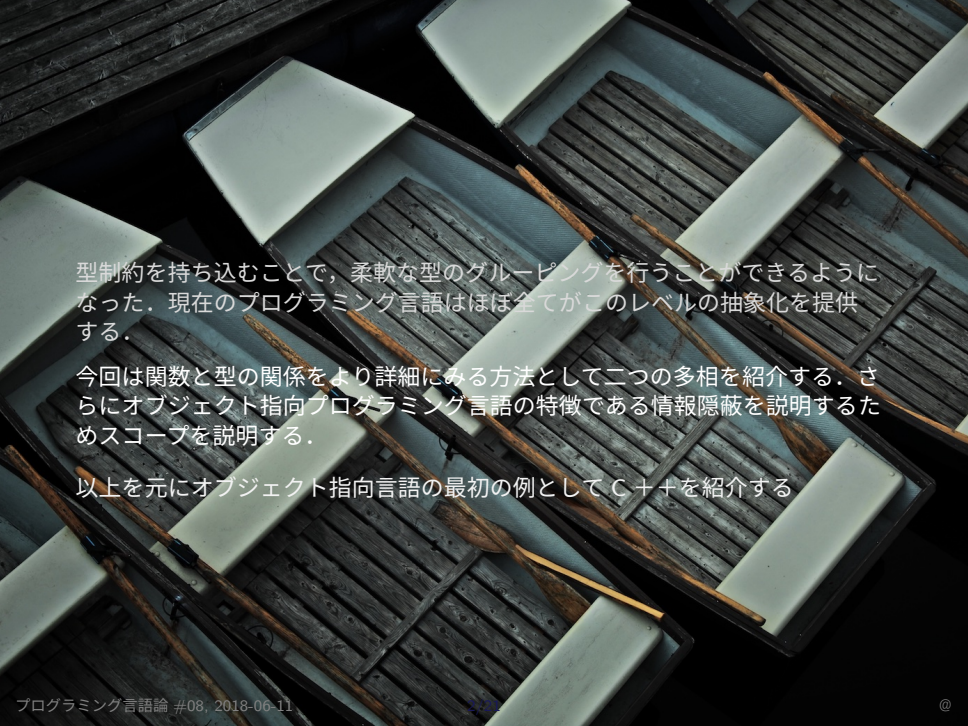


プログラミング言語論 #08

2 Polymorphisms, C++

2018-06-11





型制約を持ち込むことで、柔軟な型のグルーピングを行うことができるようになった。現在のプログラミング言語はほぼ全てがこのレベルの抽象化を提供する。

今回は関数と型の関係をより詳細にみる方法として二つの多相を紹介する。さらにオブジェクト指向プログラミング言語の特徴である情報隠蔽を説明するためスコープを説明する。

以上を元にオブジェクト指向言語の最初の例としてC++を紹介する

プログラミング言語論専門用語

8

polymorphism, 多相

[▶ wikipedia](#)

型が一意に定まらないこと．例えば

- 型推論の結果，関数の引数や返値，変数などが特定の型に定まらない場合
- すなわち，型変数を使った型宣言を持つ変数や関数

parametric polymorphism, parametric 多相である

ある1つの関数定義が，型変数（任意であっても制約が付いていてもよい）で表される引数や返値を持ち，その結果，引数や返値に複数の型が許されること

Haskell における parametric 多相な関数の例

- 引数は多相なのに返値が多相でない関数の例

```
1 length ::  
2 length [] = 0  
3 length (_,l) = 1 + length l
```

引数が多相なので関数 `length` は parametric 多相な関数である。

- 引数は多相ではないのに返値が多相になっている関数の例

```
1 f ::  
2 f x = (x == 1,
```

返値が多相なので関数 `f` は parametric 多相な関数である。

プログラミング言語専門用語

9

ad-hoc polymorphic, ad-hoc 多相（である）

ある関数が型に応じて選択される複数の定義（実装）を持つこと。

- parametric 多相な関数：定義は**一つ**のみ。これで複数の型の引数（または返値）に対応できる。
- ad-hoc 多相な関数：定義が引数の型ごとに存在する。従って定義は**複数**。

