数理情報セミナー・第 14 回

2017 年 1 月 12 日

今回の内容

14.1 これまでのまとめ

オブジェクト指向プログラミング [第 1 回]

- オブジェクト指向プログラミングとは

  > 特定の役割や機能を持った多数のオブジェクトが、お互いに仕事を依頼し合ったり
  > 情報を交換し合ったりすることで全体が機能する

  という考え方でプログラミングを行うこと。

クラスとクラス宣言 [第 4, 5, 12 回]

- クラスとはオブジェクトの種類のこと。
- クラスは、次のような書式[1]のクラス宣言を行うことで定義する。クラス宣言は、主に、そのクラスのオブジェクトの設計図として働く。

  ```
  class クラス名 {
      各種の宣言の並び
  }
  ```

- クラス宣言の 各種の宣言の並び の部分には、

  1. インスタンス変数 — そのクラスのオブジェクトがそれぞれ保持する変数
  2. インスタンスメソッド — そのクラスの各オブジェクトができる仕事の具体的な手続き (プログラム)
  3. コンストラクタ — そのクラスのオブジェクトの初期化の手続き (プログラム)
  4. クラス変数 — そのクラスに関連してプログラム全体で保持する変数
  5. クラスメソッド — そのクラスに関連した手続き (プログラム)

  などの宣言が記述される[2]。この内、1、2、3 がオブジェクトの設計図として働く。

- `static` 修飾子のない変数宣言やメソッド宣言は、インスタンス変数やインスタンスメソッドの宣言となり、`static` 修飾子付きの変数宣言やメソッド宣言は、クラス変数やクラスの宣言となる。

---

[1] `class` の前には、`public`、`abstract`、`final` などの修飾子が、クラス名 と `{` の間には、(後述の) `extends` 節や `implements` 節が置かれることがある。

[2] この他にも、メンバクラスやメンバインタフェース、インスタンス初期化子、クラス初期化子を宣言することができる。

インスタンス [第 1, 2 回]

- クラス宣言に基づいて生成されたオブジェクトを、そのクラスのインスタンスと呼ぶ。
- インスタンスの生成は、次の書式のインスタンス生成式で生成する。

  `new` クラス名 (コンストラクタの引数の列)

- それぞれのインスタンスは、そのクラスで宣言された (継承したものを含む) インスタンス変数を内部に保持する。
- インスタンス変数には、次の書式でアクセスできる。

  オブジェクトを表す式 . インスタンス変数名

  ただし、オブジェクトを表す式 が `this` の場合[3]、`this.` は省略することもできる。
- それぞれのインスタンスは、そのクラスで宣言された (継承したものを含む) インスタンスメソッドを実行することができる。
- コンストラクタやインスタンスメソッドの宣言の中で、初期化されようとしている、あるいはそのインスタンスメソッドを実行しようとしているインスタンスを `this` で参照することができる。
- インスタンスメソッドは、次の書式のメソッド起動式を使って、ターゲットとなるオブジェクト (インスタンス) を指定した上で起動する。

  オブジェクトを表す式 . インスタンスメソッド名 (引数の列)

  ただし、オブジェクトを表す式 が `this` の場合、`this.` は省略することもできる。

スーパークラスとサブクラス [第 6 回]

- 既存のクラスを元にして、新しいクラスを宣言することができる。
- 元になったクラスを、新しいいクラスの (直接の) スーパークラスと呼び、新しいクラスは、元となったクラスの (直接の) サブクラスと呼ばれる。
- そのクラスの直接のスーパークラスは、クラス宣言の クラス名 の後に `extends` 節で宣言する。
- 新しいクラスの宣言では、インスタンス変数やインスタンスメソッドを追加したり、インスタンスメソッドを再定義したりすることができる。
- 再定義しない限り、元のクラスの変数やメソッドはサブクラスに継承される。
- Java では、複数のクラスを継承したクラスを宣言すること (多重継承) はできない。
- コンストラクタは継承されないが、必ずスーパークラスのコンストラクタが起動された後に、サブクラスのコンストラクタ本体が実行されるようになっている。

[3] かつ、仮引数や局所変数と名前が衝突しない場合。

- 新しいクラスで再定義される前のメソッド (直接のスーパークラスでの) 定義を、次の書式で起動することができる。

  `super.` メソッド名 `(` 引数の列 `)`

インタフェース [第 9 回]

