プログラム言語論-第 8 週-

亀山幸義

筑波大学コンピュータサイエンス専攻

筑波大学 情報科学類 講義, 2009 年 6 月 8 日

概要

前回まで
- プログラムの構文
- プログラムの意味と評価 (処理) 方法
- 型システム、抽象データ型: (情報隠蔽、モジュラリティ)

今回
- (2 限) オブジェクト指向の基本概念
- (3 限) miniML 演習

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？

オブジェクトとは？

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 (かつ、後に述べる条件を満たすもの)。
- オブジェクトの関数たち: メソッド (method, member function)
- オブジェクトのデータたち: 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。

オブジェクト指向の基本概念

Mitchell による 4 つの基本概念
- Dynamic lookup 動的検索
- Abstraction 抽象化
- Subtyping サブタイピング (部分型付け)
- Inheritance 継承

J. C. Mitchell, “Concepts in Programming Languages”, 2003.

Dynamic Lookup-1

ここでいう Lookup とは？
- オブジェクトに送られたメッセージ (に含まれるメソッドの名前) 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 (オブジェクトの実装ごとに)、決まる。

Dynamic Lookup-2

```
x -> add(y)
```

- オブジェクト x に add(y) というメッセージを送信。
- オブジェクト x が持つ add という名前のメソッドを、y という引数で起動。
- 起動されるメソッドは、オブジェクト x ごとに決まる。
- プログラム上では同じ変数 x 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 x の値となるオブジェクトごとに) 異なり得る。

Dynamic Lookup-3

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。

抽象データ型と Dynamic Lookup

```
abstype matrix =
 with create(..) = ..
      update(m,i,j,x) = ...
      add(m1,m2) = ...
 end
add(x,y)
-----------------------------------------
class matrix
 (representation)
 update(i,j,x) = set (i,j) of *this* matrix
 add(m) = add m to *this* matrix
x -> add(y)
```


Abstraction

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。

Subtyping

- 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]
- 複数の型に同じ操作が適用できるようになる。
- 例: airplane クラスに対する操作は、Boeing 757 クラスに対しても適用できる。

`A <: B` (A は B の subtype)

Inheritance

継承によるコード再利用

```
class A =
  private val v = ...
  public fun f(x) = ..g(..)...
         fun g(y) = ...old definition...
  end;
class B = extend A with
  private val w = ...
  public fun g(y) = ...new definition...
  end;
```

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。

Subtyping vs Inheritance

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。)

大規模ソフトウェアの設計

- 関数 (手続き) 指向 vs オブジェクト指向
- デザインパターン

OO 言語たち

- 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]