5章 変数を理解しよう

変数の概要や使い方、変数名のつけ方ルールを学びます。

90分 - 未読

5.1 本章の目標

本章では以下を目標にして学習します。

- 変数の概要や使い方を知ること
- 変数名のつけ方ルール（命名規則）を知ること

本章では、プログラミングに欠かせない変数を学んでいきます。Javaの変数まわりは覚えることが多いですが、変数を使えなければWebアプリケーションの開発は不可能です。

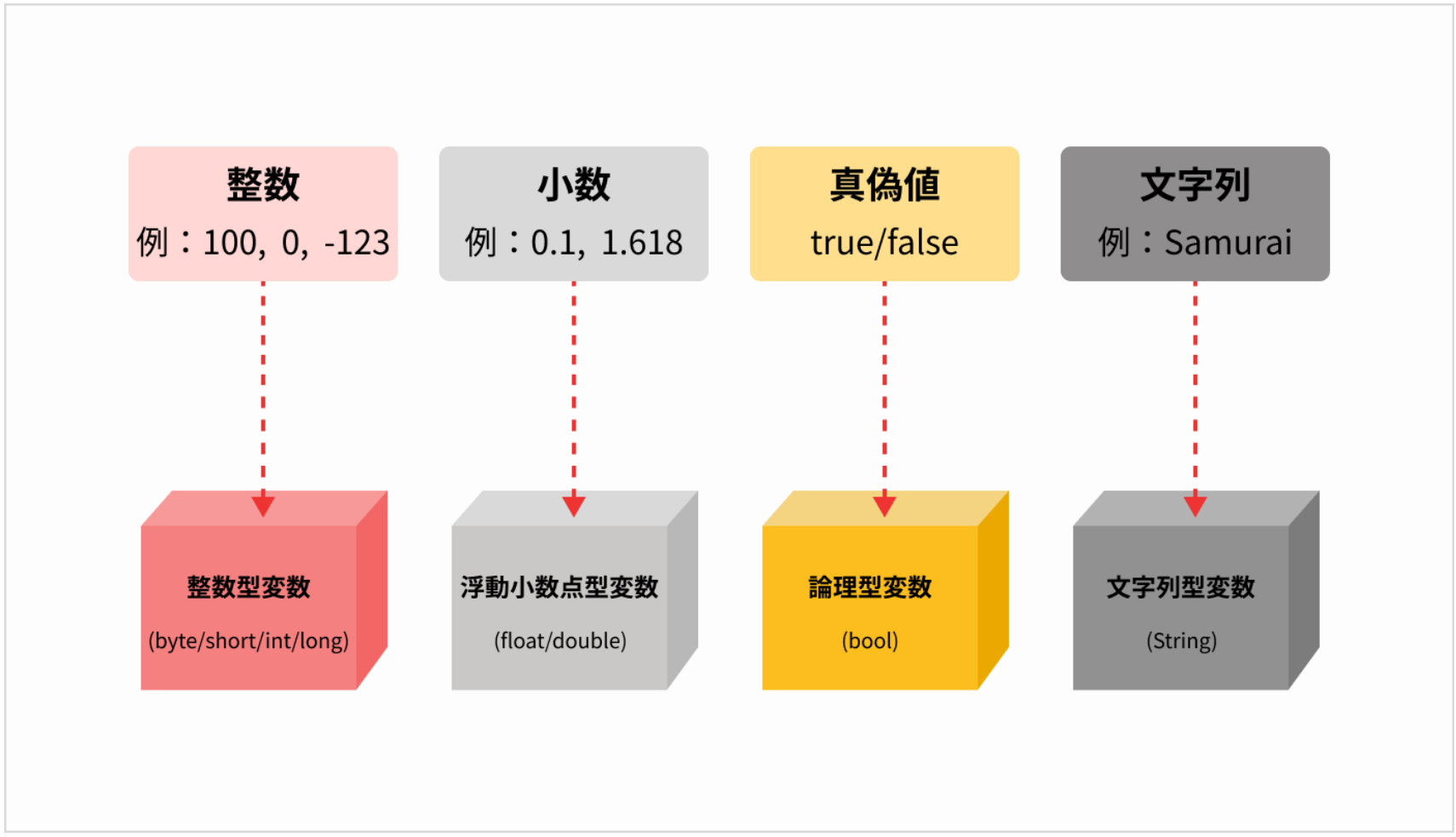
変数は文法の中でも重要度が高いため、しっかり覚えましょう。なお本章の実践パートでは、Eclipseのプロジェクトに**パッケージ「text.section_05」を作成**してください。

