

プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 8

オブジェクト指向

Object Orientation

Object Oriented (OO) Programming Languages

質問: オブジェクト指向プログラミングとは何か?

オブジェクトを持つプログラムを作成すること?

オブジェクトを構成要素としたプログラムを作成すること (制御あるいは手続きを構成要素としたプログラムではない)?

「オブジェクト」とは何か?

プログラミングスタイル(プログラムの構成方法)

コントロール指向のプログラミングスタイル:

プログラム構成における主要な関心事が「制御」であるもの
一連の処理 (いろいろなデータを操作する) をまとめて記述したもの
データ指向のプログラミングスタイル:

プログラム構成における主要な関心事が「データ」であるもの
一種類のデータ (いろいろな処理において操作される) をまとめて記述したもの

例: スタックの操作をひとまとまりにしたもの (スタック抽象データ型の実装)

プログラミング言語との関係:

C 言語でもオブジェクト指向のプログラミングは可能。

ただし、オブジェクト指向をサポートする機能は (ほとんど) ない。

Java 言語でもコントロール指向のプログラミングは可能。(全ての操作を main クラスに押しこめる、など)

ただし、コントロール指向では非常に書きにくい。

オブジェクトとは?

それに所属するデータたちと、それを操作する関数たちをまとめて「1つ」にしたもの。

操作する関数: メソッド (method, member function)

所属するデータ: インスタンス変数 (instance variable, field, data member)

オブジェクトのインタフェース: メソッドのうち公開されているもの (public) の名前や型。

オブジェクトの実装: メソッドやインスタンス変数の具体的な実現方法。

クラス: オブジェクトの「型」のようなもの。(ただし、「クラス」という概念がない OO 言語もある。例 JavaScript)

Dynamic lookup 動的ルックアップ

Abstraction 抽象化

Subtyping サブタイピング (部分型付け)

Inheritance 継承

引用元: J. C. Mitchell, "Concepts in Programming Languages", 2003.

`foo.add(e)`

オブジェクト `foo` に `add(e)` というメッセージを送信。

オブジェクト `foo` が持つ `add` という名前のメソッドを、`e` という引数で起動。

起動されるメソッドは、オブジェクト `foo` ごとに決まる。

プログラム上では同じ変数 `foo` であっても、あるときは整数オブジェクト、別のときは、集合オブジェクトかもしれない。

起動される `add` メソッドは、実行の時点ごとに (変数 `foo` の値となるオブジェクトごとに) 異なり得る。

Lookup とは？

メソッドの名前 から、実際に起動されるべき**メソッドの実装**を得ること。

cf. 変数名から、(現在の環境における) その変数の値を得る。

ルックアップが動的 (dynamic) であるとは？

ルックアップの結果は、静的に決まるのではない。

実行時に決まる。

静的ではなく、動的なルックアップは、プログラミング上、極めて有用。
例: グラフィクスプログラムにおいて、四角形、円、三角形などの図形オブジェクトごとに `draw` メソッドを用意。

抽象データ型における Abstraction と同様。

オブジェクトへのアクセスは、インタフェース関数(メソッド)のみに限定される。

実装と仕様(インタフェース)の分離を達成。

型 A が型 B の subtype(部分型) のとき、型 B の式を書くべきところに、型 A の式を書いても良い。[代入可能性]

```
class Point {  
    ...  
    ... void move (int dx, int dy) { ...}  
}  
class Circle extends Point {  
    ...  
    ... void move (int dx, int dy) { ...}  
}
```

Point クラスのオブジェクトに対する操作は、Circle クラスのオブジェクトに対しても適用できる。

OO 言語では:

move メソッドが Point オブジェクトにも Circle オブジェクトにも適用可能。

move メソッドは、Point クラスを継承した任意のクラスのオブジェクトに対して適用可能。

一種の多相性 (subtyping polymorphism ⇔ ML 言語の parametric polymorphism)

継承によるコード再利用

```
class Point {  
    private int x = ...;  
    public int getX() {...};  
    ...  
}  
class CPoint extends Point {  
    private int c;  
    public int getC() {...};  
    ...  
}
```

プログラマは、1つのコードを2回書かない。
処理系内部でも、1つのコードを2重に持たない。

これらの違いは何か?

subtyping: 2つのオブジェクト(やクラス)の**インタフェース**の間の関係。

inheritance: 2つのオブジェクト(やクラス)の**実装**の間の関係。

いくつかの OO 言語 (C++ など) では、両者は緊密な関係にあるが、一般的には、必ずしも一致しない。(継承関係にある2つのクラスが、subtyping の関係にないことがある、等。)

