの書法

次のプログラムを読んでみましょう。このプログラムがどのようなプログラムかが分かるでしょうか。答えは大方の人が No だと思います。(分かったとしても相当時間がかかったのではないでしょうか。)

これからこのプログラムを、人にやさしいソースコードに直していきましょう。

リスト 3: 例題プログラム

```
 1: public class Example {
 2:
 3:   public static void main(String[] args) {
 4:     Example example = new Example();
 5:     example.main();
 6:   }
 7:
 8: void main(){
 9: int a = 49;
10: int b = 73;
11: int c = 100;
12: int d = 45;
13: int e = 25;
14: double x = a + b + c + d + e;
15: double y = x / 5.0;
16: y = y * 10;
17: if((y % 10) >= 5){
18: y = y + 10;
19: }
20: int z = (int)(y / 10);
21: System.out.println(z);
22: }
23: }
```

オマジナイ

Java はオブジェクト指向言語であるがゆえに、どんなに簡単なプログラムでも、初心者にとって理解するのが難しいいくつかの記述が必要です。

そんな”オマジナイ”をここで確認しておきましょう。

package—, import— この記述は、テキスト作成上の都合で記述されているものです。実際のプログラムでは**記述しないで**下さい。

クラス宣言 Java のプログラムは全て「クラス」と呼ばれるプログラムの単位の中に記述されます。（詳しくはオブジェクト指向編で）ですから、どのような小さなプログラムでも必ず一つクラスを作る必要があります。

クラスの記法は次のとおりです。クラス名は自分で決められますが、他はオマジナイです。中括弧「{}」で囲まれたブロックの中にメソッドなどのプログラムの要素が記述されます。

```
public class [クラス名]{

  （ここにプログラムの要素を記述する）

}
```

また、Java のプログラムではファイル名に気をつける必要があります。ファイル名は必ず**クラス名に「.java」をつけたもの**にしなければなりません。

例えば、この例ですと、ファイル名は「Sample.java」にする必要があります。

public static void main(String args[]) メソッド クラスの中に、中括弧で囲まれた記述が２つあります。これをメソッドといいます。クラスの中には、いくつでもメソッドを記述することができますが、第 4 章でメソッドを詳しく学習するまで、2 つのメソッドでプログラムを記述します。

このメソッド記述は全てオマジナイです。次のような書式で記述してください。

```
public static void main(String[] args){
    [クラス名] [クラス名の先頭を小文字にした変数] = new [クラス名]();
    [クラス名の先頭を小文字にした変数].main();
}
```

void main() メソッド　2 つのメソッドのうち、こちらがアプリケーションプログラムを記述するメソッドです。プログラムは main() メソッドから始まる [1]ことを覚えておきましょう。書式は次のとおりで、中括弧の中にアプリケーションプログラム (行いたい仕事) を記述します。

```
void main(){

    （ここにアプリケーションプログラムを記述する）

}
```

[1] 本当は public static void main(String[] args) メソッドからプログラムが始まります。今回は public static void main(String[] args) から main() メソッドを呼び出し（メソッドに関しては 4 章以降を参照）ています。

1.2.1 意味のまとまりを意識する

ただプログラムの処理がだらだらと書かれているソースコードは、人にやさしくありません。処理の意味を考えて、いくつかのまとまりを作ると人間にとってわかりやすくなります。

プログラムにおいて、処理のまとまりのことをブロックと呼びます。ブロックが分かりやすいように記述するには、インデントと空行を効果的に使うことが重要です。例をリスト 4 に示します。

リスト 4: 例題プログラム（ブロックを意識した）

```
 1: public class Example {
 2:
 3:   public static void main(String[] args) {
 4:     Example example = new Example();
 5:     example.main();
 6:   }
 7:
 8:   void main() {
 9:
10:     int a = 49;
11:     int b = 73;
12:     int c = 100;
13:     int d = 45;
14:     int e = 25;
15:
16:     double x = a + b + c + d + e;
17:     double y = x / 5.0;
18:
19:     y = y * 10;
20:     if ((y % 10) >= 5) {
21:       y = y + 10;
22:     }
23:     int z = (int) (y / 10);
24:
25:     System.out.println(z);
26:   }
27: }
```

ブロック プログラムにおいて、処理のまとまりのことをブロック (図 1.1) と呼び、Java 言語では、中括弧「{}」で囲って表現します。（クラスやメソッドもブロックの一種です。）Java のプログラムはブロックの入れ子で構成されています。

インデント インデント (図 1.2) とは、字下げをすることです。字下げには TAB やスペースを使用します。（TAB なら 1 つ分、スペースなら 2 つ分を目安とするとよいでしょう）字下げをすることによって、入れ子になったブロックの関係をはっきり表すことができます。

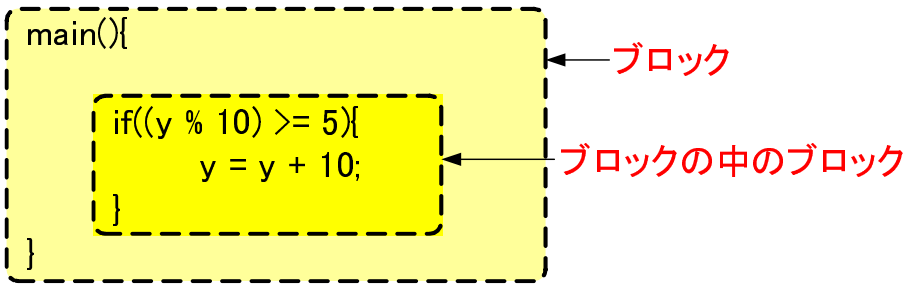


図 1.1: ブロック

```
main(){
    if((y % 10) >= 5){
        y = y + 10;
    }
}
```

図 1.2: インデント

空行で表現されたブロック　中括弧で囲まれたブロックだけでなく、空行を入れることにより、ブロックの中に意味のまとまりを作って、プログラムをより見やすくします。(図 1.3)

```
double x = a + b + c + d + e;
double y = x / 5.0;

y = y * 10;
if ((y % 10) >= 5) {
        y = y + 10;
}
int z = (int) (y / 10);

System.out.println(z);
```

図 1.3: 空行によるブロック

1.2.2 変数に適切な名前をつける

Java 言語では、クラスや変数、メソッドの名前は自由につけられます。では「a」や「bbb」のような名前がついていた場合、他の人が果たしてその意味を理解してくれるでしょうか。人に分かりやすいソースコードを書くためには、 クラスや変数の名前はその意味が分かるような名前にする必要があります。クラスや変数に名前をつけるときは、子供に名前を付ける気持ちで望みましょう。クラスや変数、メソッドについて後ほど詳しく説明します。

リスト 5: 成績管理プログラム（適切な変数名をつけた）

```
 1: public class ScoreAdministratorSample {
 2:
 3:   public static void main(String[] args) {
 4:     ScoreAdministratorSample scoreAdministratorSample =
 5:       new ScoreAdministratorSample();
 6:     scoreAdministratorSample.main();
 7:   }
 8:
 9:   void main() {
10:
11:     int japanese = 49;
12:     int mathematics = 73;
13:     int science = 100;
14:     int civics = 45;
15:     int english = 25;
16:
17:     double total = japanese + mathematics + science + civics + english;
18:     double average = total / 5.0;
19:
20:     average = average * 10;
21:     if ((average % 10) >= 5) {
22:       average = average + 10;
23:     }
24:     int result = (int) (average / 10);
25:
26:     System.out.println(result);
27:   }
28: }
```

1.2.2.1 Java の命名規則

Java 言語を記述するときの一般的な命名規則 [2]の説明をします。本テキストのソースコードはすべてこの規則に従っています。

[2] クラス名やメソッド名、変数名に関しては、数字が先頭にくるもの、予約語、演算子を含むものは使用できません。

クラス名

– 最初の文字は大文字に

– 複数の単語の時は各単語の最初の文字を大文字に

例えば、旅券予約アプリケーションには、「TicketReserveApplication」となります。クラス名とファイル名は同じにしなければならないので、この場合ファイル名は「TicketReserveApplication.java」です。

メソッド名

– 最初の文字は小文字に

– 複数の単語の時は各単語の最初の文字を大文字に

メソッドはある処理を行うものなので大抵は動詞から始めます。例えば、現在の時刻を調べるメソッドは、「getCurrentTime」 といったメソッド名になります。

変数名

– 最初の文字は小文字に

– 複数の単語の時は各単語の最初の文字を大文字に

変数名は、メソッド名と同様の規則です。例えば、苗字を記憶する変数名は、「familyName」 といった変数名になります。

1.2.3 コメントをつける

ソースコードの中にコメントを書き込むことができます。このコメントはプログラムの処理には全く影響しません。コメントは内容が重要です。内容については次項で詳しく議論します。とりあえずコメントをつけたプログラムをリスト 6 に示します。

リスト 6: 成績管理プログラム（とりあえずのコメントを付けたもの）

```
 1: public class ScoreAdministratorSample {
 2:
 3:   public static void main(String[] args) {
 4:     ScoreAdministratorSample scoreAdministratorSample =
 5:       new ScoreAdministratorSample();
 6:     scoreAdministratorSample.main();
 7:   }
 8:
 9:   void main() {
10:     //変数を宣言し値を代入する
11:     int japanese = 49; //整数型 japanese という変数
12:     int mathematics = 73; //整数型 mathematics という変数
13:     int science = 100; //整数型 science という変数
14:     int civics = 45; //整数型 civics という変数
15:     int english = 25; //整数型 english という変数
16:
17:     //合計を保存しておく変数 total を 5 で割り、変数 average に代入する
18:     double total = japanese + mathematics + science + civics + english;
19:     double average = total / 5.0;
20:
21:     //average を 10 倍し，10 で割ったあまりを調べる
22:     average = average * 10;
23:     if ((average % 10) >= 5) { //余りが 5 以上なら
24:       average = average + 10; //10 を加える
25:     }
26:     //結果を result に代入する
27:     int result = (int) (average / 10);
28:
29:     //変数 result の値を表示する
30:     System.out.println(result);
31:   }
32: }
```