- インタフェースとはオブジェクトの資格のこと。
- インタフェースはインタフェース宣言を行うことで定義される。
- インタフェース宣言では、抽象メソッド (どのような型の引数をもらって、どのような型の戻りを返す、どのような名前のメソッドなのか) の宣言を行う[4]。
- 既存のインタフェースを元にして、新しいインタフェースを宣言することもできる。
- あるクラスのオブジェクトが、あるインタフェースが要求する条件を満たすとき、そのクラスはそのインタフェースを実装していると言う。
- 1 つのクラスが複数のインタフェースを実装することもできる。
- そのクラスが実装しているインタフェース群は、クラス宣言の `extends` 節の後 (なければ クラス名 の後) に、`implements` 節を使って宣言する。

配列 [第 3 回]

- Java の配列はオブジェクトの一種である。
- 配列は、次の書式の配列生成式で生成する。

  `new` 要素の型名 `[` 要素の個数 `]`

- 配列オブジェクトは、`length` という `int` 型のインスタンス変数を持つ。
- Java では、多次元の配列を、配列型を要素とする配列で表現する。

参照型 [第 3 回]

- Java ではオブジェクトを、すべて参照で扱う。
- 値を参照で扱うデータ型を参照型と呼ぶ。
- どのオブジェクトも指していないことを示す値を null 参照と呼ぶ。
- クラス C を宣言すると、C は参照型の 1 つとなる。C 型は、そのクラスとそのサブクラスのインスタンスへの参照、および null 参照からなる型となる。
- インタフェース I を宣言すると、I は参照型の 1 つとなる。I 型は、I を実装しているすべてのクラスのインスタンスへの参照、および null 参照からなる型となる。
- T 型の要素持つ配列の型を T`[]` で表す。T`[]` は参照型の 1 つとなる。

[4] 他にも、定数 (final なクラス変数) やメンバクラス、メンバインタフェースの宣言を行うことができる。

動的ディスパッチ [第 6 回]

- インスタンスメソッドの起動では、通常[5]、そのときターゲットとなったオブジェクト (のクラス) が持つメソッドの定義が用いられる。
- つまり、プログラム中の同じメソッド起動式で起動されるメソッドの定義が、起動の度に異なる場合がある。
- このように、プログラムの実行時に (メソッドが起動される度に)、そのとき使われるメソッド定義が選ばれることを動的ディスパッチと呼ぶ。

クラス変数とクラスメソッド [第 5 回]

- C 言語の大域変数に相当するものとして、Java にはクラス変数がある。
- クラス変数は、プログラム全体に 1 つだけ存在する。
- クラス変数には、次の書式でアクセスできる。

  クラス名 . クラス変数名

  ただし、同じクラスのクラス変数へアクセスする場合[6]、クラス名 . を省略することもできる。
- C 言語の関数に相当するものとして、Java にはクラスメソッドがある。
- クラスメソッドは、次の書式のメソッド起動式で起動する。

  クラス名 . クラスメソッド名 (引数の列)

  ただし、同じクラスのクラスメソッドを起動する場合、クラス名 . を省略することもできる。

例外 [第 10 回]

- プログラムの実行中に発生する予期しない出来事を例外と呼ぶ。
- 例外の発生を検知して、それに対する処置を行うことを例外処理と呼ぶ。
- Java では、例外が発生すると、例外を表すクラスのインスタンスが生成され、そのオブジェクトがスローされる。
- Java では `try` 文とその `catch` 節を用いて、スローされた例外をキャッチし、例外処理を行うことができる。

スレッド [第 13 回]

- プログラムの一筋の実行の流れをスレッドと呼ぶ。
- Java では、複数のスレッドを同時並行的に実行することができる。

---

[5] `super.` を使ったインスタンスメソッドの起動は除く。

[6] かつ、仮引数や局所変数と名前が衝突しない場合。

- プログラムの特定の部分が、同時に複数のスレッドで実行されないようにすることを排他制御と呼ぶ。
- Java では、synchronized 文や、メソッド宣言に対する synchronized 修飾子を用いて排他制御を行うことができる。

基本的なクラス [第 7 回]