またパッケージの中には、**ファイル「Variable_番号.java」** を順番に作成しましょう（作成方法は3章を参照）。Variableは「変数」です。

5.2 変数とは

変数とは簡単にいえば、**文字列や数値などのデータを入れる箱のようなもの**です。後で使いたいデータを保管しておいたり、必要なときに取り出したりできます。

Javaで変数を使うときは、**入れたいデータの種類（データ型）に合わせた箱**が必要です。たとえば「100」なら整数型の箱、「Samurai」なら文字列型の箱を用意します。



変数の大きなメリットは、**好きなときに中身を入れ替えられる**ことです。

前章で扱ったデータは、「100」「こんにちは」のように中身が決まったものだけでした。しかし実際のプログラムやサービスでは、データの中身を入れ替えることが必要になってきます。

たとえば、画面に「おかえりなさい、〇〇さん」と表示するWebアプリケーションを考えてみましょう。赤枠の「SAMURAI」の部分は、ログインしたユーザー名に合わせて変える必要があります。

仮に上の「SAMURAI」をソースコードに直接記述していたら、すぐに表示を変えられません。「侍太郎」「侍花子」とユーザーが変わるたびに、プログラムの作り直しが必要です。

こうしたケースでは、変数が役に立ちます。ユーザー名の変数を用意しておき、ユーザーに合わせてプログラム内で変数を書き換えれば、プログラムを作り直す必要がありません。

Javaに限らず、プログラミングでは変数を当たり前のように使います。以降の節で、変数の使い方をしっかり学びましょう。

5.3 変数の使い方を理解しよう

変数を使うときの流れは、**宣言→代入→参照**が基本です。

1. 宣言（箱を用意する）
2. 代入（箱にデータを入れる）
3. 参照（箱のデータを使う）

ひとつずつ、順番に説明します。

変数の宣言

変数の宣言とは、データを入れるための箱を用意する手続きのこと。「こういう変数をこれから使いますよ」と宣言することで、変数の箱が作られます。書き方は以下のとおりです。

```
[データ型] [変数名];
```

4章ではint型やfloat型など、さまざまなデータ型があることを学びました。Javaの変数は、データ型によって箱が異なるため、**変数名だけでなくデータ型も指定が必須**です。

たとえば、int型で「test」という変数を使いたいときは、以下のように宣言します。

```
int test;
```

こう書くことで、int型の変数 `test` が作られます。変数 `test` には「123」などの整数は入れられますが、小数の「0.1」や文字列の「こんにちは」などは入れられません。