Simula [1960 年代, K. Nygaard]

Smalltalk [1970 年代, Xerox PARC 研究所, Alan Kay]

C++ [1984-, Stroustrup]

Java [1990-, Gosling]

Ruby [1993-, Matsumoto]

関数(手続き)指向 vs オブジェクト指向
デザインパターン

module と object の比較:

基本的な違い: module は内部状態(OO 言語のインスタンス変数)を持たない。

抽象化: 同じ。

関数のルックアップ: module は静的, object は動的。

継承: module に継承はないが、実装の再利用は可能。

サブタイピング: module にはサブタイピング機能はない。

オブジェクト指向の4つの基本概念
モジュールとの共通点、相異点

前回の課題に関して

```
class Point { ...
    public String toString () {
        return "Point...";
    }
}
class ColoredPoint extends Point { ...
    public String toString () {
        return "ColoredPoint...";
    }
}
```

Override (上書き):

親クラス (Point) を継承した子クラス (ColoredPoint) では、メソッド `toString` の定義 (実装) をそのままもらうのではなく、違うものにかきかえている。

`toString` の引数の個数、型、返すものの型は、まったく同じ。

質問 1. 「動的ルックアップ」とは何か、説明せよ。

質問 2. Java では、変数束縛は静的である一方で、`method` のルックアップは動的である。なぜそのような設計が良いのか、考えなさい。

Override in Java

```
class Test1 {
    public static void main(String args[]) {
        Point p = new Point(10.0, 20.0);
        ColoredPoint cp = new ColoredPoint(10.0, 20.0, 3);

        System.out.println(p.toString()); => 親の toString
        System.out.println(cp.toString()); => 子の toString
    }
}
```

Override with Cast in Java

(1) 親クラスの変数に、子クラスのオブジェクトを代入してもよい。

```
class Test1 {
    public static void main(String args[]) {
        ColoredPoint cp = new ColoredPoint(10.0, 20.0, 3);
        Point p = cp;
        System.out.println(p.toString()); => 子の toString
```

cp.toString() について、動的にルックアップしている。

(2) 子クラスの変数に、親クラスのオブジェクトを代入するのはいけない。

```
class Test1 {
    public static void main(String args[]) {
        Point p = new Point(10.0, 20.0);
        ColoredPoint cp = p; => コンパイルエラー
```

Override in Java

(型に関する) インタフェースは継承。
実装は書き換える。

Overload in Java

```
class Test1 {...
    public static void foo(Point p) {
        System.out.println("foo-1:" + p.toString());
    }
    public static void foo(ColoredPoint cp) {
        System.out.println("foo-2:" + cp.toString());
    }
    public static void foo(Point p, ColoredPoint cp) {
        System.out.println("foo-3:" + p.toString() + ":" + cp.toString());
    }
}
```

Overload:

1つのメソッド名に複数の実装。

引数の个数、引数の型、返す値の型で区別。

Overload with Cast in Java

親クラスの変数に、子クラスのオブジェクトを代入してもよい。

```
Point p = new ColoredPoint(...);
foo(p);          ==> foo-1 が呼ばれる。
```

変数 p の中には、子クラスのオブジェクトであるが、どの foo が呼ばれるかは、静的に (変数の型等で) 決定されるため、ここでは foo-2 でなく foo-1 が呼ばれる。

(一方、foo の中で呼ばれる toString は動的に決定されるため、子クラスの toString が呼ばれる。

```
class Test4 {
    public static void foo(Point p) {
        System.out.println("foo-1:" + p.toString());
    }
}
class Test5 extends Test4 {
    public static void main(String args[]) {
        Point p = new Point(10.0, 20.0);
        ColoredPoint q = new ColoredPoint(10.0, 20.0, 5);
        Point r = q;
        foo(r);    ==> 何が返るか?
    }
    public static void foo(ColoredPoint cp) {
        System.out.println("foo-2:" + cp.toString());
    }
}
```

Java は静的型付きオブジェクト指向言語

Override: 親クラスと子クラスで、同じ名前・型で実装が異なるメソッド (この場合は toString メソッド) を持つこと。

「どの型 (クラス) の変数か」ではなく、「実行時に、その変数にどのクラスのオブジェクトがはいっているか」によって、使われるメソッドが決まる (**動的ルックアップ**)。

Overload: 同一クラス内で1つのメソッド名に、複数の定義を与えること。

複数の定義は、引数パターン (型, 個数, 順番) が異なる。どのメソッド定義が使われるかは、メソッド呼び出しの引数パターンにより (**つまり静的に**) 決定される。