- すべてのオブジェクトのスーパークラスとして Object クラスがある。
- 配列型オブジェクトも Object 型である。
- Java では、文字列を String クラスのインスタンスで表す。
- Java の + 演算子は文字列を連結することができる。
- Java のすべての値は、文字列 (String のインスタンス) に変換できる。
- 三角関数、指数関数、対数関数など基本的な数学関数が Math クラスのクラスメソッドとして用意されている。
- System クラスを利用すると、標準入力、標準出力、標準エラー出力、現在時刻などにアクセスできる。
- プリミティブ型 (boolean、char、byte、short、int、long、float、double) のそれぞれに対応するラッパクラス (Boolean、Character、Byte、Short、Integer、Long、Float、Double) があって、プリミティブ型の値をラッパクラスのインスタンスとして扱うこともできる。

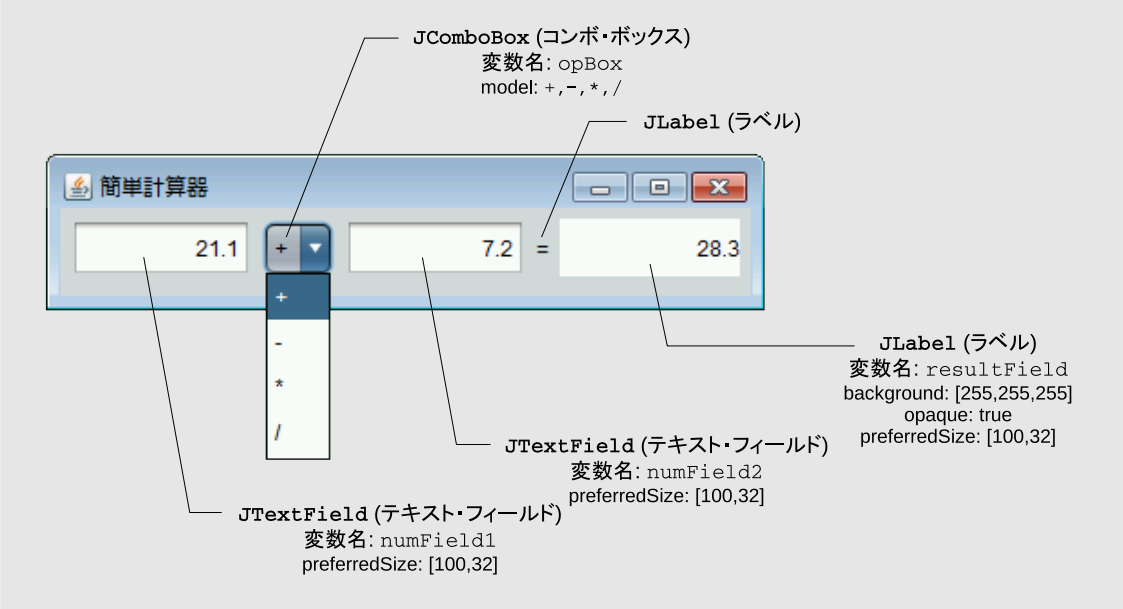
JFC による GUI の構築 [第 8, 9, 11 回]

- Java には、Java Foundation Classes (JFC) と呼ばれる、グラフィカルユーザーインタフェイス (GUI) を構築するための枠組み (クラス群) が用意されている。
- JFC のクラスライブラリとして、Swing と呼ばれる GUI 部品のセットが提供されている。
- GUI アプリケーションのウィンドウやその内部の GUI 部品に対するユーザーの操作 (キーボード操作、マウス操作など) を、一般にイベントと呼び、イベントの発生に対するプログラム側の処理をイベント処理と呼ぶ。
- JFC では、イベント処理や、再描画の処理をイベントディスパッチスレッドど呼ばれるスレッドで行う。
- Swing に含まれる GUI 部品が描画される際には、そのオブジェクトをターゲットとして、paintComponent というメソッドが起動されるので、これを再定義することで、そのクラス独自の描画を行うことができる。
- JFC では、GUI の各部品に対してイベントハンドラ (イベントリスナ) を登録しておくことでイベント処理を行う。
- 各イベントハンドラの資格は ActionListener や MouseListener などのインタフェースとして定義されており、発生したイベントが ActionEvent や MouseEvent などのクラスのイ