変数の代入・初期化

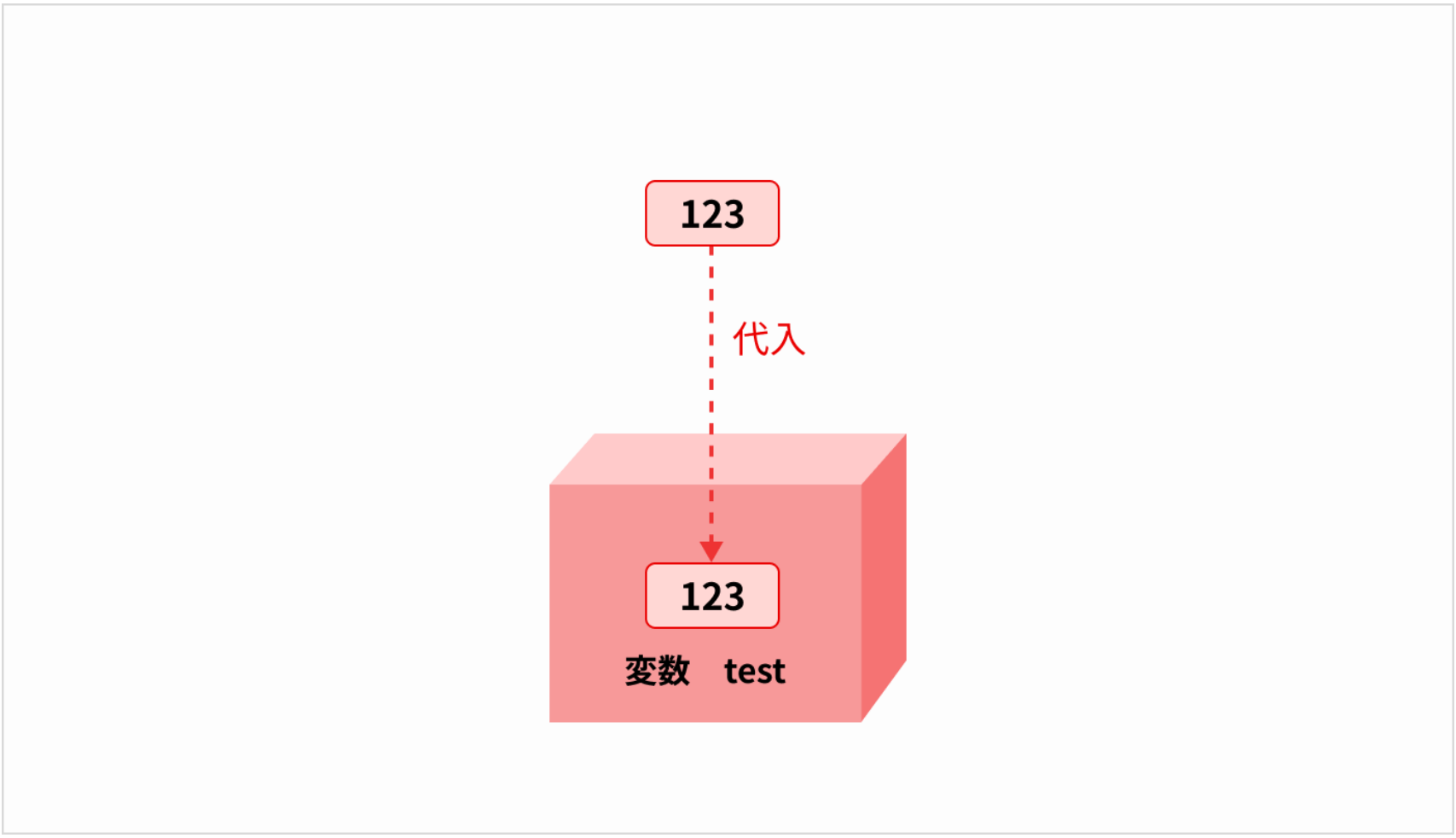
上記のように宣言しただけの変数には、まだ何も入っていません。変数を宣言したら、中身を入れる手続きである**代入**が必要です。代入には「=」（**代入演算子**）を使います。

```
[変数名] = [変数に入れたい値];
```

中身が入っていない変数を使おうとするとエラーとなるため、宣言後には代入が欠かせません。たとえば、宣言済みの変数 `test` に「123」を代入する書き方は、以下のとおりです。

```
test = 123;
```

testという箱に「123」を入れていることをイメージしてください。



なお、変数の宣言時にデータを代入することも可能です。これを**初期化**と呼びます。変数の初期化を行うときは、以下のように宣言と代入を1行で書きましょう。

```
[データ型] [変数名] = [変数の初期値];
```

たとえば、変数 test の宣言時に「123」で初期化する場合の書き方は、以下のとおりです。

```
int test = 123;
```

この1行だけで変数 test が作られ、「123」が代入されます。上記のような書き方で変数を初期化することも多いため、しっかり覚えておきましょう。

変数の参照

データを代入した変数は、必要なときに**参照**できます。参照とは、変数の中身を使う手続きのこと。参照するときは、プログラム中に**変数名をそのまま書くだけ**でOKです。

たとえば以下のように書くと、「123」が代入された変数 test を参照して表示できます。

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

こうすることで変数 test を参照でき、中身である「123」が表示されます。前述のとおり、変数 test の宣言や代入を行っていないと、エラーになるため注意しましょう。

変数を使ってみよう

では、変数を実際に使ってみましょう。ファイル「Variable_1.java」を作成し、以下のように書いてください（以降はファイルの番号を1ずつ増やしましょう）。

Eclipseはリアルタイムに構文チェックしてくれるため、書いている途中に赤い波線が出ます。1行を正しく書き終われば赤い波線は消えるため、気にせず最後まで書いてください。

Variable_1.java

```
1  package text.section_05;
2
3  public class Variable_1 {
4      public static void main(String[] args) {
5
6          // 変数testを宣言
7          int test = 123;
8
9          // 変数testの値を表示
10         System.out.println(test);
11     }
12 }
13
```