ただし、どちらにおいても名前は一つであり、型（宣言）も**一つ**。多相は「複数の型を持つ」のではない。

▶ 部分点が貰えない説明

Haskell での例/1

型クラスに対する ad-hoc 多相

```
1 class C2 a where
2   test :: a -> String      -- testに対する唯一の型宣言
3   -- 一方、定義はない
4
5 instance C2 Int where
6   test x = "int"          -- Int引数に対するtestの定義
7   -- 一方、型宣言はない
8
9 instance C2 Bool where
10  test x = "bool"         -- Bool引数に対するtestの定義
11  -- 一方、型宣言はない
```

- test の型宣言は一つ (L2)
- 定義は型クラスに含まれる型に対して一つずつ、全体で複数

従って parametric 多相ではなく、ad-hoc 多相な関数である。

Haskell での注意点

ad hoc 多相な関数は instance 文の中でしか定義できない

同じ名前の関数が複数存在するのでエラーになる例

```
1 test :: Int -> String
2 test x = "int"
3
4 test :: Bool -> String    // まず型に関する2重宣言エラー
5 test x = "bool"         // さらに2重定義エラー
6
7 test :: String -> String  // 以下同様
8 test x = "string"
```

適切な型クラスを定義し，その class 文中で関数 test の型宣言をし， instance 文中にこれらの定義を移せばエラーはなくなる。

多相の区別

関数 length はどちらの多相か？

どちらかが必要：

- length の定義を調べる
- 自分で同等の関数を定義してみる

今回は第4シートで length の定義を示したので（自分で定義することなく）理由を説明できる．

以下の状況で (+) はどちらの多相か？

```
1 instance Num Bool where
2   ...
3 instance Num GHP where
4   Goo + Goo = Goo
```


型階層モデルの考え方

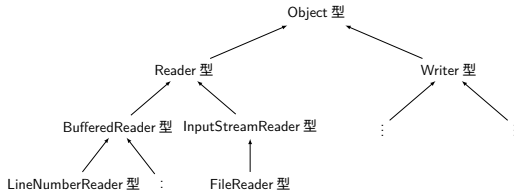
- 必要に応じて型をまとめた型を作ればよい
- 実際には型を定義する時にどの型に含めるかを宣言する
- 個別の型ではなくまとめた型を引数に指定すれば、その関数はまとめられた型のどれでも受け付ける
- 型をまとめたものをさらにまとめることもできるので、どういう関係でも表現できるはず

C++, Java, JavaScript, Python では型階層モデルを採用。
このような言語では型を **class**, **クラス**と呼ぶ。

	基本概念名	上位概念名	備考
C	型	なし	古い
Haskell	型	型クラス	型 ∈ 集合 2 層モデルを採用
他言語	クラス	必要なし	型階層モデルを採用

Java ▶ Java 8での例

- データを入力できる対象，あるいは書き込める対象
- その中で，データをまとめ読み，あるいは先読みできるもの
- さらに，行番号を考えながらデータをまとめ読みできるもの．．．



対応する，型（クラス）を包含関係も含めて定義する Java コード

```
1 class Reader extends Object .... // Readerクラス
2 class BufferedReader extends Reader ... // これもクラス
3 class InputStreamReader extends Reader ... // これもクラス
```

Python の例

定義のキーワードは **class**

定義時に何型に含まれるかを指定

```
1 # 新しいクラスSampleClassを定義して、objectクラスに包含せざる
2 class SampleClass (object): ...
3
4 class Integer (SampleClass): ...
5
6 # 数値の例
7 class Number:                # ここで加減乗を定義
8 class Float (Number): ...    # ここで(/)を定義
9 class Integer (Number): ...  # ここで(%)を定義
10 class LongInteger (Integer): ... # 特殊な整数も整数の一種
```