1.2.3.1　Java でのコメントの記述方法

コメントの記述方法を間違えると、コンパイルエラーになるので注意しましょう。

範囲指定コメント　プログラム中、「/*」 から 「*/」までのすべての文字は、コメントとして扱われます。複数行でも構いません。

```
/* この中がコメント */
```

Java の慣習として（Javadoc というドキュメントを自動生成するため）始まりは「/**」とし、複数行にわたる場合は、行の始めに「*」をつけます。

```
/**
 * Java での標準形式
 */
```

行コメント　プログラム中、「//」からその行の最後までのすべての文字は、コメントとして扱われます。次の行までの効果はありません。

```
int x;//この行のここから先はすべてコメント
```

考えてみよう

先ほどの (リスト 6) を読んでみてこのプログラムが何をするものであるのか考えてみましょう

1.2.4 ひとにやさしいコメントの書法

1.2.4.1 プログラムの目的を明らかにするコメント

コメントはただ書けばよいというものではありません。
具体的に見てみましょう。次のコードは先のリスト 6 から抜き出したものです。

```
//average を 10 倍し，10 で割ったあまりを調べる
average = average * 10;
if ((average % 10) >= 5) { //余りが 5 以上なら
  average = average + 10; //10 を加える
}
```

このコメントを次のものと比べてみましょう。

```
//平均を四捨五入する
average = average * 10;
if ((average % 10) >= 5) { //1の位が5以上なら
  average = average + 10; //繰り上げる
}
int result = (int) (average / 10);
```

人にやさしいコメントをつけたプログラムをリスト 7 に示します。

リスト 7: 成績管理プログラム（人にやさしいコメント付）

```
 1: /**
 2:  * ×○中学校の成績管理プログラム
 3:  *
 4:  * 五教科（国語・数学・理科・公民・英語）の平均（四捨五入済み）を求める
 5:  *
 6:  * @author Manabu Sugiura
 7:  * @version $Id: ScoreAdministratorSample.java,v 1.11 2003/08/08 13:15:50 duskin Exp $
 8:  */
 9: public class ScoreAdministratorSample {
10:
11:   public static void main(String[] args) {
```

```
12:     ScoreAdministratorSample scoreAdministratorSample =
13:       new ScoreAdministratorSample();
14:     scoreAdministratorSample.main();
15:   }
16:
17:   void main() {
18:
19:     //各教科の点数を設定する
20:     int japanese = 49; //国語
21:     int mathematics = 73; //数学
22:     int science = 100; //理科
23:     int civics = 45; //公民
24:     int english = 25; //英語
25:
26:     //５教科の合計点を求める
27:     double total = japanese + mathematics + science + civics + english;
28:     //５教科の平均を求める
29:     double average = total / 5.0; //平均を計算する
30:
31:     //平均を四捨五入する
32:     average = average * 10;
33:     if ((average % 10) >= 5) { //１の位が５以上なら
34:       average = average + 10; //繰り上げる
35:     }
36:     int result = (int) (average / 10);
37:
38:     //四捨五入した平均を表示する
39:     System.out.println(result);
40:   }
41: }
```

1.2.4.2 コメントの種類

コメントは書く位置によって役割が異なります。次の3種類を目安にすると分かりやすいでしょう。

タイトル（見出し）コメント

タイトルコメントは、プログラムの一番初めに書くコメントで、そのプログラムが何を目的としたものであるかを簡潔に表記します。タイトルコメントの要素として以下のものが挙げられます。

標題 (タイトルとプログラム目的) プログラムの名称と、その機能を簡潔に表したもの。

ファイル名 そのプログラム自身のファイル名。

作者 プログラムを作成した人の名前。設計した人が別の人ならば、設計者の名前を書いておきます。後からそのプログラムを変更する人が、誰に尋ねれば情報を得られるのかを簡単に知ることができます。複数の人でプログラムを書くときに役に立ちます。

バージョンアップの履歴　改造や修正によっていったん出来上がったプログラムが変更されると、そのプログラムのバージョンが変わることになります。このバージョンアップが行われた日付と、行った人の名前を記入しておくと、誰によっていつ変更されたものなのかが分かります。

ブロックコメント

対象ブロックが何を目的としたプログラムなのかを記述するコメントです。(ブロックが始まる前に書く）ブロック全体に通用します。(図 1.4)

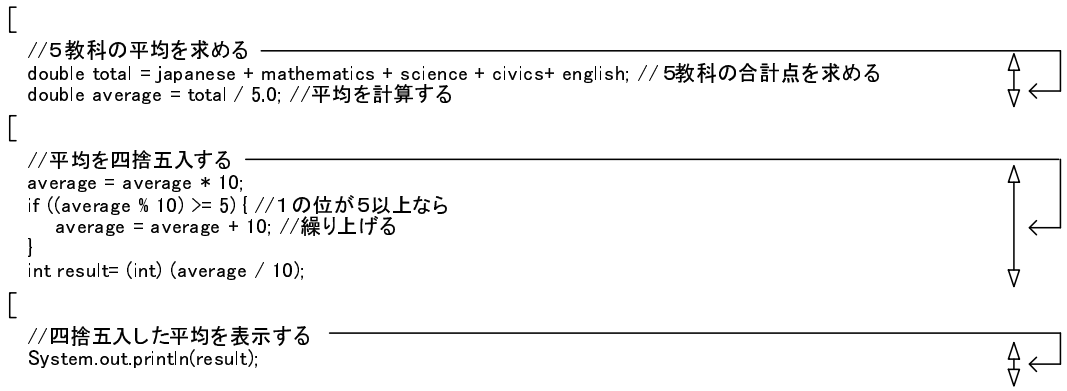


図 1.4: ブロックコメントの有効範囲

行コメント

その行は何を目的として書かれているのかを行末に書くコメントです。その行だけに通用する小目的を書きます。

1.3　プログラムの目的と階層構造

1.3.1　目的の階層構造

前節で学習したのは、コメントにはプログラムの手段ではなく、目的を書くことが重要ということでした。先ほど紹介してきた「成績を管理する（５教科の平均を求め四捨五入をする)」というプログラムの目的を、さらに細かく見ていきましょう。

各ブロックに記載されているコメントを抜き出すと次のようにまとめられます。

５教科の平均（四捨五入済み）を求める
- 各教科の点数を設定する
- ５教科の平均を求める
- 平均を四捨五入する
- 四捨五入した平均を表示する

このように小さなプログラムでも目的を階層化し、構造を整理することができます。このように整理されたプログラムは、どのようなプログラムであるかが、プログラムを読めない人でさえ一目瞭然です。

1.3.2　HCP チャートによるプログラムの設計

プログラムのソースコードを実際に書き始める前に、これから作るプログラムの構造を分析、整理しまとめることを、プログラムの設計といいます。

この「人にやさしいプログラミング哲学」の前半では、NTT で開発された、「HCP チャート (Hierarchical and Compact Description Chart : 階層化コンパクトチャート)」を用いて設計を行います。

HCP チャートの特徴は、プログラムの目的の階層構造を図で表現できることです。また、プログラムでよく出てくる繰り返しの処理や、条件によって処理を変更するなどといった、プログラムの処理の構造を簡単に書くことができる記号も用意されています。

例として、成績管理プログラムにおける HCP チャートを図 1.5 に示します。

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。

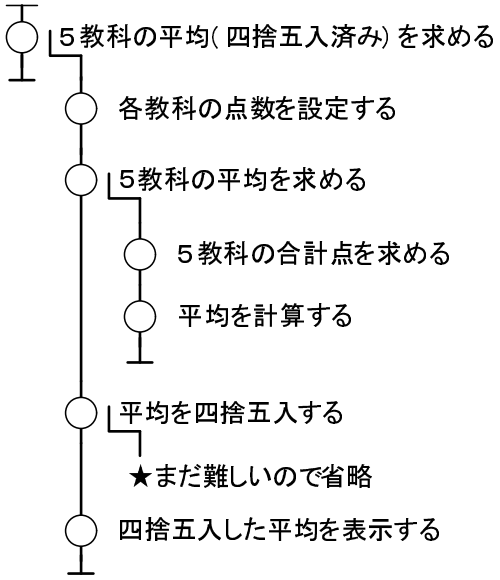


図 1.5: 成績管理プログラムの HCP チャート

1.3.3 HCP チャートを反映したソースコードの記述

HCP チャート (図 1.5) と、最終的に人にやさしいソースコードになったソースコード (リスト 7) を見比べてみましょう。

HCP チャートを描くとプログラムの目的の階層構造がきれいに整理されます。あとはこの階層構造にあわせて、それぞれの目的を実現するようにソースコードを書けば、自然に人にやさしいプログラムを書くことができます。

設計が終ったら、まずソースコードにコメントとして HCP チャートに書かれた目的を書いてしまいましょう。**ソースコードはコメントから書く**のが人にやさしいプログラマーへの道です。

1.4　人にやさしいユーザインターフェイス

1.4.1　ユーザインターフェイスの基本

ソフトウエアは人が使うために、そして人の助けをするために開発されます。本節では、いかに” 使う人” にやさしいプログラムを書くかということを考えていきたいと思います。

