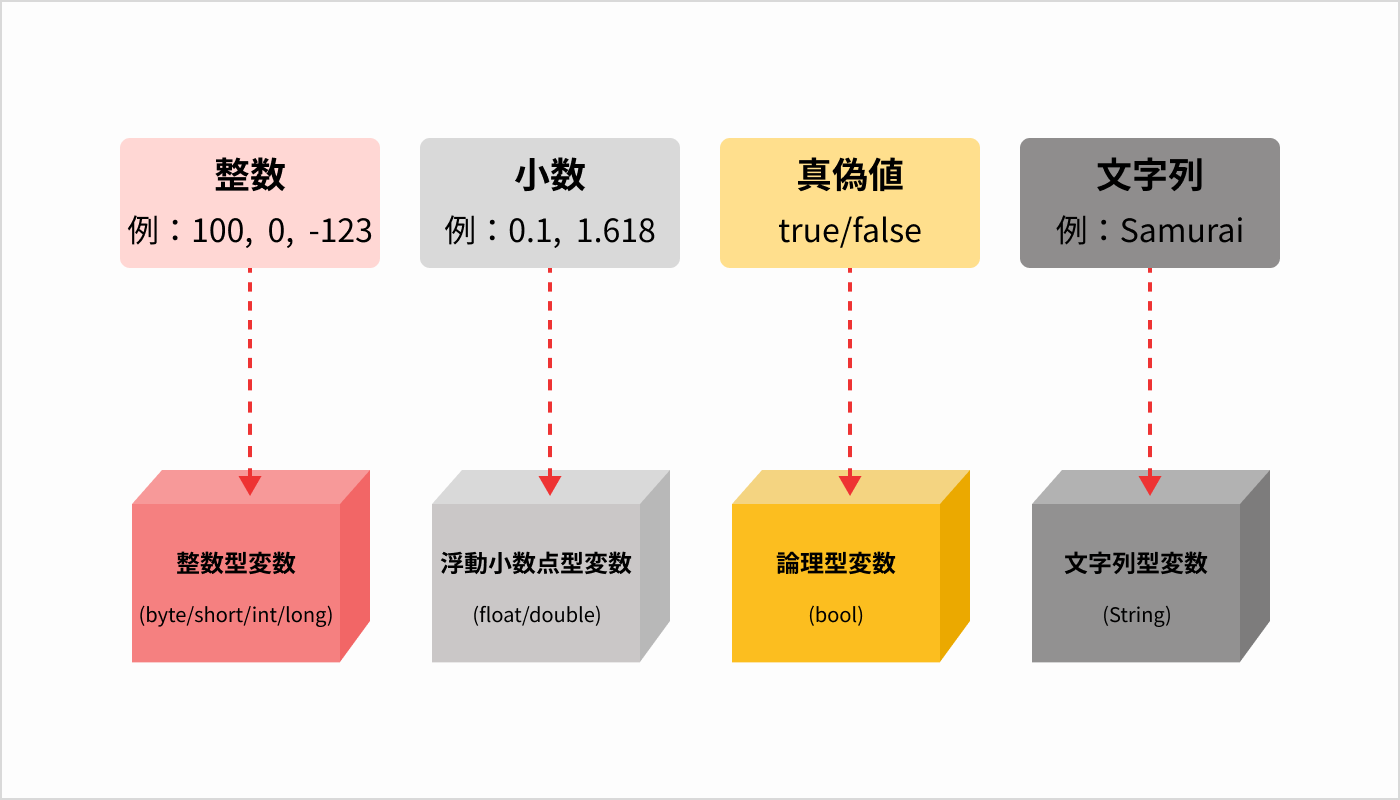
変数とは、文字列や数値などのデータを入れる箱のようなもの

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る

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

例えば「100」なら整数型の箱、「テスト」なら文字列型の箱を用意する



変数の大きなメリットは好きな時に中身を入れ替えられる事

「100」「テスト」のように中身が決まったものを扱う事もできるが

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られることが必要になってくる

例えば画面に「おかえりなさい、〇❍さん」と表示するWebアプリを考えてみると

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る。



仮に上の「SAMURAI」のソースコードを直接記述していたら、すぐに表示を変えられない

「太郎」「二郎」とユーザーが変わるたびにソースコードを書き直す必要がある

こういったケースは変数が役に立つ、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なくなる

Javaだけではなくプログラミングでは変数を当たり前のように使うので理解しておくこと

変数を使う時の流れは 宣言→代入→参照　が基本

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）

変数の宣言

データを入れるための箱を用意する手続きのようなもの

「こういう変数をこれから使いますよ」と宣言をすることで、変数の箱が作られる

基本的な書き方

「データ型」「変数名」;