ンスタンスとして表現され、そのオブジェクトを引数として、イベントハンドラの特定のインスタンスメソッド (actionPerformed など) が起動される。

14.2 演習問題

1. NetBeans IDE を用いて、次の図のような電卓プログラム Calc.java を作成しなさい。



ただし、Calc クラスは JFrame のサブクラスとし、タイトル (title) を「簡単計算器」としてください。また、numField1 や opBox、numField2 で ActionEvent が発生した時に、次のようなメソッドを起動して、resultField が更新されるようにしてください。

```
void doCalc() {
    try {
        double x = Double.parseDouble(numField1.getText());
        double y = Double.parseDouble(numField2.getText());
        double r = 0.0;
        String s = (String) opBox.getSelectedItem();
        if ("+".equals(s)) {
            r = x + y;
        } else if ("-".equals(s)) {
            r = x - y;
        } else if ("*".equals(s)) {
            r = x * y;
        } else if ("/".equals(s)) {
            r = x / y;
        }
        resultField.setText("" + r);
    } catch (NumberFormatException e) {
        resultField.setText("Error!");
    }
}
```

14.3 付録：Java のキーワード類一覧

`abstract` — 抽象クラス、あるいは抽象メソッドの宣言であること示す修飾子 [第 9 回 4 ページ, 第 11 回 10 ページ]

`boolean` — 真理値の型 (プリミティブ型の一つ) [第 3 回 1 ページ]

`break` — `break` 文の始まり (`break` ラベル `;` の書式で、ラベル を持つ文を `break` することもできる)

`byte` — 8 bit の符号付き整数の型 (プリミティブ型の一つ) [第 3 回 1 ページ]

`catch` — `try` 文の `catch` 節の始まり [第 10 回 7 ページ]

`char` — UTF-16 の文字コードでの 1 符号単位 (16 bit の符号付き整数) の型 (プリミティブ型の一つ) [第 3 回 1 ページ]

`class` — クラス宣言の始まり[7] [第 4 回 1 ページ]

`const` — 使用できない

`continue` — `continue` 文の始まり (`continue` ラベル `;` の書式で、ラベル を持つ `while` 文、`for` 文、`do` 文を `continue` することもできる)

`default` — `switch` 文内の `default:` ラベル

`do` — `do` 文の始まり

`double` — 64 bit の浮動小数点数の型 (プリミティブ型の一つ) [第 3 回 1 ページ]

`else` — `if` 文の `else` 節の始まり

`extends` — クラス宣言の `extends` 節 (そのクラスの直接のスーパークラスを指定する) の始まり [第 6 回 2 ページ]

`false` — `boolean` 型で偽を表すリテラル [第 3 回 1 ページ]

`final` — そのクラスのサブクラスが宣言できないこと、その変数が (一旦初期化したら) 値を変えられないこと、あるいは、そのメソッドが (サブクラスで) 再定義できないことを示す修飾子 [第 3 回 5 ページ, 第 6 回 2 ページ]

`finally` — `try` 文の `finally` 節の始まり [第 10 回 8 ページ]

`float` — 32 bit の浮動小数点数の型 (プリミティブ型の一つ) [第 3 回 1 ページ]

`for` — `for` 文の始まり

`goto` — 使用できない

`if` — `if` 文の始まり

`implements` — クラス宣言の `implements` 節 (そのクラスが実装するインタフェース群を指定する) の始まり [第 9 回 5 ページ]

`import` — インポート宣言 (特定のパッケージに属するクラスやインタフェース、あるいは、それらのクラス変数やクラスメソッドなどを単純名で参照する設定) の始まり [第 5 回 7 ページ]

---

[7] 型名 `.class` という書式の式で、型名 に対応する `java.lang.Class` クラスのインスタンスを表すのにも使われます。

`instanceof` — 式 `instanceof` 参照型名 という書式の式を書いて、式 を評価した結果が 参照型名 で指定した型のオブジェクトであるかどうかを `boolean` 型で表す演算子 (式 が null 参照の場合は `false` となる)

`int` — 32 bit の符号付き整数の型 (プリミティブ型の一つ) [第 3 回 1 ページ]

`interface` — インタフェース宣言の始まり [第 9 回 4 ページ]