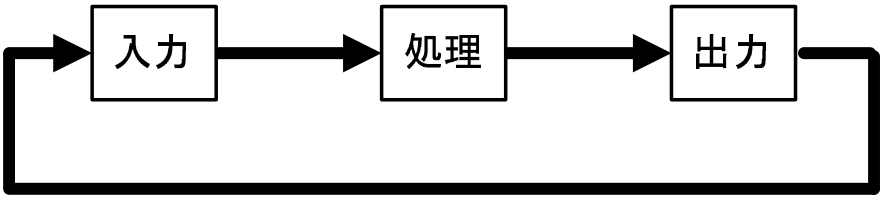


図 1.6: ユーザインターフェイスの基本

ユーザインターフェイスを備えたプログラムの基本構造は、図 1.6 に示すように、入力、処理、出力の繰り返しです。入力はユーザの入力、出力は処理結果の表示と考えるとより分かりやすいと思います。この構造は、今日広く使われている GUI でも、本テキストで扱っていく CUI でも変わりません。

この基本構造のうち、プログラムの使い手（ユーザ）がプログラムに接する部分、つまりコンピュータの処理以外の入力、出力の個所のことをユーザインターフェイスと呼び、プログラムの使い勝手のことを表します。本テキストでは、ここからほとんど全ての例題（–Application と名前がついているもの）について、使いやすい、人にやさしいユーザインターフェイスを備えたプログラムを示すようにしています。

本テキストではじめてのユーザインターフェイスを備えたプログラムの HCP を図 1.7 に、ソースをリスト 8 に示します。

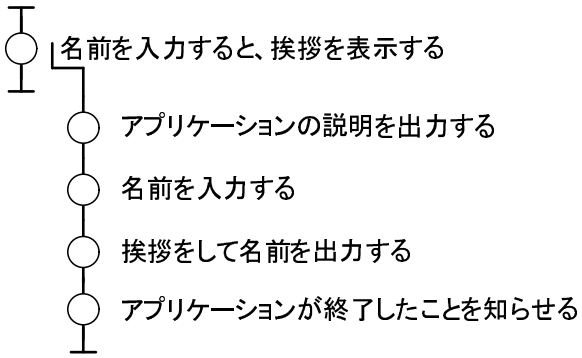


図 1.7: 名前を入力すると挨拶を表示するアプリケーションの HCP チャート

リスト 8: 名前を入力すると挨拶を表示するアプリケーション

```
 1: /**
 2:  * 自分の名前をキーボードから入力し、挨拶を表示するアプリケーション
 3:  *
 4:  * @author Manabu Sugiura
 5:  * @version $Id: GreetingApplication.java,v 1.3 2003/05/04 16:25:14 macchan Exp $
 6:  */
 7: public class GreetingApplication {
 8:
 9:   public static void main(String[] args) {
10:     GreetingApplication greetingApplication = new GreetingApplication();
11:     greetingApplication.main();
12:   }
13:
14:   void main() {
15:
16:     String name; //表示する名前
17:
18:     //アプリケーションの説明を出力する
19:     System.out.println("名前を入力すると、挨拶を表示します。");
20:
21:     //名前を入力する
22:     System.out.println("名前を入力してください");
23:     System.out.print(">>");
24:     System.out.flush();
25:     name = Input.getString();
26:
27:     //挨拶をして名前を出力する
28:     System.out.println("こんにちは。" + name + "さん。");
29:     System.out.println("これから長いお付き合いになりますね。");
30:
31:     //アプリケーションが終了したことを知らせる
32:     System.out.println("アプリケーションが終了しました。");
33:   }
34: }
```

人にやさしいユーザインターフェイスにするために、ここでは 2 つのことに気をつけています。

タイトル プログラムを開始したときに、そのプログラムがいったい何をするものであるかが分かると、使う人は、次に何をしてよいか、どのように使ったらよいかが分かりやすくなります。そのため、プログラムの開始と同時に、それがどのようなプログラムなのかということをタイトルとして表示します。これが「人にやさしいプログラム」です。

プロンプト プログラムが実行して、何か入力して欲しいと（あなたが勝手に）思っていても、あなた以外の他人には、何を入力してほしいか分かりません。このため、何を入力するかというのを表示してあげなければなりません。これが「プロの道」です。

1.4.2　繰り返し

先ほどの名前と挨拶を表示するアプリケーションでは、一度実行されただけで終了するものでした。これではいろいろな名前を入れて試してみたいときはいちいちプログラムを起動しないといけません。

これは電卓を使う際、一つ計算をするたびに機械が終了してしまい、次の計算をするときはまたスイッチを入れ直さないといけないことと同じことです。

この不便を解消するために繰り返しを用いてプログラムを書き直したのがリスト 9 です。

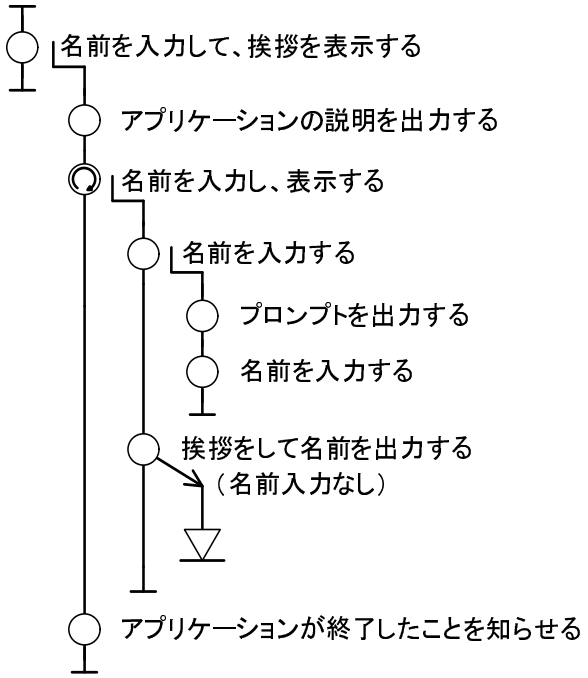


図 1.8: 名前を入力すると挨拶を表示するアプリケーションの HCP チャート

リスト 9: 名前を入力すると挨拶を表示するアプリケーション（繰り返しを導入）

```
 1: /**
 2:  * 名前をキーボードから入力し、挨拶を表示することを繰り返し行うアプリケーション
 3:  *
 4:  * なにも入力せずに改行するとアプリケーションを終了する
 5:  *
 6:  * @author Manabu Sugiura
 7:  * @version $Id: GreetingApplication.java,v 1.5 2003/05/04 20:30:23 gackt Exp $
 8:  */
 9: public class GreetingApplication {
10:
11:   public static void main(String[] args) {
12:     GreetingApplication greetingApplication = new GreetingApplication();
13:     greetingApplication.main();
14:   }
15:
16:   void main() {
17:
18:     String name; //表示する名前
19:
20:     //アプリケーションの説明を出力する
21:     System.out.println("名前をすると、挨拶を表示します。名前が空文字の場合は終了します。");
22:
23:     //名前を入力する
24:     System.out.println("名前を入力してください");
25:     System.out.print(">>");
26:     System.out.flush();
27:     name = Input.getString();
28:
29:     //名前を入力し、表示する
30:     while (name.length() != 0) {
31:
32:       //挨拶をして名前を出力する
33:       System.out.println("こんにちは。" + name + "さん。");
34:       System.out.println("これから長いお付き合いになりますね。");
35:
36:       //次の名前を入力する
37:       System.out.println("名前を入力してください");
38:       System.out.print(">>");
39:       System.out.flush();
40:       name = Input.getString();
41:     }
42:
43:     //アプリケーションが終了したことを知らせる
44:     System.out.println("アプリケーションが終了しました。");
45:   }
46: }
```

1.4.2.1　while 文

while 文は繰り返しを行う場合に用います。
while 文の書式を次に示します。

```
while ( [継続条件] ) {

  (繰り返す処理)

}
```

継続条件を調べて、合致すれば、中括弧内に書かれた処理を行い、また継続条件を調べるということを繰り返し、継続条件が合致しなくなるまで、処理を繰り返します。この制御の流れをフローチャートで示したものが、図 1.9 です。また、HCP チャートで書くと図 1.10 のようになります。

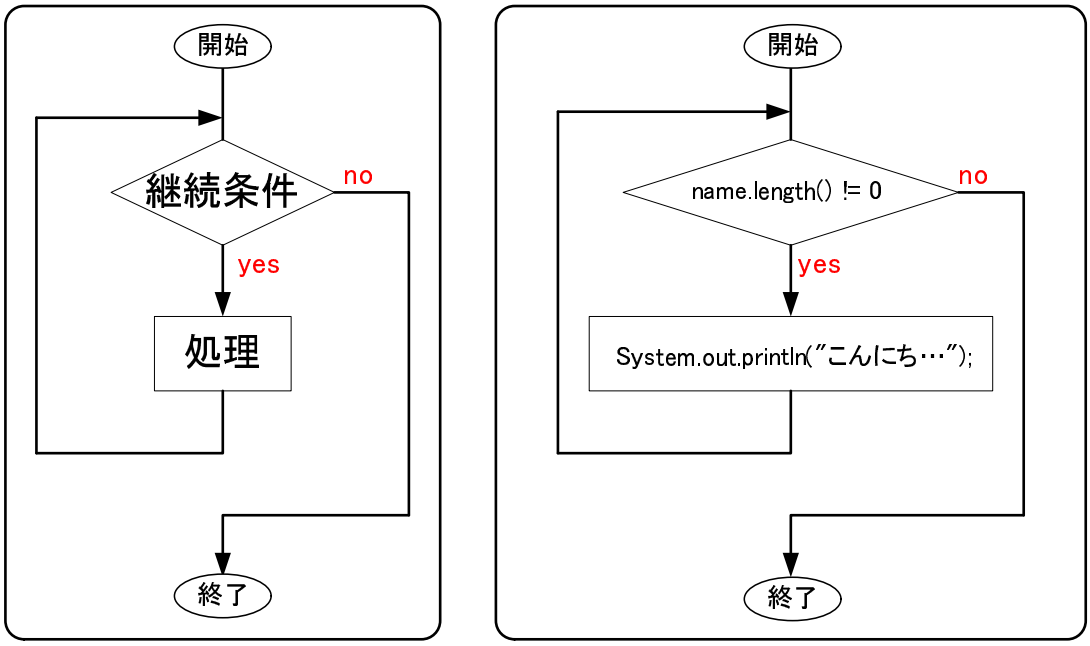


図 1.9: while 文のフローチャート

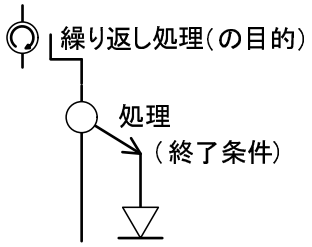


図 1.10: 繰り返しの HCP チャート

1.4.3 不正な入力の処理

ユーザは必ずしもプログラムを作った人が期待するような入力だけをするとは限りません。ユーザの不正な入力などによってプログラムに予期しない命令が入力されることがあります。

