

目次

- クラス変数とクラスメソッド
- Java API に現れるクラス変数, クラスメソッド
- 宿題?

★4 クラス変数とクラスメソッド (第4章)**★4.1 クラスとは (p.36)**

オブジェクト指向言語におけるクラスとは、オブジェクトを定義する設計図にあたるものである(☆1)。Javaのプログラムはクラスの集まりであり、あるクラスが他のクラスを利用する形で作られている。このように、オブジェクト指向言語では、クラスを「部品」としてそれを組み合わせることで大きなプログラムも作りやすくなっている(☆2)。

例えば、次頁のT23.javaの例は、T23という名前のクラスを定義するプログラムである。このT23クラスは、TurtleクラスやTurtleFrameクラス、Colorクラスを利用している。これらのクラスはそれぞれ別のプログラムで定義されており、やはり様々なクラスを利用している。

Javaのクラスは、**パッケージ**と呼ばれるより大きな単位にまとめて扱われることがある。TurtleやTurtleFrameはtgパッケージに、Colorはjavafx.scene.paintパッケージに属するクラスである。以前説明したように、このようなクラスを利用するプログラムを作るときは、import文を用いたインポート宣言を行うことで、簡潔に書くことができる(☆3)。

☆1) 第2回講義資料★2.1,2.2も参照

☆2) 作るだけでなく、できたプログラムの保守(デバッグしたり修正したり新機能を追加したり)もしやすい。

☆3) 第2回講義資料★2.7参照

★4.2 クラス変数とクラスメソッド (p.37)

以前の授業で説明したように、インスタンス変数は、個々のインスタンス(オブジェクト)が保持する値である。また、インスタンスメソッドを呼び出すと、特定のオブジェクトに処理を実行させたり状態を変化させたりすることができる。これに対して、個々のオブジェクトに属するのではなく、あるクラスの内でも共有した変数やメソッドがあると便利なことがある。それが、「**クラス変数**」、「**クラスメソッド**」である(☆4)。

クラス変数 API仕様のフィールド(☆5)の欄で先頭にstaticという修飾子がついているもの。TurtleクラスにはwithTurtleAllという名前のクラス変数がある(p.13参照)。個々のオブジェクト毎に異なる値をとり得るインスタンス変数と異なり、実行中のプログラムにおいてそのクラスで1つの値を持つ。インスタンス変数は、生成されたオブジェクト毎にその情報を保持するものだが、クラス変数はクラス全体の情報を表すものであり、そのクラスのオブジェクトが1つも生成されていなくても存在する。

クラスメソッド API仕様のメソッドの欄で先頭にstaticという修飾子がついているメソッド。TurtleクラスにはspeedAllという名前のクラスメソッドがある(p.13参照)。インスタンス変数/クラス変数の違いと同様のことがインスタンスメソッドとクラスメソッドの間にもいえる。

☆4) クラス変数は**静的変数**または**static変数**と言うこともある。クラス・メソッドは**静的メソッド**または**staticメソッド**と言うこともある。

☆5) Javaなどの言語では、クラスのデータを表す変数(インスタンス変数とクラス変数)を**フィールド**と呼ぶことがある。インスタンス変数/クラス変数を**インスタンスフィールド/クラスフィールド**ということもある。

★ 4.3 使ってみよう

以前登場した T23.java(p.19) を改造してクラス変数, クラスメソッドを使ってみよう (p.37 練習問題 4.1 & p.38 練習問題 4.2) .

```
T23.java
1  import tg.*;
2  import javafx.scene.paint.*;
3  public class T23 {
4      public static void main(String[] args){
5          double d = 100, x, y, a;
6          TurtleFrame f = new TurtleFrame();
7          Turtle m = new Turtle(200, 300, 0);
8          f.add(m);
9          m.fd(d);
10         x = m.getX();           // m の X 座標のとり出し
11         y = m.getY();          // m の Y 座標のとり出し
12         a = m.getAngle() - 45; // m の角度のとり出し

13         Turtle m1 = new Turtle(x, y, a); // m1 の作成
14         f.add(m1);
15         m1.fd(d);
16         Turtle m2 = m.clone();          //m2 の作成
17         f.add(m2);
18         m.rt(45);
19         m.fd(d);

20         double newscale = m2.tScale * 4; // m2 の tScale の 4 倍の数
21         m2.tScale = newscale;           // m2 の tScale に代入
22         m2.tColor = new Color(0.0, 1.0, 1.0, 1.0); // m2 の亀の色を水色に変える
23         m2.fd(d);
24         Point p = f.getMousePosition();
25         m2.moveTo(p.x, p.y);
26     }
27 }
```

【注意】 上記のプログラムの例でクラス変数を参照しているところやクラスメソッドを呼び出しているところは、クラス名を使って `Turtle.withTurtleAll` のように書くべきですが、実は `m.withTurtleAll` のように書いても正しく動作します。 `m` が `Turtle` クラスのオブジェクトを指す変数なので、これは `Turtle` クラスのクラス変数だとして正しく処理されるからです。クラスメソッドについても同様のことが可能です。でも、動くからといってそういう書き方をしていると、インスタンス変数とクラス変数、インスタンスメソッドとクラスメソッドそれぞれの違いがわからなくなります。慣れるまではきちんと書き分けることをおすすめします。

★4.4 Java API に現れるクラス変数, クラスメソッド (p.39)

javafx.scene.paint.Color のようなクラスは, Java に標準で用意されたクラスである (☆6). このようなクラスは膨大な数のものがある. Turtle/TurtleFrame クラスについては p.12,13 の API 仕様を参照すればよいが, Java 標準のクラスについては, **Java API** と呼ばれる文書にその仕様が記されているので, これを参照することになる. Java API はウェブページとして公開されている.