`long` — 64 bit の符号付き整数の型 (プリミティブ型の一つ) [第 3 回 1 ページ]

`native` — そのメソッドが Java 以外の言語で実装されていることを示す修飾子

`new` — クラスのインスタンスや配列オブジェクト生成する式を始める [第 1 回 6 ページ, 第 3 回 4 ページ]

`null` — null 参照を表すリテラル [第 3 回 2 ページ]

`package` — パッケージ宣言 (このソースファイルで宣言されるクラスやインタフェースが属するパッケージの指定) の始まり [第 5 回 6 ページ]

`private` — メンバクラス、メンバインタフェース、インスタンス変数、インスタンスメソッド、コンストラクタ、クラス変数、クラスメソッドの宣言が、それを囲んでいるクラス宣言内だけでアクセスできることを示すアクセス修飾子 [第 6 回 10 ページ]

`protected` — メンバクラス、メンバインタフェース、インスタンス変数、インスタンスメソッド、抽象メソッド、コンストラクタ、クラス変数、クラスメソッドの宣言が、同じパッケージ内と、サブクラスの宣言内からアクセスできることを示すアクセス修飾子 [第 6 回 10 ページ]

`public` — クラス、インタフェース、インスタンス変数、インスタンスメソッド、抽象メソッド、コンストラクタ、クラス変数、クラスメソッドの宣言が、プログラムのどこからでもアクセスできることを示すアクセス修飾子 [第 6 回 10 ページ]

`return` — `return` 文の始まり

`short` — 16 bit の符号付き整数の型 (プリミティブ型の一つ) [第 3 回 1 ページ]

`static` — その宣言が、クラス変数、クラスメソッド、クラス初期化子、あるいは静的メンバクラスの宣言であることを示す修飾子[8] [第 5 回 4 〜 5 ページ]、あるいは、`import` 宣言の中で、これら (クラス初期化子を除く) を単純名で参照することを示す [第 5 回 8 ページ]

`strictfp` — そのクラス宣言、インタフェース宣言、メソッド宣言では、`IEEE 754` 規格に厳密に従った方法で浮動小数点数の計算を行わなければならないことを示す修飾子

`super` — コンストラクタ本体の先頭で、直接のスーパークラスのコンストラクタを起動する、あるいは、コンストラクタやインスタンスメソッド内などで、スーパークラスの変数やメソッドにアクセスするために `this` の代りに使用する [第 5 回 4 ページ]

[8] メンバインタフェースの宣言にも付加できますが意味は変りません。

`switch` — `switch` 文の始まり

`synchronized` — `synchronized` 文 (特定のオブジェクトをロックした上で実行する文) の始まり、あるいは、ターゲットとなるオブジェクトのロックを獲得した上でメソッド本体を実行することを示す修飾子 [第 13 回 8 ページ]

`this` — コンストラクタ本体の先頭で、同じクラスの他のコンストラクタを起動する [第 5 回 2 ページ]、あるいは、コンストラクタやインスタンスメソッド本体[9]で、初期化の対象となっているオブジェクトやメソッド起動の対象 (ターゲット) となっているオブジェクトを表す式 [第 4 回 5, 7 ページ]

`throw` — `throw` 文 (例外をスローする文) の始まり [第 10 回 6 ページ]

`throws` — コンストラクタやメソッド宣言の `throws` 節の始まり [第 10 回 7 ページ]

`transient` — そのインスタンス変数やクラス変数が記憶するのは一時的な値のみで、そのインスタンスやクラスの状態の表現に関わっていないこと (状態を保存する際に記録する必要がないこと) を示す修飾子

`true` — `boolean` 型で真を表すリテラル [第 3 回 1 ページ]

`try` — `try` 文の始まり [第 10 回 7 ページ]

`void` — メソッド宣言の戻り値の型の位置に書いて、そのメソッドに戻り値がないことを示す

`volatile` — 他のスレッドによる代入も含めて、常に最新の値を参照しなければならないインスタンス変数またはクラス変数であることを示す修飾子 [第 13 回 11 ページ]

`while` — `while` 文の始まり、あるいは `do` 文の `while` 節の始まり

数理情報セミナー・第 14 回・終わり

[9] あるいは、インスタンス初期化子やインスタンス変数の初期化子内。