人にやさしいプログラムは、そのような場合のことも考慮されている必要があります。最悪でも、止まってしまうようなプログラムは避けたいものですね。

割り算を行うアプリケーション (リスト 10) を見てみましょう。割り算では割る数に 0 が入ってしまうと計算ができません。このプログラムでは、これを防ぐために次のような工夫がされています。

> 2 番目の数字の入力プロンプトで、0 以外の数字を入れるように予めユーザに注意させている。
> もし割る数に 0 が入力された場合は、計算を行わず、ユーザに 0 では割り算ができないことを伝える。

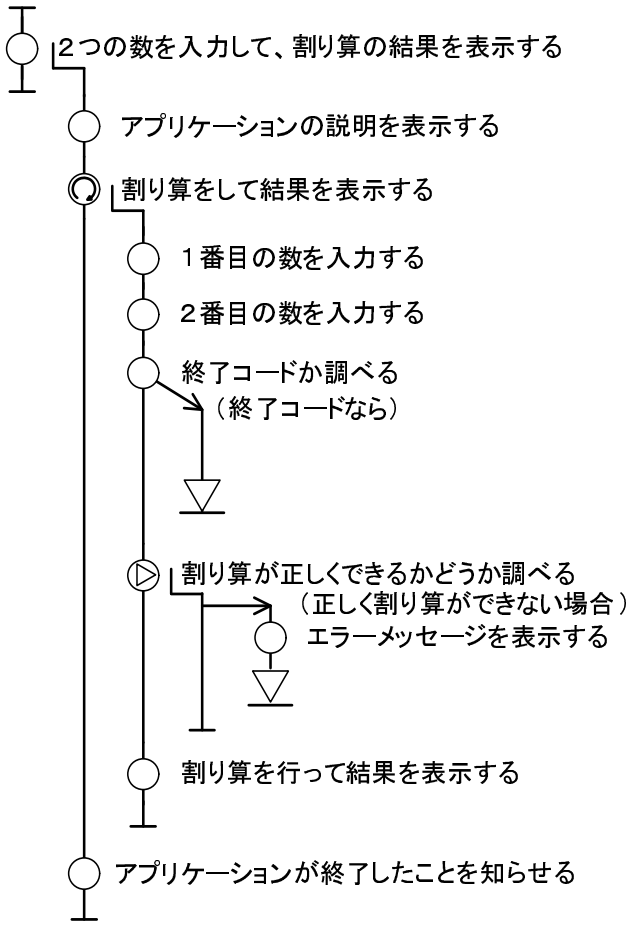


図 1.11: 割り算を行うプログラムの HCP チャート

リスト 10: 割り算を行うアプリケーション

```
 1: /**
 2:  * 割り算計算機アプリケーション
 3:  *
 4:  * ２つの数を入力すると、割り算の結果を出力する
 5:  * 入力が両方０の場合、アプリケーションを終了する
 6:  *（エラーチェックバージョン）
 7:  *
 8:  * @author Manabu Sugiura
 9:  * @version $Id: DivideTwoNumberApplication.java,v 1.10 2003/05/04 20:29:51 gackt Exp $
10:  */
11: public class DivideTwoNumberApplication {
12:
13:   public static void main(String[] args) {
14:     DivideTwoNumberApplication divideTwoNumberApplication =
15:       new DivideTwoNumberApplication();
16:     divideTwoNumberApplication.main();
17:   }
18:
19:   void main() {
```

```
20:
21:     int firstNumber; //１番目の整数
22:     int secondNumber; //２番目の整数
23:     int result; //割り算の結果
24:
25:     //アプリケーションの説明をする
26:     System.out.println("２つの数の割り算の結果を求めます。");
27:     System.out.println("（少数点以下は計算できません）");
28:
29:     //１番目の数を入力する
30:     System.out.println("１つ目の整数を入力してください");
31:     System.out.print(">>");
32:     System.out.flush();
33:     firstNumber = Input.getInt();
34:
35:     //２番目の数を入力する
36:     System.out.println("２つ目の整数を入力してください（割る数は０以外）");
37:     System.out.print(">>");
38:     System.out.flush();
39:     secondNumber = Input.getInt();
40:
41:     //割り算をして結果を表示する
42:     while (firstNumber != 0 || secondNumber != 0) {
43:
44:       //割り算が正しくできるかどうか調べる
45:       if (secondNumber == 0) { //正しく割り算ができない場合
46:         System.out.println("エラー：０で割り算はできません!");
47:         break; //アプリケーションを終了する
48:       }
49:
50:       //割り算を行って結果を表示する
51:       result = firstNumber / secondNumber;
52:       System.out.println("２つの数の商は" + result + "です");
53:
54:       //１番目の数を入力する
55:       System.out.println("１つ目の整数を入力してください");
56:       System.out.print(">>");
57:       System.out.flush();
58:       firstNumber = Input.getInt();
59:
60:       //２番目の数を入力する
61:       System.out.println("２つ目の整数を入力してください（割る数は０以外）");
62:       System.out.print(">>");
63:       System.out.flush();
64:       secondNumber = Input.getInt();
65:     }
66:
67:     //アプリケーションが終了したことを知らせる
68:     System.out.println("アプリケーションが終了しました。");
69:   }
70: }
```

1.5　練習問題

練習問題 1

自分用のタイトルコメントのテンプレートを作ってください。練習問題 2 からそれを使ってください。

練習問題 2

健康を管理するプログラム（HealthCareSample.java）を「人にやさしいプログラム」にしてください。

練習問題 3

足し算計算機アプリケーション（AddTwoNumberApplication.java）に、非常に大きな数字や、数字以外の文字などを入力し、どうなるか調べてみましょう。

第 2 章

変数を使った抽象化 (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

2.1　変数と値

2.1.1　データをどう扱うか

この章ではじゃんけんができるゲーム（アプリケーション）を作ろうと思います。交互に互いの手を入力しあって、じゃんけんができるようなアプリケーションです。HCP チャートを使って大まかな仕様を考えてみました。(図 2.1)

目的の階層構造は固まりましたが、問題となるのは、データ（グー、チョキ、パーの手）をどうコンピュータで扱えるようにするかです。コンピュータでは、そのままこの概念を表すことができないので、ここでは、1,2,3 の数値で表すという方法で実現することを考えましょう。(図 2.2)

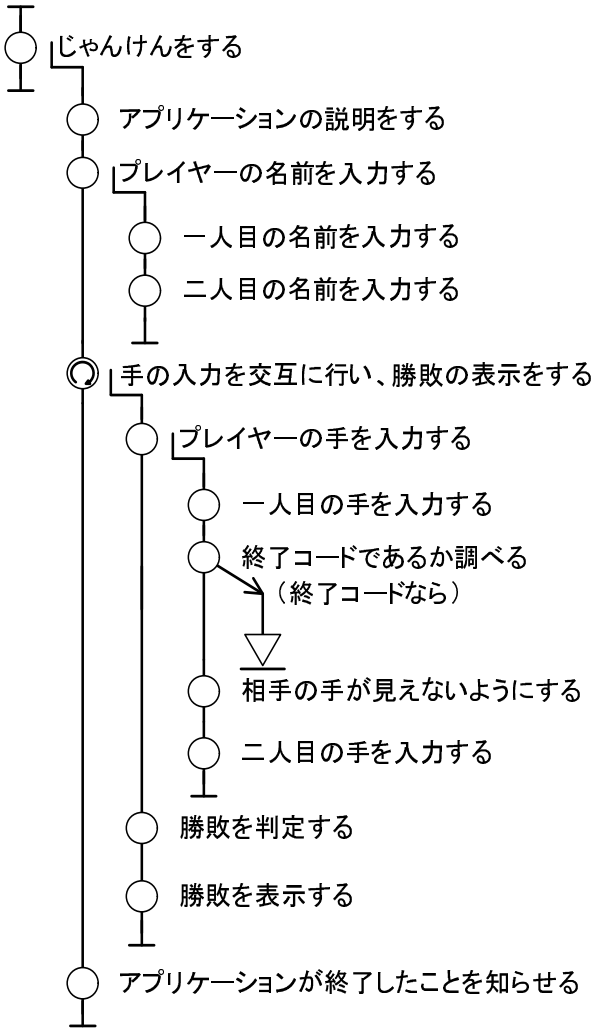


図 2.1: じゃんけんアプリケーションの HCP チャート

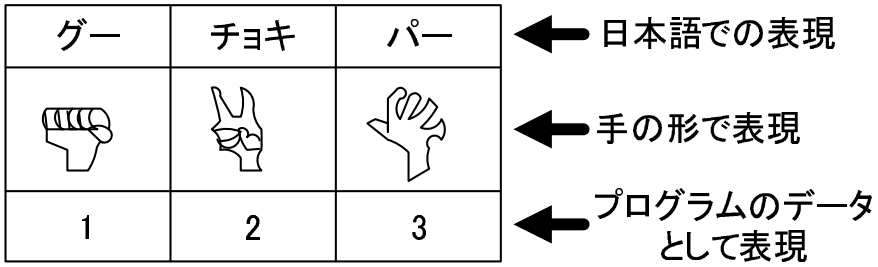


図 2.2: グー、チョキ、パーの表し方

このように、新しいプログラムを作成するためには、データをコンピュータが扱う形式に変換するという作業が必要になります。この作業のことを「データ構造」の設計といいます。

