プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 10 追加: オブジェクト指向 (Subtyping, Generics)

Generics

型をパラメータを取るクラス定義:

```
class Pair<A,B> {
  private A a;
  private B b;
  Pair (A x, B y) {
     a = x; b = y;
  }
  A First  () { return a; }
  B Second () { return b; }
}
```

パラメータ多相の一種: (ML と違って) 型をパラメータとして明示的に与える。

Generics の使い方

```
Integer i = new Integer(3);
String v = new String("pippo");
Pair<Integer,String> c = new Pair<Integer,String> (3,v);
String w = c.Second ();
```

メソッド

```
Pair<Object,Object> diagonal (Object x) {
  return new Pair<Object,Object> (x,x);
}
Pair<Object,Object> cp = diagonal(v);
Pair<String,String> cs = diagonal(v); // error!!!
```


Generic なメソッド

```
<T> Pair<T,T> diagonal (T x) {
  return new Pair<T,T> (x,x);
}
```

使い方:

```
Pair<String,String> cs = diagonal(v); // ok!
```

Generics と Subtyping

Subtyping (復習) $A <: B$

- 型 (あるいはクラス) A が B の部分型である。
- 型 B の式を書くべきところには、いつでも、型 A の式を書いてよい。(代入可能性)
- 例: ColoredPoint ¡: Point (子クラスは親クラスの subtype)

Generics: 型パラメータを明示した多相性 (教科書から引用)

```
class Stack<A> {
  private Elem<A> top = null;
  boolean isEmpty() { return top==null; }
  void push (A o) {
     Elem<A> ne = new Elem<A>();
     ... }
  A pop () { ...}
}
```


Subtyping と Generics

ここでの仮定

`Stack<Integer> <: Stack<Object>`

```
Stack<Integer> si = new Stack<Integer>();
Stack<Object>  so = si;
so.push (new String("pluto"));
Integer i = si.pop ();
```

上記プログラムに関する観察

- もし、`Stack<Integer> <: Stack<Object>`なら、すべて型が整合する。
- 実行すると、4 行目でおかしなことが起きる。
- → 静的型付けの型システムの良い性質が壊れる。

静的型付けをもつ型システムの良い性質

- Well typed programs don't go wrong.

Subtyping と Generics

前ページの結論

`Stack<Integer> <: Stack<Object>` としてはいけない。

Java Generics は subtyping 関係を保たない。

$$A <: B \text{ does NOT imply } Stack < A > <: Stack < B >$$

Subtyping と Override

Override とは見なされない例 ($B <: A$ のとき)

```
class F { C foo (A p) { ...} }
class G extends F { C foo (B p) { ...} }
```

※ コンパイルエラーではなく、Overload と見なされる

Override と見なされる例 ($D <: C$ のとき)

```
class F { C foo (A p) { ...} }
class E extends F { D foo (A p) { ...} }
```

発展課題. 上記の 1 つ目を Override とすると、なぜ問題か、Generics の例を参考に考えなさい。

Variance

Subtyping が「保存される」かどうかの性質:

- Java 配列は covariant: $A <: B$ ならば $A[] <: B[]$
- Java Generics は invariant: 「$A <: B$ ならば $G < A > <: G < B >$」は不成立
- 関数型 $X \to Y$ の結果型 Y は covariant: $A <: B$ ならば $X \to A <: X \to B$
- 関数型 $X \to Y$ の引数型 X は contravariant: $A <: B$ ならば $B \to Y <: A \to Y$

まとめ

Subtyping

- オブジェクト指向言語では非常に自然に出現する概念
- Subtyping Polymorphism (サブタイプ多相)
- (参考) Generics はパラメータ多相
- Subtyping を持つ型システムでは、variance をきちんと理解する必要がある。