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の指定が必須

例えばint型で「test」という変数を使いたいときは、このように宣言する

int test;

このように書くことでint型の変数testが作られる、変数testには「123」などの整数は入れられるが、小数「0.1」や文字列の「こんにちは」などは入れられない

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、まだ何も入っていない

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

代入には「＝」代入演算子を使う

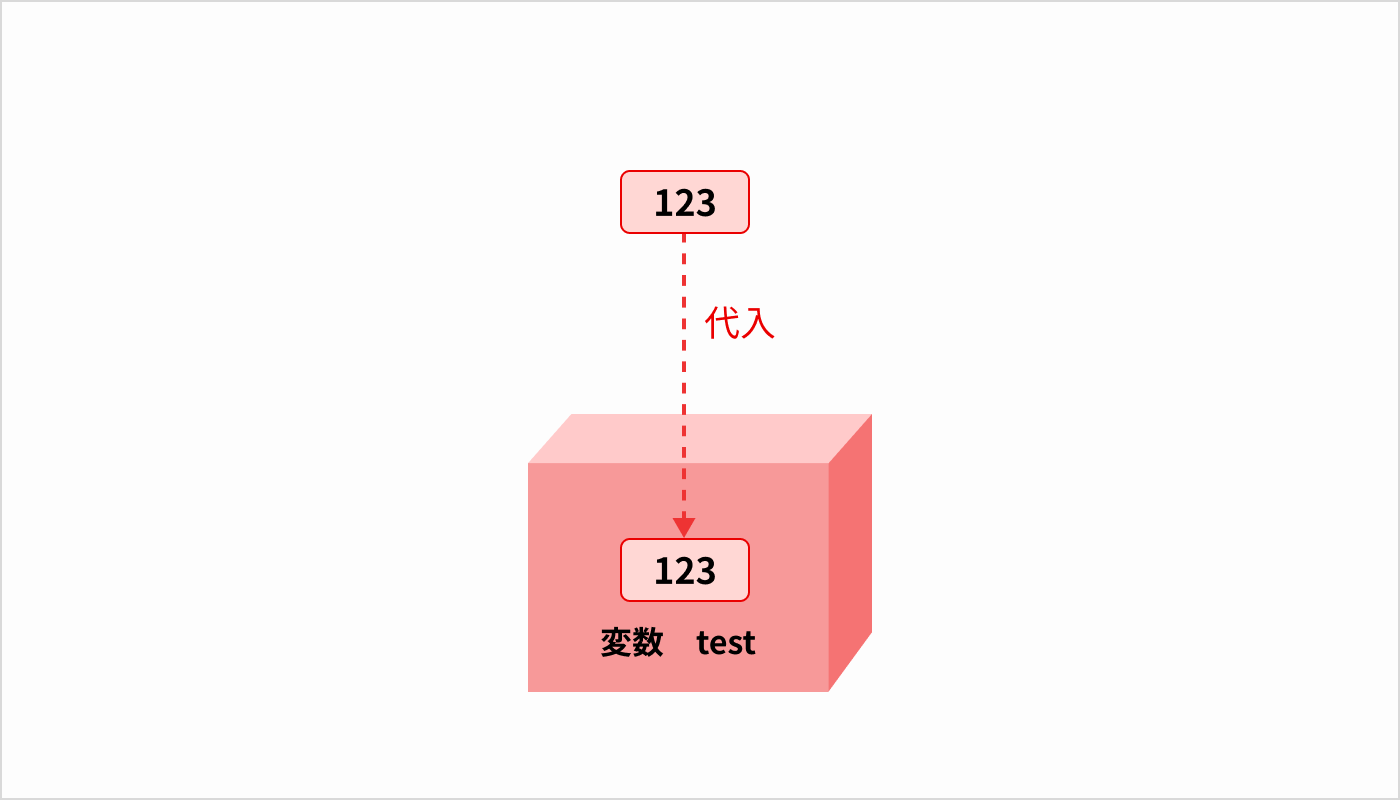
[変数名] = [変数に入れたい値];

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

宣言後には代入は必須となる

例えば宣言済みの変数testに「123」を代入する書き方

test = 123;



なお、変数の宣言時にデータを代入することも可能、これを初期化と呼ぶ

変数の初期化を行うときはｍ以下のように宣言と代入を１行で書く

[データ型] [変数名] = [変数の初期値];

例えば、変数testの宣言時に「123」で初期化する書き方

int test = 123;

この一行で変数testが作られ、「123」が代入される

上記のような書き方で変数を初期化することが多い

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る、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

例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る

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」が表示される

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

変数を使ってみる

public class variable01 {

public static void main(String[] args) {

// 変数testを宣言

int test = 123;