上記のプログラムを実行して、以下のように表示されることを確認しましょう。

```
1  123
2
```

変数 test の初期値である「123」が表示されました。

前章では、int型以外にも数多くのデータ型を学びました。ほかのデータ型でも、同様に変数の宣言・代入・初期化を実践しましょう。書き方はint型と基本的に同様です。

Variable_2.java

```java
package text.section_05;

public class Variable_2 {
    public static void main(String[] args) {

        // 整数型の変数を宣言
        byte  testByte  = 1;
        short testShort = 12;
        int   testInt   = 123;
        long  testLong  = 1234;

        // 浮動小数点型の変数を宣言
        float  testFloat  = 0.123F;  // 代入する値の末尾にはFが必要
        double testDouble = 0.123456;

        // 論理型の変数を宣言
        boolean testBoolean = true;

        // 文字列型・文字型の変数を宣言
        String testString = "Samurai";
        char   testChar   = 'S';

        // 変数testの値を表示
        System.out.println(testByte);
        System.out.println(testShort);
        System.out.println(testInt);
        System.out.println(testLong);
        System.out.println(testFloat);
        System.out.println(testDouble);
        System.out.println(testBoolean);
        System.out.println(testString);
        System.out.println(testChar);
    }
}
```

実行結果

```
1
12
123
1234
0.123
0.123456
true
Samurai
S
```

このように変数は宣言して、代入して、参照するのが基本的な使い方です。しっかり覚えておきましょう。

補足：float型の値は末尾に「F」が必要

注意点として、**float型の変数に代入する値の末尾には「F」が必要**です。以下のようにFをつけたリテラル（実際の値）は、float型として扱われます。

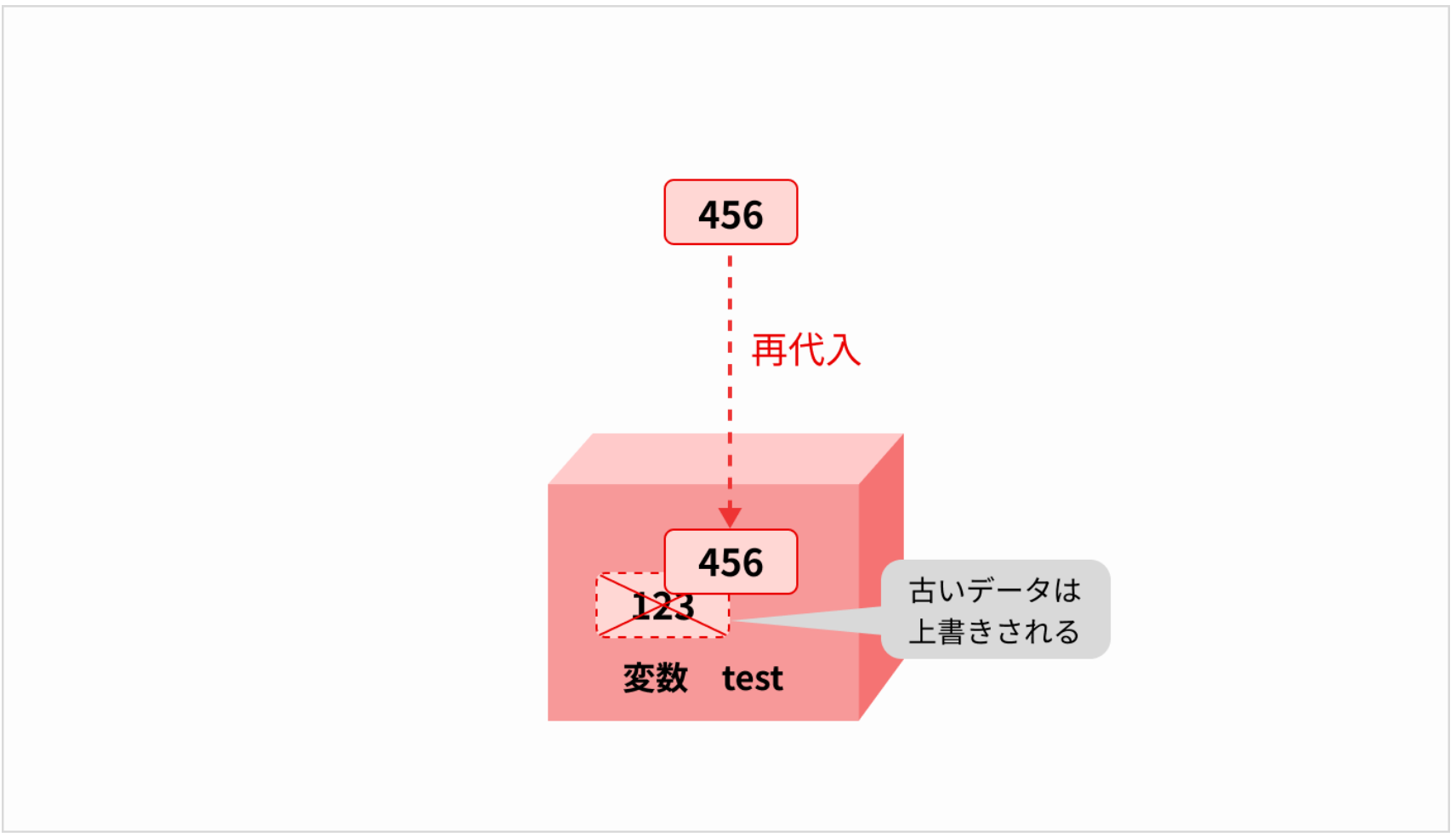
```java
// 浮動小数点型の変数を宣言
float  testFloat  = 0.123F;
```