これに対して、第 1 章で学習した、手順（目的）の階層構造を設計することを「アルゴリズム」の設計といいます。「アルゴリズムとデータ構造」の 2 つが、プログラムを構成する重要な要素です。本章では、データ構造の設計を議論していきましょう。

じゃんけんアプリケーションでは、グー、チョキ、パーをそれぞれ 1,2,3 で表現することにして、HCP チャートにデータの表を書き足しました（図 2.3)。実装したものが、リスト 11 です。

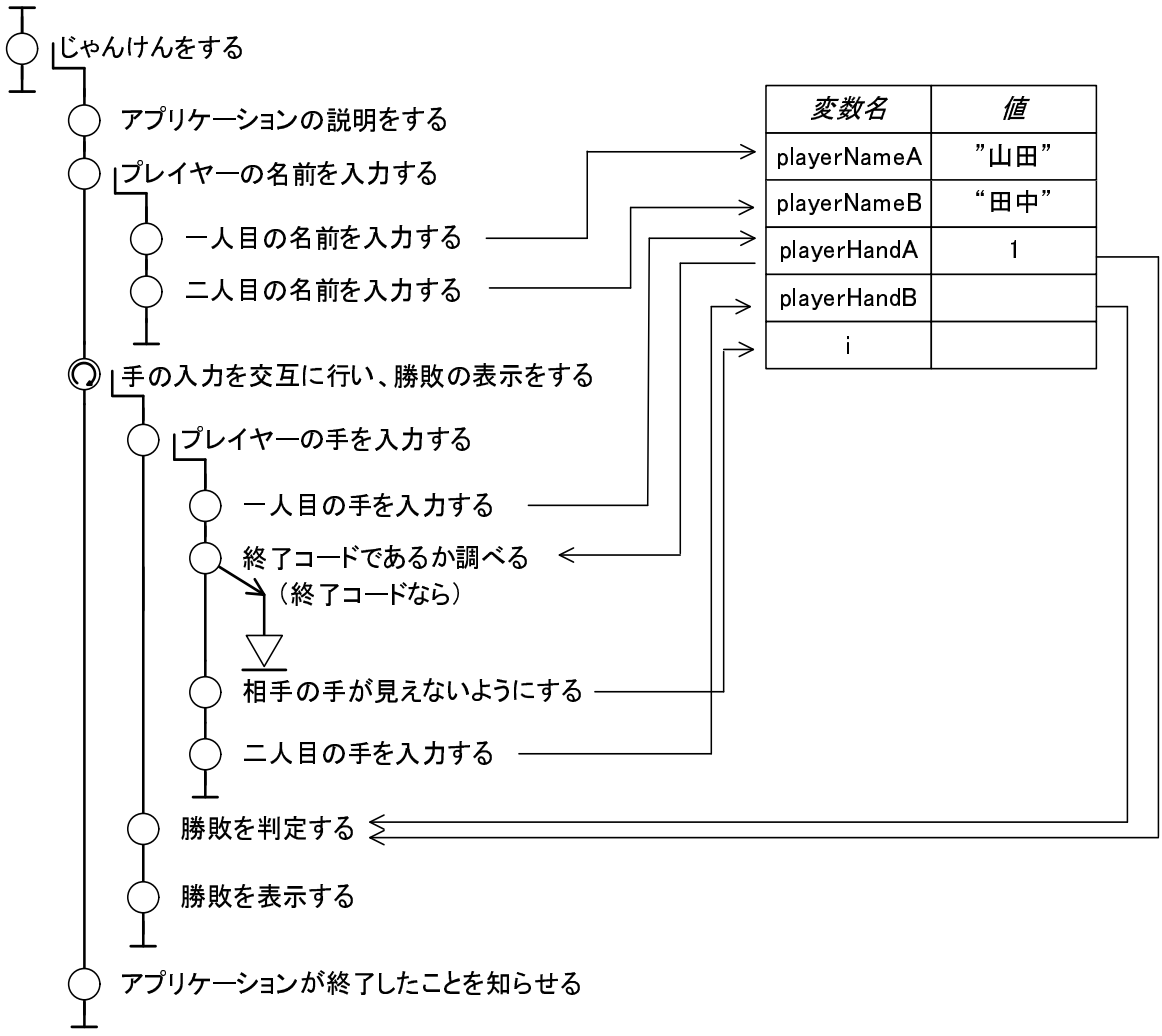


図 2.3: じゃんけんアプリケーションの HCP チャート（データあり）

リスト 11: じゃんけんアプリケーション

```
 1: /**
 2:  * じゃんけんアプリケーション
 3:  *
 4:  * 二人のプレイヤーがじゃんけんの手を交互に入力すると、勝敗を調べて、
 5:  * 勝者の名前を表示する
 6:  *
 7:  * @author macchan
 8:  * @version $Id: JankenApplication.java,v 1.11 2003/05/04 20:54:13 gackt Exp $
 9:  */
10: public class JankenApplication {
11:
12:   public static void main(String[] args) {
13:     JankenApplication jankenApplication = new JankenApplication();
14:     jankenApplication.main();
15:   }
16:
17:   void main() {
18:
19:     String playerNameA; //プレイヤー A の名前
20:     String playerNameB; //プレイヤー B の名前
21:     int playerHandA; //プレイヤー A の出した手
22:     int playerHandB; //プレイヤー B の出した手
23:     int i; //ループ用
24:
25:     //アプリケーションの説明をする
26:     System.out.println("じゃんけん♪アプリケーション");
27:     System.out.println("（プレイヤー A の手に 0 を入力すると終了します）");
28:
29:     //プレイヤー A の名前を入力する
30:     System.out.println("プレイヤー A の名前を入力してください");
31:     System.out.print(">>");
32:     System.out.flush();
33:     playerNameA = Input.getString();
34:
35:     //プレイヤー B の名前を入力する
36:     System.out.println("プレイヤー B の名前を入力してください");
37:     System.out.print(">>");
38:     System.out.flush();
39:     playerNameB = Input.getString();
40:
41:     //手の入力を交互に行い、勝敗の表示をする
42:     while (true) {
43:
44:       //プレイヤー A の手を入力する
45:       System.out.println(playerNameA + "さんの手を入力してください");
46:       System.out.println("1. グー, 2. チョキ, 3. パー (0. 終了)");
47:       System.out.print(">>");
48:       System.out.flush();
49:       playerHandA = Input.getInt();
50:
51:       //終了コードであるか調べる
52:       if (playerHandA == 0) { //終了コードなら
53:         break; //アプリケーションを終了する
```

```
54:       }
55:
56:       //相手の手が見えないようにする
57:       i = 0;
58:       while (i < 100) {
59:         System.out.println();
60:         i = i + 1;
61:       }
62:
63:       //プレイヤー B の手を入力する
64:       System.out.println(playerNameB + "さんの手を入力してください");
65:       System.out.println("1. グー, 2. チョキ, 3. パー");
66:       System.out.print(">>");
67:       System.out.flush();
68:       playerHandB = Input.getInt();
69:
70:       //勝敗を判定して、結果を表示する
71:       if (playerHandA == 1 && playerHandB == 2) { //グー VS チョキ
72:         System.out.println(playerNameA + "さんのかち");
73:       } else if (playerHandA == 1 && playerHandB == 3) { //グー VS パー
74:         System.out.println(playerNameB + "さんのかち");
75:       } else if (playerHandA == 2 && playerHandB == 1) { //チョキ VS グー
76:         System.out.println(playerNameB + "さんのかち");
77:       } else if (playerHandA == 2 && playerHandB == 3) { //チョキ VS パー
78:         System.out.println(playerNameA + "さんのかち");
79:       } else if (playerHandA == 3 && playerHandB == 1) { //パー VS グー
80:         System.out.println(playerNameA + "さんのかち");
81:       } else if (playerHandA == 3 && playerHandB == 2) { //パー VS チョキ
82:         System.out.println(playerNameB + "さんのかち");
83:       } else if (playerHandA == playerHandB) { //あいこ
84:         System.out.println("あいこでした");
85:       } else { //不正な入力の処理
86:         System.out.println("不正な入力です");
87:       }
88:     }
89:
90:     //アプリケーションが終了したことを知らせる
91:     System.out.println("アプリケーションが終了しました。");
92:   }
93: }
```

2.1.2 変数と値

リスト 11 の playerHandA と playerHandB に注目してみます。これにグーやチョキやパーといったじゃんけんの「手」のデータが入ります。このような具体的なデータのことを「値」と呼びます。値には、1,2,3 といった整数、0.05 のような実数、”山田”のような文字などの種類があります。

これに対して、具体的な値を格納する場所のことを「変数」といいます。具体的な値 (例えば 1) に対して、意味のある名前 (例えば playerHand) が結び付いていることを想像してみましょう。

本テキストでは、その様子を図 2.4 のような表を使って表していきます。一般的に「変数表」と呼ばれるものですが、このテキストでは「表モデル [1]」と呼んでいます。