// 変数testの値を表示

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

}

}

出力結果

123

このように、変数testの初期値である「123」が表示される

次にこれまでに学んだデータ型を同様に宣言、代入、参照をしてみる

public class variable02 {

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;

// 浮動店小数点の変数を宣言

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 = "test1";

char testChar = 'T';

// 変数の値を表示

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

test1

T

このようにして変数は宣言して代入して、参照するのが基本的な使い方になる

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

以下のようにしてFを付けたリテラル（実際の値）はfloat型として扱われる

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型扱いとなる

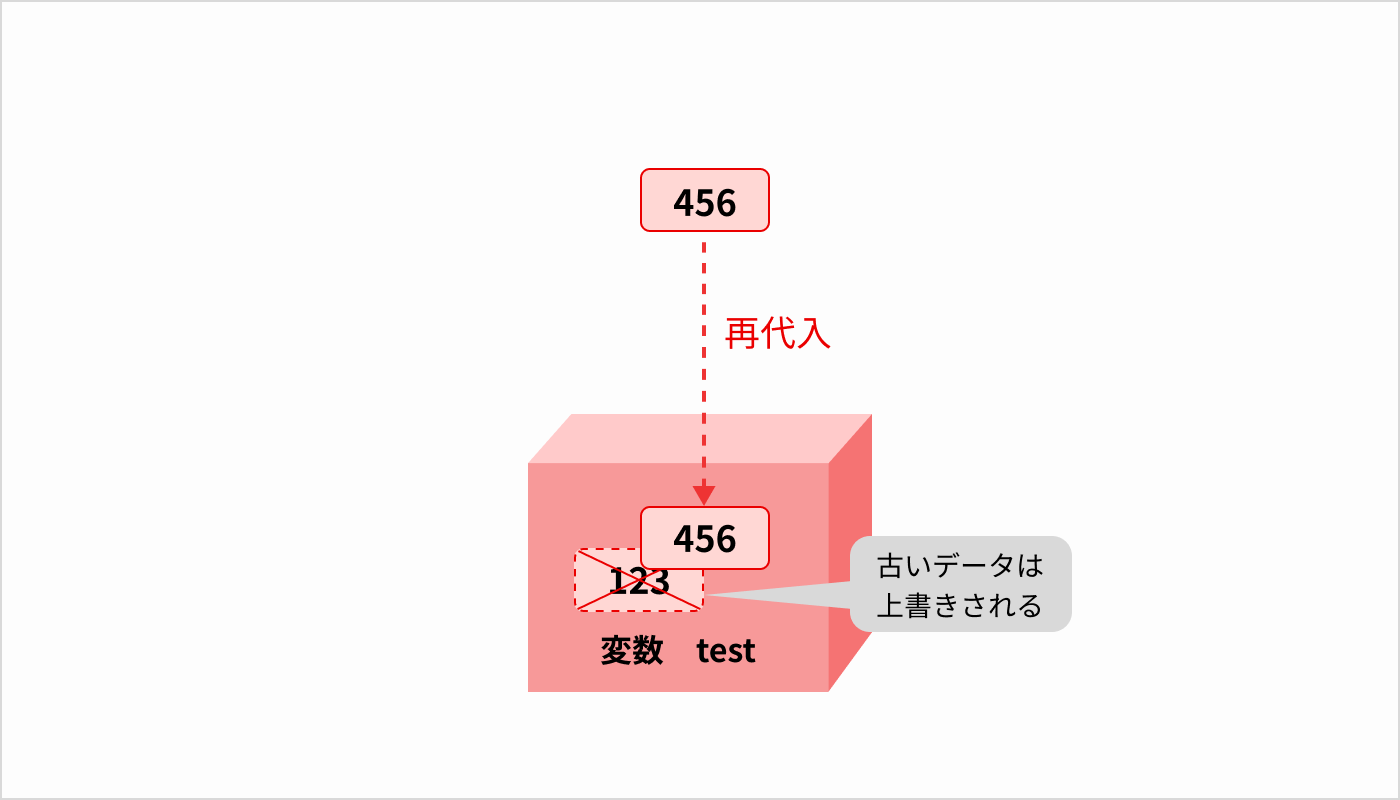
変数はfloat型なのに入れる値がdouble型だとデータ型の不一致でエラーとなってしまう仕組み

変数の中身を入れ替えてみる（再代入）

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、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

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」に入れ替わる。

後で「123」を使いたくなっても参照することはできない



実際に入れ替えてみる

public class variable3 {

public static void main(String[] args) {

// 変数testに123をセット

int test = 123;

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

// 変数testの中身を456に入れ替える

test = 456;