「0.123」のように小数のリテラルは、Fをつけないとdouble型扱いとなります。変数はfloat型なのに、入れる値がdouble型だと、データ型の不一致でエラーとなってしまうのです。

5.4 変数の中身を入れ替えてみよう

変数に代入した中身は、**別の値を再代入**することで入れ替えられます。このとき、新しい値に上書きされるため、元々のデータはなくなる点に注意が必要です。

たとえば、「123」で初期化した変数 `test` に「456」を再代入すると、値が「456」に入れ替わります。後で「123」を使いたくなっても、参照することはできません。



では、実際に変数testの中身を入れ替えてみましょう。以下のようなソースコードを書いてください。

Variable_3.java

```
1  package text.section_05;
2
3  public class Variable_3 {
4      public static void main(String[] args) {
5
6          //  変数testに123をセット
7          int test = 123;
8          System.out.println(test);
9
10         // 変数testの中身を456に入れ替える
11         test = 456;
12         System.out.println(test);
13     }
14 }
15
```

「test = 456;」のように、**すでにある変数の中身を入れ替えるときにデータ型は不要**です。データ型のintをつけると、変数 `test` を再度宣言する扱いとなるため注意しましょう。

上記プログラムの実行結果は、以下のとおりです。まず変数 `test` の初期値「123」が表示されますが、値を入れ替えた後は「456」が表示されました。

```
123
456

```

このように、値を再代入することで変数の中身を入れ替えられます。ユーザー情報に合わせて変数を上書きする場合など、Webアプリでもよく使われるため覚えておきましょう。

5.5 変数を使って数値の計算や文字列の連結をしてみよう

変数は数値（整数または小数）や文字列と組み合わせて、計算や連結が可能です。前章の復習も兼ねて、変数を使って数値の計算や文字列の連結をしてみましょう。

では、以下のようにソースコードを書いてください。

Variable_4.java

```java
package text.section_05;

public class Variable_4 {
    public static void main(String[] args) {

        // 整数型と浮動小数点型の足し算
        int   number1 = 5;
        float number2 = 2.5F;
        System.out.println(number1 + number2);

        // 文字列型と文字列型の連結
        String lastName  = "侍";
        String firstName = "太郎";
        System.out.println(lastName + firstName);
    }
}

```

上記のプログラムでは以下2つの計算を行い、結果を表示しています。

- int型の変数「number1」、float型の変数「number2」の加算
- 文字列型の「lastName」「firstName」の連結

プログラムを実行して、以下のように表示されることを確認しましょう。

```
7.5
侍太郎

```

また、変数を使って計算した結果を、別の変数にセットすることも可能です。以下の例では、変数 `number3` の値を「4」で割った結果を、変数 `number4` に代入しています。

```
int number3 = 32;
int number4 = number3 / 4;

```

注意点として、上記の2行目では変数 number3 を計算に使っていますが、**それ自体は書き換えていません。** number3の値は、あくまで初期化時にセットした「32」のままです。

こちらも実践しましょう。以下のようにソースコードを書き、実行してください。

Variable_5.java

```
package text.section_05;

public class Variable_5 {
    public static void main(String[] args) {

        int number3 = 32;
        int number4 = number3 / 4;
        System.out.println("number3は" + number3);
        System.out.println("number4は" + number4);
    }
}