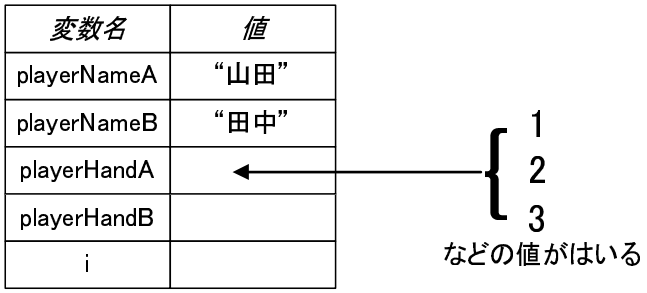


図 2.4: 変数と値の表モデル

[1] 先に書かれたデータ構造付き HCP チャート (図 2.3) では、データの表現として、表モデルを使用しています。これは、HCP チャートの正しい書きかたではありませんが、利便性のため、このテキストでは全ての HCP チャート上のデータを表モデルで記述しています。

2.1.3 プログラムの実行と変数の評価

プログラムが実行されるとデータがどのように移り変わっていくのかを、表モデルを利用して見ていきましょう。

2.1.3.1 変数の宣言

変数は宣言しないと使えません。変数の宣言は「型　変数名」という書式で記述します。変数を宣言すると、表モデルの左側の列に変数名が書き込まれます（図 2.5)。

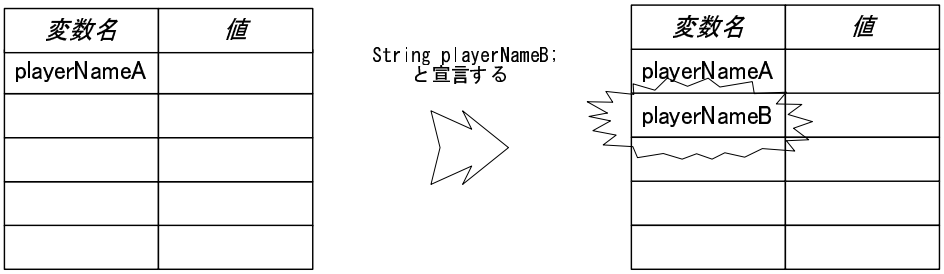


図 2.5: 宣言の表モデル

2.1.3.2 代入

宣言した変数には具体的な値を対応付けて使うことになります。この具体的な値を対応付けることを代入といいます。代入は「=」演算子を使います。数学の「=」とは違って、右から左に代入するという意味になります。「じゃんけんアプリケーション」では一人目のプレイヤーの名前が入力し終わると、図 2.6 のような表の状態になっています。

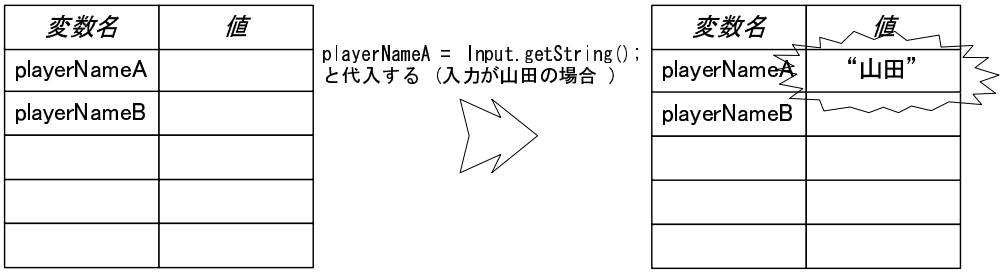


図 2.6: 代入の表モデル

2.1.3.3　変数の評価

宣言した変数は使用される際に、代入されている具体的な値との置き換えが起こります (図 2.7)。この置き換えのことを変数の「評価」といいます。

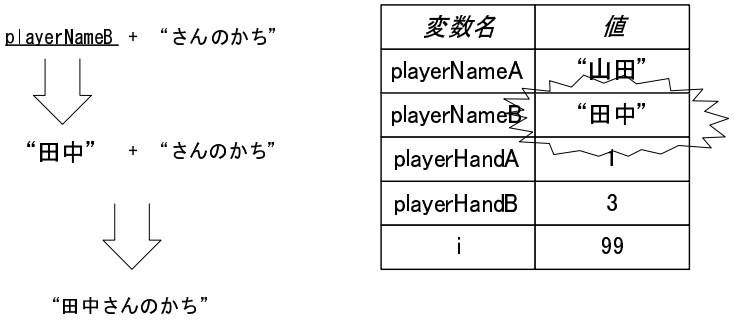


図 2.7: 評価の表モデル

2.1.3.4　プログラムの実行と表の変化

プログラムの実行に伴って、図 2.8 のように、表が変化していきます。

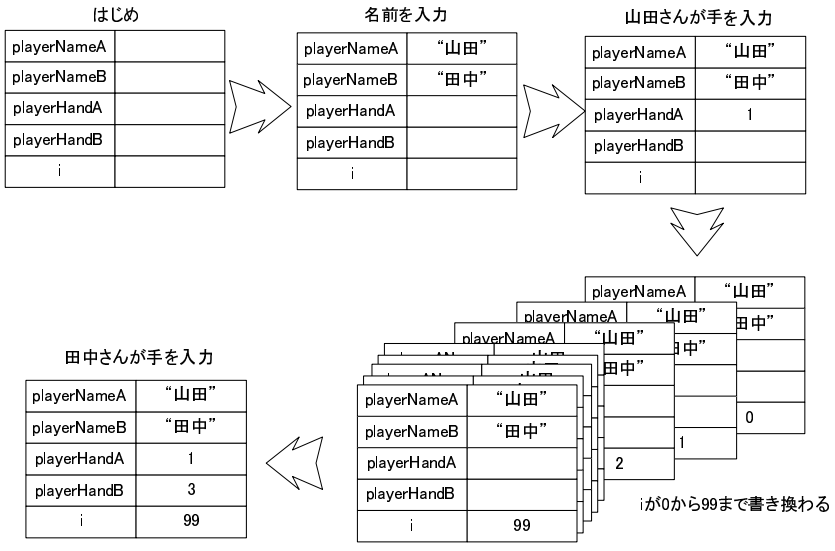


図 2.8: プログラムの実行と表モデルの変化

2.2　式と評価

2.2.1　式と値

評価はこれまで説明したように変数に限って行われるものではなく、「i ＋ 1」のような記述の実行でも行われ、評価されて値に置き換えられます（図 2.9）。このようにプログラム上で評価を行う対象になりうるものを**式**と呼びます。

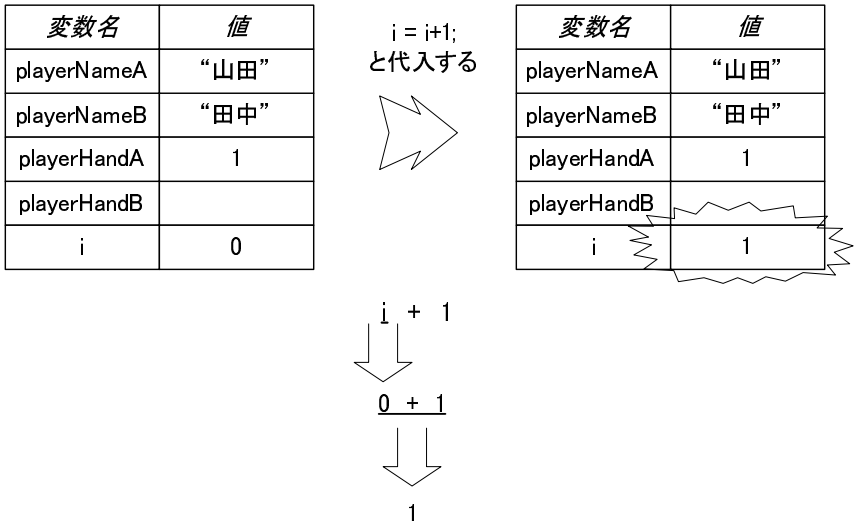


図 2.9: 式の評価の表モデル

2.2.2　条件式の評価

2.2.2.1　if 文

「じゃんけんアプリケーション」では、プレイヤーがそれぞれ出した手によって、じゃんけんの勝敗を判別して、結果を表示させることが必要でした。このように、条件に応じてプログラムが処理する内容を変えたいというときは if 文を用います。

```
if ( [条件式] ) {
  文  //条件式を評価して成立（true）のときはここを実行
} else {
  文  //条件式を評価して不成立（false）のときはここを実行
}
```

if 文は上のような文法で、条件式が成立すればその後の文を実行し、不成立の時は、else 節を実行します。else 節は省略可能で、その場合不成立の時には何も実行しません。

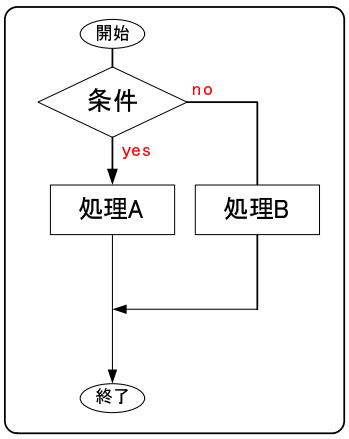


図 2.10: if 文のフローチャート

制御の流れは図 2.10 のようになります。

3 つ以上の条件で場合分けしたい時は下記のように、if else 文を使ってさらに条件分岐を増やします。

```
if ( [条件式 1] ) {
  文  //条件式 1 を評価して成立 (true) のときはここを実行
} else if( [条件式 2] ){
  文  //条件式 2 を評価して成立 (true) のときはここを実行
} else {
  文  //条件式 1 も条件式 2 を評価しても不成立 (false) のときはここを実行
}
```