以下, Java API に登場する様々なクラスのうち, クラス変数やクラスメソッドをよく使うものをいくつか紹介する.

☆6) 細かいことを言うと, このクラスは JavaFX ライブラリに属するものであるが, JavaFX は標準では入っていない環境もある.

javafx.scene.paint.Color クラス (p.40)

Color クラス (javafx.scene.paint パッケージ内) の API のページへ

この授業のページ → Graphics/Link → JavaFX 11 の API 仕様
→ javafx.graphics モジュール → javafx.scene.paint パッケージ
→ Color クラス

Q1. JavaFX の API 仕様の中からこのクラスの説明を探そう。「コンストラクタ」の項にはどんな説明があるだろう? フィールドの項に様々な色を表す定数があるのがわかる. これらはインスタンス変数だろうかクラス変数だろうか.

Math クラス (p.41)

Math クラス (java.lang パッケージ内) の API のページへのたどり着き方

この授業のページ → Graphics/Link → Java SE 11 の API 仕様
→ java.base モジュール → java.lang パッケージ → Math クラス

java.lang というパッケージに, Math というクラスがある. 数学関係の関数や定数がクラスメソッド/変数として用意されている (☆7). このクラスの正式な名前は java.lang.Math ということになるけれど, java.lang パッケージのクラスは自分で import 宣言を書かなくても自動で import されるので, Math. なんたら のように書くだけで使える.

☆7) 下記の PI のような定数は, final という修飾子をつけることで, 代入不可のクラス変数として宣言されている. p.41 の API 仕様参照.

Math クラスの利用

```
double x = Math.sqrt(2.0); // Math クラスのクラスメソッド sqrt()
System.out.println(x);
double theta = Math.PI; // Math クラスのクラス変数 PI
System.out.println(theta);

x = 30.0;
theta = x*Math.PI/180.0; // theta = π/6
System.out.println(Math.sin(theta));
```

左の実行結果

```
1.4142135623730951
3.141592653589793
0.49999999999999994
```

System クラス (p.42)

System も, java.lang パッケージのクラスである. インスタンスは生成できない. メソッド System.out.print() や System.out.println() はこのクラスに関係している (☆8). この授業では詳しいことは省略する.

☆8) p.41 を見るとわかるように, System.out は System クラスのクラス変数である. print や println は, この変数が表すオブジェクトに対するメソッド呼び出しである.

String クラス (p.43)

Math クラス同様, String も java.lang パッケージのクラスである (☆9). 文字列を表す (☆10). String クラスについてはいろいろ知っておくとよいのだが, この授業では簡単にしか触れない. 当面は以下の説明と例を理解しておこう.

- Java では文字列を String クラスのオブジェクトとして扱う.
- " と " で囲まれた文字列は文字列定数. new 演算子で生成せず使える.
- (文字列) + (文字列) という演算は, 二つの文字列の連結.

☆9) java.lang パッケージは自動的に import される. Math クラスの説明参照.

☆10) C 言語では文字列を char 型の配列に格納していたが, Java では扱い方がずいぶん異なっている.

文字列の扱い**左の実行結果**

```
System.out.println("HOGE!");
System.out.println("What is \"HOGE\" ?");

String s = "hoge", s1 = "fuga";
System.out.println(s);
System.out.println(s + s1); // 文字列に対する+演算は文字列の連結

int x = 10, y = 26;
System.out.println(x); // println が文字列に変換して出力
System.out.println("x = " + x); // +演算の対象のうち一方でも文字列
// なら文字列同士の連結
System.out.println(x + y); // この場合は?
System.out.println(x + "" + y); // この場合は?

// String クラスのインスタンスメソッドの使用例
System.out.println(s + "は" + s.length() + "文字です");

if(s.equals("Hoge")){
    System.out.println(s + "は Hoge に等しいです");
}else{
    System.out.println(s + "は Hoge に等しくありません");
}
```

[発展] 文字列同士の比較には, 上記のように String クラスのインスタンスメソッド equals を用いる. ==で比較してもうまくいかないので注意 (p.107 参照).

宿題?**QUIZ**

Q2. ソース 1 のようなプログラムを実行した場合, 変数 m1 (m ではない) が指すかめは何色ですか.

Q3. ソース 2 の誤りを見つけなさい (いっぱいあります).

次回までに...

今回は教科書第 5 章の内容を説明します. あらかじめ読んでおくこと. 例題のプログラムを作成し実行しておくこと.

```
ソース 1
import javafx.scene.paint.*;
:
Turtle m = new Turtle(), m1 = m;
:
m.setColor(new Color(1.0, 1.0, 0.0, 1.0));
:
```

```
ソース 2
:
TurtleFrame f = new TurtleFrame();
Turtle m = new Turtle();
:
f.add(m);
TurtleFrame.addMesh();
m.fd(100);
m.withTurtleAll = false;
Turtle.tscale = 0.8;
Turtle.moveto(0, 0);
:
```