```

実行結果

```
number3は32
number4は8

```

変数 number3 は書き換えていないため、初期化時の「32」が表示されました。変数 number4 は、変数 number3 の値を4で割った「8」が表示されており、正しい計算結果です。

このように、変数の数値計算や文字列連結は、リテラルと同様に行えます。

5.6 final変数

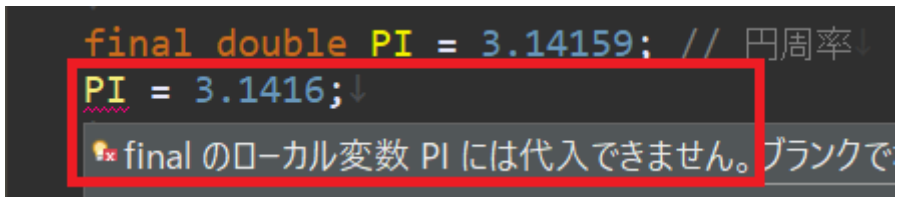
final 変数とは、宣言時に代入した初期値を入れ替えられない変数です。言い換えれば、**最初にセットした値が最終的（final）な値となる変数**、ということですね。

final 変数を宣言するときは、通常の変数宣言の頭にfinalをつけます。

```
final double PI = 3.14159; // 円周率

```

こうして宣言した final 変数には、再代入ができません。たとえば、以下ではPI（円周率）という final 変数に再代入しようとして、エラーが出ています。



`final` 変数は、繰り返し参照するものの、値を変えたくないデータの扱いに便利です。上記のように、誤って変えてしまうミスを防げます。

また `final` 変数を使えば、プログラム変更の負担も減らせます。以下では円周率の意味で「3.14159」を2度使っていますが、「3.14」に変えるなら2ヶ所とも変更が必要です。

```
// 円の面積を求める
circleArea    = 3.14159 * radius * radius;

// 円周を求める
circumference = 3.14159 * (radius * 2);

```

一方、以下のように `final` 変数「PI」を追加すれば、円周率が「3.14」になってもPIの初期値だけの変更で済みます。PIが同じものを指していることも明確となりますね。

```
final double PI = 3.14159; // 円周率

// 円の面積を求める
circleArea    = PI * radius * radius;

// 円周を求める
circumference = PI * (radius * 2);

```

では、実際に `final` 変数を使ってみましょう。以下のようなソースコードを書いてください。半径5の円の面積・円周を求めて、表示するプログラムです。

Variable_6.java

```
package text.section_05;

public class Variable_6 {
    public static void main(String[] args) {

        final double   PI = 3.14159;              // 円周率
        double circleArea = 0, circumference = 0; // 円の面積・円周
        int        radius = 5;                    // 円の半径

        // 円の面積を求める
        circleArea    = PI * radius * radius;
        System.out.println(circleArea);

        // 円周を求める
        circumference = PI * (radius * 2);
        System.out.println(circumference);
    }
}

```

なお、**データ型の同じ変数は「,」を用いてまとめて宣言・初期化が可能**です。以下の箇所では、double型の変数「circleArea」「circumference」を1行で宣言・初期化しています。

```
double circleArea = 0, circumference = 0; // 円の面積・円周

```

ではプログラムを実行して、以下のように表示されることを確認してください。

```
78.53975
31.4159

```

プログラムの中で値を変えたくないデータを扱うときは、このように `final` 変数を活用しましょう。

補足：定数名は大文字＋アンダースコア（_）

`final` 変数のように、値が一定となるデータのことを「定数」と呼びます。**定数名は全て大文字、かつ単語間をアンダースコア（_）で区切るのが慣例**です。

たとえば、通常の変数なら「testSamurai」ですが、定数なら「TEST_SAMURAI」としましょう。このように書くことで、通常の変数と定数を区別しやすくなります。