HCP チャートで条件分岐を表現する際は図 2.11 のように記述します。今後、プログラムで条件分岐を使うようになるので、覚えておくと便利です。

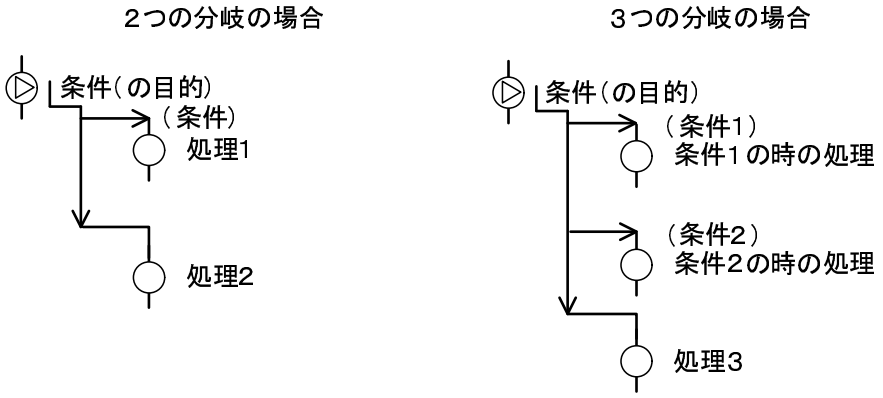


図 2.11: if 文の HCP

2.2.2.2 真偽値型と演算子

真偽値型

Java での条件式は、すべて真偽値型 (boolean) で判定されます。真偽値型は評価すると true か false のどちらかの値が入るデータの型です。while 文の継続条件も if 文の条件文も、評価すれば値は必ず true か false の値になります。

比較演算子と論理演算子

表 2.1: Java における数値の比較演算子（A,B は同じ型の変数とする）

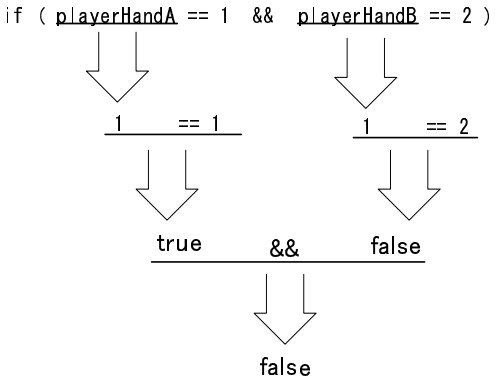
演算子	表記例	評価
==	A==B	A と B が同じ値の時 true
!=	A!=B	A と B が同じ値でない時 true
>	A>B	A が B より大きい時 true
<	A<B	A が B より小さい時 true
>=	A>=B	A が B 以上の時 true
<=	A<=B	A が B 以下の時 true

表 2.2: Java における論理演算子（A,B は真偽値型）

演算子	表記例	意味
&&	A&&B	A と B 両方とも true の時 true
||	A||B	A もしくは B が true の時 true

2.2.2.3　条件式の評価

if 文を考える際にも「評価」の考え方を使えば簡単です。例えば、if(playerHandA==1 && playerHandB==2) という式は、図 2.12 のように評価が行われます。これまでの式の評価とまったく同じです。



playerNameA	“山田”
playerNameB	“田中”
playerHandA	1
playerHandB	2
i	99

図 2.12: 条件式の評価

2.3 変数の型

2.3.1 変数の型

2.1.3.1 節で変数を宣言する時には、変数の「型」を指定する必要があると説明しました。これは、Java では、変数の種類によって、それぞれ記憶できるデータの種類も決まっているからです。

例えば、整数型 (int) なら-1, 0, 1, 2, 3 のといった整数の値が代入できますし、文字列型 (String) なら”山田”, ”田中”のといった文字列の値が代入できます (図 2.13)。型が違う値を代入することは出来ません。。

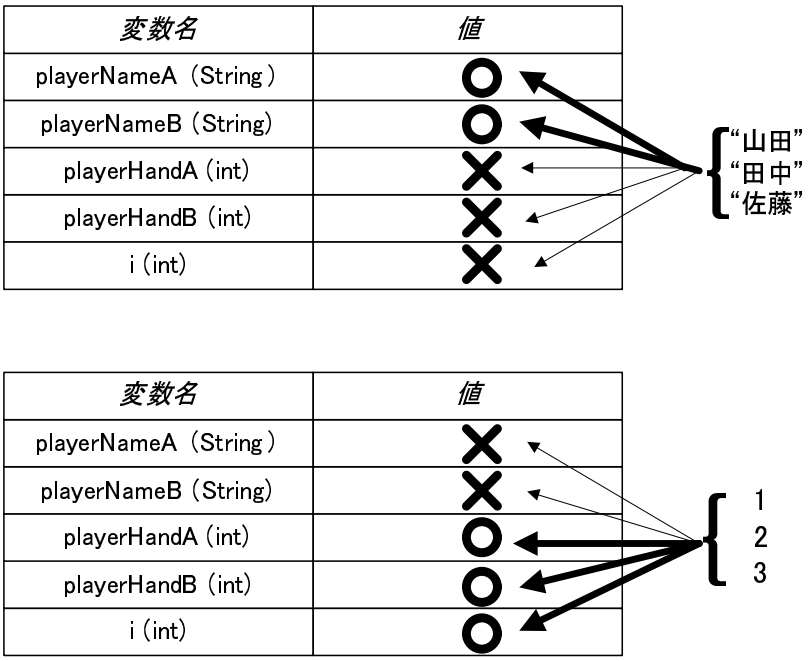


図 2.13: 変数の型と代入できる値

Java には 8 種類の基本データ型 [2]が用意されています。

実は、今までよく使ってきた String 型は、基本データ型ではありません。Java では、基本データ型のほかに、基本データ型をいくつか組み合わせた型が存在します。(String 型はその一種です。) 組み合わせ型については、オブジェクト指向編で詳しく説明します。

2.3.2 型の変換

変数は型によって扱いがことなるので、型の違う変数同士を足したり比較したりすることはできません。正札を出力するアプリケーション (リスト 12) のように、浮動小数点型