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

}

}

出力結果

123

456

「test = 456」のように既にある変数の中身を入れ替えるときにデータ型は不要

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する

上記のプログラムの実行結果は、まず変数testの初期値「123」が表示されるが値を入れ替えた後は「456」が表示される

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る

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

変数を使って数値の計算や文字列の連携をしてみる

変数は数値（整数または小数）や文字列を組み合わせて、計算や連結が可能

public class variable04 {

public static void main(String [] args) {

// 整数型と浮動小数点の足し算

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 = "テスト";

String firstName = "太郎";

System.**out**.println(lastName + firstName);

}

}

実行結果

7.5

テスト太郎

final変数

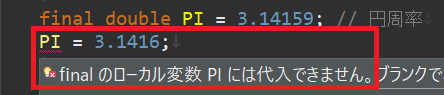
宣言時に代入した初期値を入れ替えられない変数

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

final変数を宣言するときは、通常の変数宣言の頭にfinalをつける

final double PI = 3.14159; // 円周率 2

こうして宣言したfinal変数には、再代入できない、例えば、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る



final変数は繰り返し参照するものの、値を替えたくないデータの扱いに便利

上記にように謝って変えてしまうミスを防げる

またfinal変数を使えばプログラム変更の負担も減らせる

以下では円周率の意味で「3.14159」を２度使っているが、仮に「3.14」に変えるなら２箇所とも変更が必要になる

// 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む

PIが同じものを指している事も明確となる

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);

実際にfinal変数を使ってみる

public class variable05 {

public static void main(String [] args) {

// 円周率

final double PI = 3.14159;

// 円の面積・円周

double circleArea = 0, circumfernce = 0;

// 円の半径

int radius = 5;

// 円の面積を求める

circleArea = PI \* radius \* radius;

System.**out**.println(circleArea);

// 円周を求める

circumfernce = PI \* (radius \* 2);

System.**out**.println(circumfernce);

}

}

実行結果

78.53975

31.4159

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

以下の箇所ではdouble型の変数「circleArea」「circumference」を１行で宣言・初期化をしている

// 円の面積・円周

double circleArea = 0, circumfernce = 0;

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変数を活用すると良い

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」と呼ぶ

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

例えば通常の変数なら「teatTARO」だが、定数なら「TEST\_TARO」とする

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る

変数名の付け方のルール（命名規則）

プログラミングにおいて、名前の付け方のルールを「命名規則」と言う

プログラマー自身が名づける変数にも命名規則は存在する。

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。

ここでは、ごく一般的なルールに従って解説する

大まかに３つある

・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

変数に使える文字、使えない文字

Javaの変数名に使える文字は、以下の三種類となる

これらを組み合わせる事もできる。

・半角英字（a~z A~Z）

・半角数字（0～9）

・アンダースコア（\_）

実は全角文字は日本語を変数名に含める事も不可能ではない

しかし、慣例的にまず使われることはない。はず。

以下の文字は変数名に使えずエラーとなる

・変数名の先頭に数字　3test　など

・アンダースコア以外の記号　test@ $teest　など

・予約語と同盟　boolean final など

予約語とは特別な意味を持つキーワード、例えば「boolean」は論理型のデータを示す為の予約語なので変数名にはできない

仮にbooleanというワードを変数名にできたとしても、Ｊａｖａコンパイラは「論理型だな」と解釈して処理しようとしてしまうのでＮＧ

要は既にデフォで使ってるから、勝手に使うなガキ、ってこと。

変数名はローワーキャメルで記述する

Javaの変数名はローワーキャメルでの記述が推奨されている

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記述する方法のこと。

キャメルケースは二種類ある

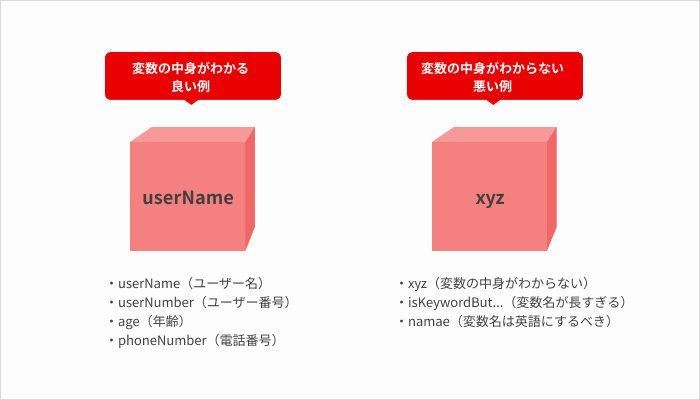


ちなみにPythonでは「user\_name」 「user\_number」のような「スネークケース」が推奨されている

言語によって推奨ルールが変わる事も知っておく。



変数の中身がわかるような名前にする



変数は、その中身が分かるような名前にする

そうでないと、後からソースコードを読むときに「なんの変数だっけ？」のような状況になるため、分かりやすくしておく