これで引数が Integer のときにしか使えない関数や引数が LongInteger のときにしか使えない関数を定義できる

プログラミング言語論専門用語

10

Object-Oriented Programming; OOP

[▶ wikipedia](#)

ad-hoc 多相に加え、いくつかの便利な機能を追加したプログラミングの考え方。現在の主流となっている。以下の特徴を持つことが要求されている。

- カプセル化, 情報隠蔽 – スコープの話 (後述)
- inheritance; 継承 – 型の中の再帰的包含関係
- polymorphism; 多相性 (特に ad-hoc 多相性)

この考え方に強く影響された言語は**オブジェクト指向プログラミング言語 OOPL**と呼ばれる。

ad-hoc 多相によって、ものごとの表現と処理をうまくプログラムに実装することができる。すなわち、汎用性を見つけたら、同じ機能を使い、一つの定義にまとめる (parameteric)。用途を見つけたら、同じも別の用途が実現には違ってくる。ad-hoc の機能を使う。この考え方を進める、簡単に言えば、X 型による分岐を組み合わせていて複雑なことをさせるのではなく、あれが XX 型で私が YY 型だから、この計算を実行するというようなプログラミングが実現になる (X 型がどこかに付いた)。一方、C 型では全てをプログラムで表現せよ。コンピュータがしなやかにならないことは全て数値計算、せいぜいサートマシンとする。これは非常に原始的で、間違いに気付かなく、拡張が困難である (「プログラムの設計や膨大なエッセンスの表のヤルを全部覚えること」みたいな感じ)。そこで大規模で、複雑で、多人開発で、変更が頻発するようなプログラミングの環境では、前者の、より堅牢なプログラミングが使われるようになってきた。その一つが Object-oriented programming、オブジェクト指向プログラミングである。プログラミング対象を数値ではなく型に実装させる流れの現在の主流である。

C++

- コンパイラ: g++
- 拡張子: cpp または C

確認事項:

- ① C との類似性
- ② 関数呼出しが拡張されていること
- ③ ad-hoc 多相の機能を持つこと
- ④ 最初のオブジェクト指向言語であるので、インスタンス、オブジェクト、コンストラクタ、インスタンス変数、メソッド、継承といった用語を紹介

この中で必要要件の、継承、多相性を持つことを確認。情報隠蔽は時間があれば。

C++イントロ

Cとの類似性

```
1  #include <stdio.h>
2
3  int show(int x) { return x + 1; }
4  int show(char c) { return 100; }
5
6  int main(void)
7  {
8      printf("1: %d\n", show(3));
9      printf("2: %d\n", show('A'));
10     return 0;
11 }
```

C++イントロ

ad-hoc 多相の確認

```
1  #include <iostream>
2  using namespace std;
3
4  int show(int x) { return x + 1; }
5  int show(char c) { return 100; }
6
7  int main(void)
8  {
9      cout << "1: ";
10     cout << show(3);
11     cout << endl;
12     cout << "2: " << show('A') << endl;
13     return 0;
14 }
```

この中で多相性を持つものは

クラス, コンストラクタ, インスタンス, オブジェクト

```
1  #include <iostream>
2  using namespace std;
3
4  class C1 {                                // struct型の拡張であるクラスを定義
5  protected:                               // ここから情報隠蔽する
6      int x;                                // 隠蔽されたインスタンス変数
7  public:                                   // ここから情報隠蔽されない
8      int y;                                // インスタンス変数
9      int result (void) { return x + y; }    // C1用関数=メソッド
10     void reset (void) { x = 0, y = 0; }    // 別のメソッド (多相性を持つ)
11     C1() { x = 0, y = 0; }                 // 生成関数=コンストラクタ
12     C1(int p1, int p2) { x = p1, y = p2; } // さらにコンストラクタ
13 };
14
15 int main(void) {
16     C1 a(1,1);                            // C1型 変数a その内部初期値1, 2
17     C1 b(3,4);                            // コンストラクタによりインスタンスb生成
18     cout << a.result() << endl;          // 主語 インスタンスa
19     cout << b.result() << endl;          // 動詞 メソッドresult
```

a, b は変数だが, OOPではこれらをインスタンスとも言うし, オブジェクトとも言う

プログラミング言語論専門用語

11

instance or object, インスタンスまたはオブジェクト

クラスから作られた値をオブジェクト指向言語ではインスタンスまたはオブジェクトという。

instance variables, インスタンス変数

インスタンスの持つ変数。その利用範囲（**スコープ**）はクラスで定義された関数全てになる。ただし、C++を含むいくつかの言語ではキーワードでスコープを制御できる。これにより細かな情報隠蔽が可能になる。