[2] 基本データ型の一覧は「Java 言語プログラミングレッスン (上)」の付録 H を参照してください。

の消費税率や割引率と整数値型の定価を掛け合わせて税込価格や価格を計算したい場合は型の変換（キャスト）を行います。

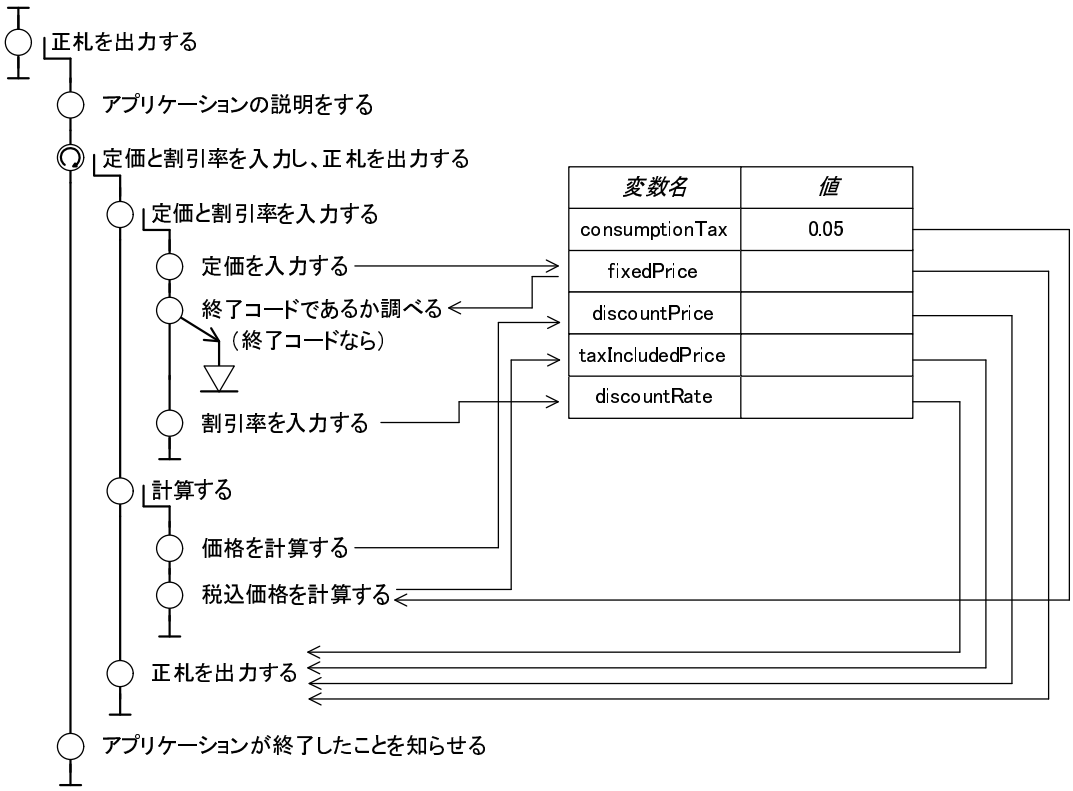


図 2.14: 正札を出力するアプリケーションの HCP チャート

リスト 12: 正札を出力するアプリケーション

```
 1: public class PriceTagApplication {
 2:
 3:   public static void main(String[] args) {
 4:     PriceTagApplication priceTagApplication = new PriceTagApplication();
 5:     priceTagApplication.main();
 6:   }
 7:
 8:   void main() {
 9:
10:     double consumptionTax = 0.05; // 消費税率
11:     int fixedPrice; // 定価
12:     int discountPrice; //（割引後の）価格
13:     int taxIncludedPrice; // 税込み価格
14:     int discountRate; //割引率
15:
16:     //アプリケーションの説明をする
17:     System.out.println("正札を出力します");
18:     System.out.println("※定価の入力が０の場合に終了します");
19:
```

```
20:     //定価と割引率を入力し、正札を出力する
21:     while (true) {
22:
23:       //定価を入力する
24:       System.out.println("定価を入力してください");
25:       System.out.print(">>");
26:       System.out.flush();
27:       fixedPrice = Input.getInt();
28:
29:       //終了コードか調べる
30:       if (fixedPrice == 0) { //終了コードなら（定価が0なら）
31:         System.out.println("アプリケーションを終了します");
32:         break; //終了する
33:       }
34:
35:       //割引率を入力する
36:       System.out.println("割引率（%）を入力してください");
37:       System.out.print(">>");
38:       System.out.flush();
39:       discountRate = Input.getInt();
40:
41:       //価格を計算する
42:       discountPrice =
43:         (int) (fixedPrice * (1 - (double) discountRate / 100));
44:
45:       //税込価格を計算する
46:       taxIncludedPrice = (int) (discountPrice * (1 + consumptionTax));
47:
48:       //正札を出力する
49:       System.out.println("--------------------");
50:       System.out.println("定価:" + fixedPrice + "円");
51:       System.out.println("--------------------");
52:       System.out.println(
53:         "価格:" + discountPrice + "円 (" + discountRate + "%OFF) ");
54:       System.out.println("--------------------");
55:       System.out.println("税込価格：" + taxIncludedPrice + "円");
56:       System.out.println("--------------------");
57:     }
58:
59:     //アプリケーションが終了したことを知らせる
60:     System.out.println("アプリケーションが終了しました。");
61:   }
62: }
```

Java Tips

キャスト

キャストは変換したい値に対して、変換先の型を () で囲ったものを前につけます。

break 文

while 文による繰り返し処理を中断したいときには break 文を用います。

2.4　プログラムの意味と変数

2.4.1　定数

章の冒頭で紹介した「じゃんけんアプリケーション」（リスト 11）は、1,2,3 という数字でじゃんけんの手である、グー、チョキ、パーを意味していました。しかし、最初に紹介したソースコードは、そのような数字の意味をコメントで補っていました。

1,2,3 という数字にグー、チョキ、パーという意味がある数字であるからには、それに名前を付けることで、よりプログラムの意味が明確になります（図 2.15）。

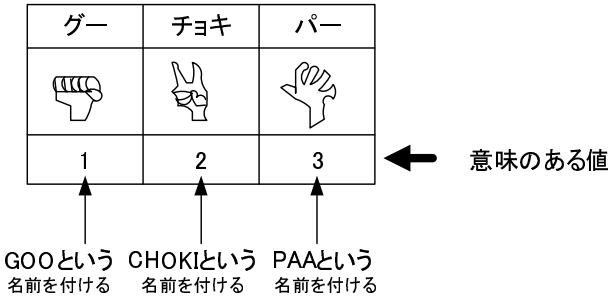


図 2.15: 意味のある値に名前を付ける

意味のある値に名前を付けたじゃんけんアプリケーションをリスト 13 に示します。

リスト 13: じゃんけんアプリケーション (定数をつけた)

```
 1: /**
 2:  * じゃんけんアプリケーション
 3:  *
 4:  * 二人のプレイヤーがじゃんけんの手を交互に入力すると、勝敗を調べて、
 5:  * 勝者の名前を表示する
 6:  * （定数を用いた）
 7:  *
 8:  * @author macchan
 9:  * @version $Id: JankenApplication.java,v 1.5 2003/05/04 20:54:13 gackt Exp $
10:  */
11: public class JankenApplication {
12:
13:   public static void main(String[] args) {
14:     JankenApplication jankenApplication = new JankenApplication();
15:     jankenApplication.main();
16:   }
17:
18:   void main() {
19:
20:     final int GOO = 1; //グーを表す定数
21:     final int CHOKI = 2; //チョキを表す定数
22:     final int PAA = 3; //パーを表す定数
```

```
23:
24:     String playerNameA; //プレイヤー A の名前
25:     String playerNameB; //プレイヤー B の名前
26:     int playerHandA; //プレイヤー A の出した手
27:     int playerHandB; //プレイヤー B の出した手
28:     int i; //ループ用
29:
30:     //アプリケーションの説明をする
31:     System.out.println("じゃんけん♪アプリケーション");
32:     System.out.println("（プレイヤー A の手に 0 を入力すると終了します）");
33:
34:     //プレイヤー A の名前を入力する
35:     System.out.println("プレイヤー A の名前を入力してください");
36:     System.out.print(">>");
37:     System.out.flush();
38:     playerNameA = Input.getString();
39:
40:     //プレイヤー B の名前を入力する
41:     System.out.println("プレイヤー B の名前を入力してください");
42:     System.out.print(">>");
43:     System.out.flush();
44:     playerNameB = Input.getString();
45:
46:     //手の入力を交互に行い、勝敗の表示をする
47:     while (true) {
48:
49:       //プレイヤー A の手を入力する
50:       System.out.println(playerNameA + "さんの手を入力してください");
51:       System.out.println("1. グー, 2. チョキ, 3. パー (0. 終了)");
52:       System.out.print(">>");
53:       System.out.flush();
54:       playerHandA = Input.getInt();
55:
56:       //終了コードであるか調べる
57:       if (playerHandA == 0) { //終了コードなら
58:         break; //アプリケーションを終了する
59:       }
60:
61:       //相手の手が見えないようにする
62:       i = 0;
63:       while (i < 100) {
64:         System.out.println();
65:         i = i + 1;
66:       }
67:
68:       //プレイヤー B の手を入力する
69:       System.out.println(playerNameB + "さんの手を入力してください");
70:       System.out.println("1. グー, 2. チョキ, 3. パー");
71:       System.out.print(">>");
72:       System.out.flush();
73:       playerHandB = Input.getInt();
74:
75:       //勝敗を判定して、結果を表示する
76:       if (playerHandA == GOO && playerHandB == CHOKI) { //グー VS チョキ
```

```
77:         System.out.println(playerNameA + "さんのかち");
78:       } else if (playerHandA == GOO && playerHandB == PAA) { //グー VS パー
79:         System.out.println(playerNameB + "さんのかち");
80:       } else if (playerHandA == CHOKI && playerHandB == GOO) { //チョキ VS グー
81:         System.out.println(playerNameB + "さんのかち");
82:       } else if (playerHandA == CHOKI && playerHandB == PAA) { //チョキ VS パー
83:         System.out.println(playerNameA + "さんのかち");
84:       } else if (playerHandA == PAA && playerHandB == GOO) { //パー VS グー
85:         System.out.println(playerNameA + "さんのかち");
86:       } else if (playerHandA == PAA && playerHandB == CHOKI) { //パー VS チョキ
87:         System.out.println(playerNameB + "さんのかち");
88:       } else if (playerHandA == playerHandB) { //あいこ
89:         System.out.println("あいこでした");
90:       } else { //不正な入力の処理
91:         System.out.println("不正な入力です");
92:       }
93:     }
94:
95:     //アプリケーションが終了したことを知らせる
96:     System.out.println("アプリケーションが終了しました。");
97:   }
98: }
```

「じゃんけんアプリケーション」ではプログラムの途中で、数字の意味が変わることがないので、一度値が設定されると、後から値を変更する（代入する）必要はありません。そのような場合は、変数ではなく、定数として宣言します。

定数を宣言する場合は以下のように行います。[3]

```
final [定数の型] [定数名] = [値];
```

定数にすると、「値に変更がなく、単一の意味を表す」ということを明確に表現できます。

[3] このテキストでは、定数の名前はすべて大文字で表記し、単語と単語の区切りは「_（アンダーバー）」で結ぶことにします。

2.5 練習問題

練習問題 1

男女の名前及び、男性、女性それぞれについて相手のことを好きか嫌いかを入力してもらい、両方の結果から相性を判断するアプリケーションを作成してください。(何度も入力を繰り返して、実行できるようにしてください)

練習問題 2

次のプログラム（リスト 14）の出力結果がどうなるかを予想して、どうしてそう思うのかという理由を表モデルを書き、評価の概念を用いて説明してください。また、問題があればどうしたら良いかを考え、どうしてその方法でよいと思うのか説明してください。

リスト 14: 2 つの数を足して、結果を表示するサンプルプログラム

```
/**
 * ２つの数を足して結果を表示するサンプルプログラム
 *
 * @author Manabu Sugiura
 * @version $Id: AddTwoNumbersSample.java,v 1.4 2003/05/04 17:16:47 macchan Exp $
 */
public class AddTwoNumbersSample {

  public static void main(String args[]) {
    AddTwoNumbersSample addTwoNumbersSample = new AddTwoNumbersSample();
    addTwoNumbersSample.main();
  }

  void main() {

    int numberA; // １番目の整数
    int numberB; // ２番目の整数

    //整数に値を設定する
    numberA = 3;
    numberB = 5;

    // 足し算の結果を表示する
    System.out.println(
      numberA + "と" + numberB + "の和は" + numberA + numberB + "です");
  }
}
```

練習問題 3*

肉屋のレシートを出力するアプリケーションを作ってください。仕様は次の通りとします。

1. 買う肉の種類の名前、その肉のグラム単価、買う量（g）を 1 つ入力すると、税込価格の値段が表示される
2. 税込価格に対して、支払うお金の金額を入力する
3. 支払い金額が税込価格より小さければ、お金が足りないというエラーを出力して、再度支払い価格の入力を求める
4. 支払い金額が税込価格より大きければ、お釣りの金額を計算する
5. レシートを出力する

   買う肉の種類の名前、その肉のグラム単価、買う量（g)、定価、税込価格、支払い金額を表示。おつりがある場合はおつりの金額も表示する