変数の名前のつけ方は次節で説明しますが、定数名は毛色が異なるため注意しましょう。

5.7 変数名のつけ方ルール（命名規則）

プログラミングにおいて、名前のつけ方のルールを **「命名規則」** と呼びます。プログラマー自身が名づける変数にも、命名規則は存在します。

チームやプロジェクトなど、複数のプログラマーが参加する場合は、その中で命名規則を定義することも多いです。ここでは、ごく一般的なルールに従って解説します。

ルール違反とならないように、変数名の正しい命名規則を知っておきましょう。以下の3つに分けて解説します（ローワーキャメルケースは後述）。

- 変数名に使える文字・使えない文字
- 変数名はローワーキャメルケースで記述する
- 変数の中身がわかるような名前にする

変数名に使える文字・使えない文字

まずは、使える文字・使えない文字を把握しましょう。Javaの変数名に使える文字は、以下の3種類です。これらを組み合わせることもできます。

- 半角英字（a～z、A～Z）

- 半角数字（0〜9）
- アンダースコア（_）

実は、全角文字や日本語を変数名に含めることも不可能ではありません。しかし、慣例的にまず使われることはないため、基本的に使用は避けましょう。

なお、以下の文字は変数名には使えず、エラーとなります。

- 変数名の先頭に数字（例：3testなど）
- アンダースコア以外の記号（例：test@や&testなど）
- 予約語と同名（例：boolean、finalなど）

3章でも軽く触れた予約語は、文法として特別な意味を持つキーワードでしたね。たとえば「boolean」は、論理型のデータを表す予約語のため、変数名にはできません。

「boolean」というワードがあると、Javaのコンパイラは「論理型だな」と解釈します。仮に変数名が「boolean」だったら、論理型と勘違いされてしまうためNGなのです。

```
1 boolean boolean; // 変数名も論理型だと勘違いされてしまうためNG
2
```

変数名はローワーキャメルケースで記述する

Javaの変数名は **「ローワーキャメルケース」** での記述が推奨されています。キャメルケースとは、複数の単語を組み合わせるときに、各単語の先頭を大文字にする記法です。

キャメルケースにも、先頭が小文字のローワーキャメルケース、先頭が大文字のアッパーキャメルケースの2種類があります。

Javaの変数名に推奨されるローワーキャメルケースの特徴は、以下のとおりです。

- 単語と単語を組み合わせる
- 1単語目はすべて小文字
- 2単語目以降は先頭だけ大文字

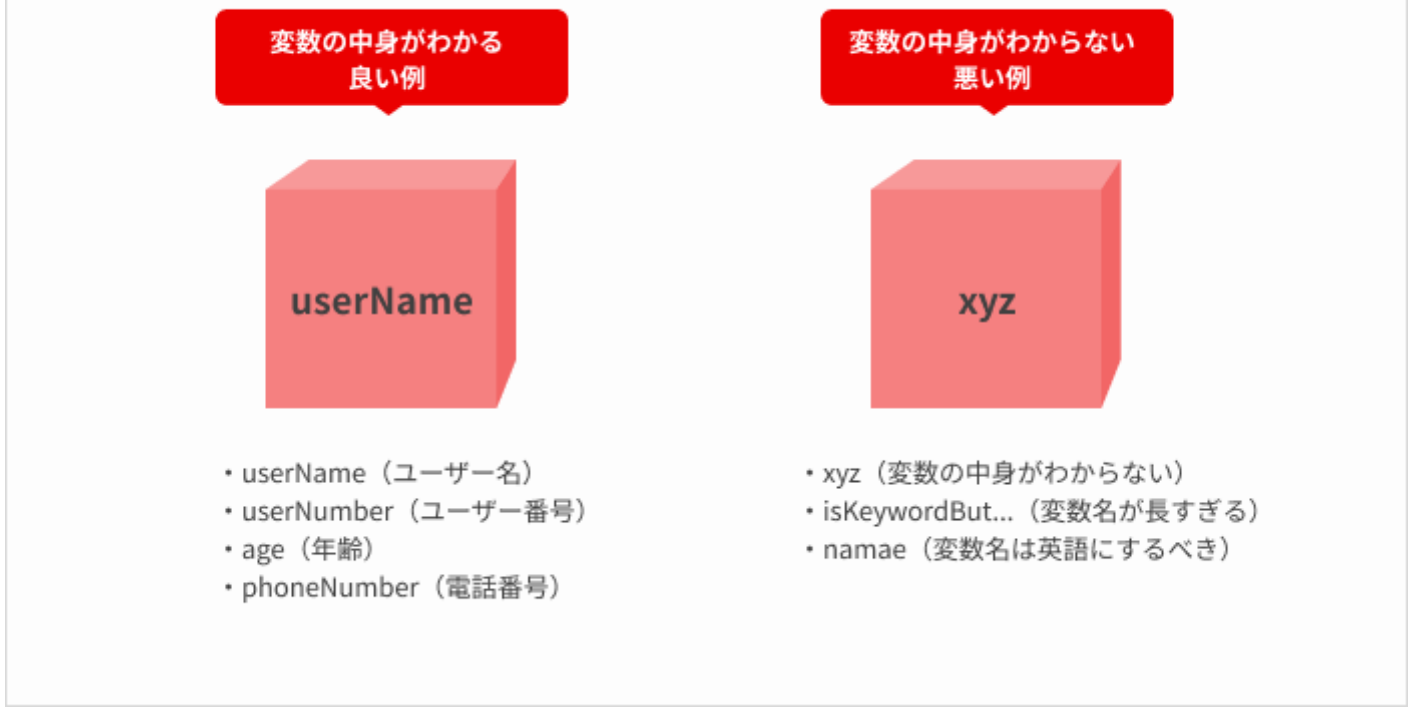
たとえば、「user」と「name」を組み合わせて変数名にするなら、「userName」としましょう。1単語目の「user」はすべて小文字、2単語目の「name」は先頭だけ大文字です。