- public: どこからでも使えるグローバルスコープ
- private: 他のクラスからは使えない、最も強い情報隠蔽
- protected: このクラスに包含されるクラスにもスコープを拡張

型の階層構造の確認

クラス C2 を追加

```
1  class C1 {
2  protected:
3      int x;
4  public:
5      int y;
6      int result (void) { return x + y; } // C1用関数1
7      C1() { x = 0, y = 0; }
8      C1(int p1, int p2) { x = p1, y = p2; }
9  };
10
11 class C2 : public C1 { // 継承による型の包含関係の指定
12 public:
13     int result (void) { z = 1; return x * y; } // ad-hoc多相
14     C2(int p1, int p2) : C1(p1, p2) { x = p1, y = p2; }
15 };
16
17 int main(void) {
18     C1 a(1,1); // 1,1 を引数にしてC1のインスタンスaを生成
19     C2 b(3,4); // 3,4 を引数にしてC2のインスタンスbを生成
20     cout << a.result() << endl;
21     cout << b.result() << endl;
```

「ad-hoc 多相を説明せよ」に対する誤った説明

- 「1つの関数に複数の異なる型が定義できる」
- 「ある関数に対して複数の型宣言（定義）が存在する」
- 「関数が型を複数持っていることを ad-hoc 多相という」
- 「型が1つに定まらず複数の型がある」
- 「ある関数が複数の型の引数を持つ」

「ad-hoc 多相を説明せよ」に対する誤った説明

- 「1つの関数に複数の異なる型が定義できる」

主語の取り違い

部分点 2 割

- 「ある関数に対して複数の型宣言（定義）が存在する」

型は常に1つ。型宣言と関数定義は違う。型定義という言葉は未使用。

1 割

- 「関数が型を複数持っていることを ad-hoc 多相という」

全ての識別子は型を1つしか持たない。変数を使っても1つ

1 割

- 「型が1つに定まらず複数の型がある」

主語不在

2 割

- 「ある関数が複数の型の引数を持つ」

引数 1 と引数 2 の型が違うということか

4 割

警告

型変数を使っても変数は1つの型しか持たないので複数の型を持つとは言えない。

固定された1つの型 から 動ける1つの型 に変わった

wikipedia での表現を確認

- ad-hoc polymorphism: *when a function denotes different and potentially heterogeneous **implementations** depending on a limited range of individually specified types and combinations* [▶ Wikipedia](#)
- ad hoc 多相 : 「～に対して◎◎を持つ」とは書いてあるが「複数の型を持つ」とは書いてない [▶ Wikipedia 日本語版](#)
- parametric polymorphism: *when code is **written without mention of any specific type** ...* [▶ Wikipedia](#)
- parametric 多相 : 「様々な型に～使用できる」とは書いてあるが「複数の型を持つ」とは書いてない [▶ Wikipedia 日本語版](#)

今年から部分点削減

蛇足：数え方に関する意識合わせ

太郎の手を変数 h で表す．グー，チョキ，パーと3種類あるので h は複数の値を取りうる．これは「 h は複数の値を持つ」なのか．

- 「 h は複数の値を持つ」と聞いた人は太郎はグーとチョキとパーを**同時**に出せると考える．それは説明が悪いから．
- 「 h は複数の値を持つ」と「 h は任意の値を取りうる」とは伝わる内容が違う．
- 型に戻すと変数 x が `Int` であり**同時に** `String` でもあるなら「変数 x は複数の型を持つ」と言える．ありえない．
- 「変数 x は任意の型を取りうる」は**型変数を使うことで一意に表現できるから「任意の型を取りうる型」は1つの型である**．型が複数あるとは型宣言文が2行以上あるようなもの考えること．しかし，**全ての識別子は1つの型しか持たない**．