1単語のみなら「user」のようにすべて小文字で構いません。

これまでの「lastName」「firstName」などもローワーキャメルケースです。本章以降でもローワーキャメルケースで変数名をつけるため、この書き方に慣れていきましょう。

なお「Python」などでは、「user_name」「user_number」のような**「スネークケース」**が推奨されています。言語によって推奨ルールが変わることも知っておきましょう。

変数の中身がわかるような名前にする



変数は、その中身がわかるような名前にしましょう。そうでないと、後からソースコードを読むときに「これはなんの変数だっけ？」となってしまうためです。

たとえば、「userName（ユーザー名）」であれば、見ただけで意味がわかります。反対に、「xyz」のような変数名だと、変数の中身がイメージしづらいためNGです。

それ以外の悪い例も紹介します。

- isKeywordButNothingOrSomething（変数名が長すぎる）
- namae（変数名は英語にするべき）

変数名に文字数制限はありませんが、あまりに長いと読みづらいため避けましょう。また、日本語をローマ字読みする変数名も避けたほうが無難です（開発チームにもよります）。

本章の学習は以上です。お疲れさまでした。

まとめ

本章では以下の内容を学習しました。

- 用語
  - 変数：文字列や数値などのデータを入れる箱のようなもの
  - 代入：変数に値をセットすること
  - 初期化：変数の宣言時に初期値を代入すること
  - 参照：変数の中身をチェックすること
  - `final` 変数：宣言時に代入した初期値を入れ替えられない変数
  - 定数：値が一定となるデータのこと
  - 命名規則：名前のつけ方のルール
  - キャメルケース：複数の単語を組み合わせるときに、各単語の先頭を大文字にする記法。先頭が小文字のローワーキャメルケース、先頭が大文字のアッパーキャメルケースの2種類がある
- 文法
  - 変数の宣言
    - [データ型] [変数名];のように記述する
      - 変数 `test` の中身をセットしないと、参照時にエラーとなる
  - 変数の代入
    - [変数名] = [変数に入れたい値];のように記述する
  - 変数の初期化（宣言時に初期値を代入）
    - [データ型] [変数名] = [変数の初期値];のように記述する
    - float型の変数に代入する値の末尾には「F」が必要
  - 変数の参照
    - 変数名をそのまま書くだけでOK
    - ただし、事前に宣言や代入を行っていないとエラーになる
  - 変数に代入した中身は、別の値を再代入することで入れ替えられる
  - 変数の数値計算や文字列連結は、リテラル（値）と同様に行える
  - `final` 変数の宣言
    - 通常の変数宣言の頭にfinalをつける
  - 変数の命名規則
    - 変数名に使える文字・使えない文字があるため注意が必要
    - 変数名はローワーキャメルケースで記述する
    - 変数の中身がわかるような名前にする

覚えることが多くて大変だったでしょう。お疲れ様でした。

実際の開発現場では、変数を使わないプログラムはありません。変数は、プログラミングにおける基本中の基本のため、本章の内容をしっかり復習しましょう。

次章では、データの型変換について学びます。

理解度を選択して次に進みましょう

ボタンを押していただくと次の章に進むことができます

～50%　50～80%　80～100%

最後に確認テストを行いましょう

下のボタンを押すとテストが始まります。

教材をみなおす　テストをはじめる

前に戻る　6 / 31 ページ　次に進む

一覧に戻る　改善点のご指摘、誤字脱字、その他ご要望はこちらからご連絡ください。