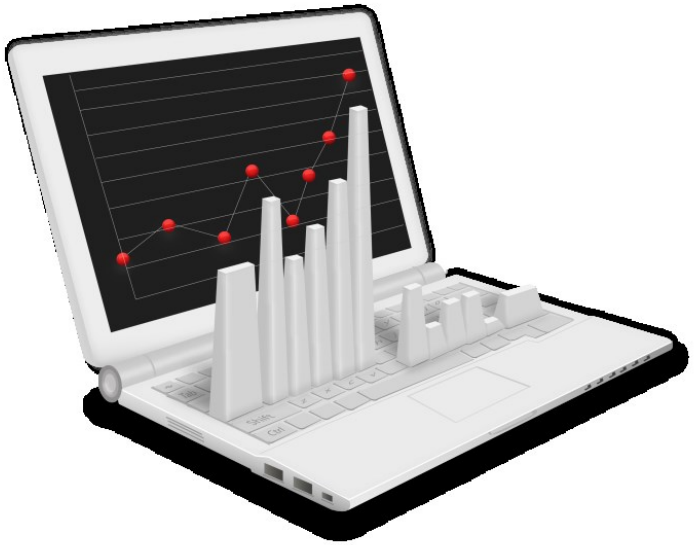
프로그래밍 언어론

Polymorphic Subprograms
(다형 부프로그램)

컴퓨터공학과

이만호



충남대학교
CHUNGNAM NATIONAL UNIVERSITY

Polymorphic Subprograms

학습목표

Polymorphic subprogram의 유형인 Overloaded subprogram과 Generic subprogram에 대해서 학습한다.

학습내용

- Overloaded subprogram(중복 부프로그램)
- Generic subprogram(포괄형 부프로그램)
- Parametric polymorphism(인자 다형성)

목 차

- 알고가기
- 학습하기
  - Overloaded subprogram(중복 부프로그램)
  - Generic subprogram(포괄형 부프로그램)
  - Parametric polymorphism(인자 다형성)
- 평가하기
- 정리하기

알고가기

아래에 주어진 2개의 코드를 하나의 subprogram으로 작성하려 할 때, parameter(매개변수)로 처리할 수 없는 것은?

```
A: int   x, y;                    B: float   x, n;
   if (x>y) x = x - y;                if (x>n) x = x - n;
   else     y = y - x;                else     n = n - x;
```

(a) A의 `int`, B의 `float`

(b) A의 `x`, B의 `x`

(c) A의 `y`, B의 `n`

(d) A의 `"if ... else ...;"`, B의 `"if ... else ...; "`

확인

Overloaded subprogram

Overloaded Subprogram(중복 부프로그램)

- 한 참조 환경에서 이름이 같은 여러 개의 subprogram
- 각 overloaded subprogram은 고유한 protocol을 갖는다.
  - Parameter의 개수(number), 순서(order), type
  - 함수의 경우 return 값의 type

Overloaded Subprogram을 호출하기

- Actual parameter의 개수 및 type들과 protocol이 일치하는 overloaded subprogram 중 하나가 호출된다.

지원 언어 : C++, Java, Ada, C#

- 미리 정의된 overloaded subprogram을 제공한다.
  - Class의 생성자(constructor) 등
- Programmer가 overloaded subprogram을 작성할 수 있다.

Overloaded Subprogram의 사용 예 - 1

3개의 Overloaded subprogram

```
int    max(int    x, int    y)          // max#1
{ return x>y ? x : y; }
double max(double x, double y)          // max#2
{ return x>y ? x : y; }
int    max(int    x, int    y, int z) // max#3
{ return x>y ? (x>z ? x : z) : (y > z ? y : z); }
```

호출 예

- `max(2, 3)` : #1
- `max(2.1, 3.2)` : #2
- `max(1, 3, 2)` : #3
- `max(2.1, 3)`
  - C++: 오류 (호출 가능: #1, #2)
    - 가능한 coercion: `int`→`double` (widening), `double`→`int` (narrowing)
  - Ada: 오류 (coercion을 허용하지 않음)
  - Java: #2
    - 가능한 coercion: `int`→`double` (widening)

Overloaded subprogram의 사용 예 - 2

2개의 Overloaded subprogram

```
int    max(int x, int y)   // max#1
{ return x>y ? x : y; }
double max(int x, int y)   // max#2
{ return x>y ? x : y; }
```

호출 예

```
int x;
...
x = max(2, 3);
```

- C++ : 오류 (호출 가능: #1, #2)
  - 함수의 return type을 고려하지 않음
- Ada : #1
- Java : 오류 (호출 가능: #1, #2)
  - 함수의 return type을 고려하지 않음

Generic Subprogram(포괄형 부프로그램)

Generic Subprogram(포괄형 부프로그램)

- **Polymorphic Subprogram(다형 부프로그램)이라고도 함**
  - Ppolymorphic은 "다양한 형태"를 뜻하는 그리스어가 어원임
- **여러 가지 type의 actual parameter를 전달 받아 실행할 수 있다.**
- **소프트웨어 재사용성을 향상시킨다.**

→ 예

배열(여러 종류의 type)을 정렬하는 subprogram

```
Sort(int    a[]) {...}
Sort(float  a[]) {...}   ➔  generic Sort ...
Sort(double a[]) {...}
Sort(char   a[]) {...}
...
```

→ 각 배열에 대해서 별도의 sort 함수를 작성하지 않고, 통합하여 단지 한 개의 sort 함수만을 작성 가능

Polymorphic Subprogram(다형 부프로그램)

- **Ad hoc polymorphism(특이 다형성)**
- **Parametric polymorphism(인자 다형성)**

Ad hoc Polymorphism(특이 다형성)

Ad hoc Polymorphism(특이 다형성)

- "Ad hoc"는 "특정 목적으로 수행되는 무엇"을 뜻하는 그리스어가 어원임
- Overloaded subprogram이 ad hoc polymorphism(특이 다형성) 특성이 있다.
  - Protocol은 다르나, 이름이 같은 여러 subprogram이 있다.
  - Overloaded된 각 subprogram은 서로 독립적이므로, 특정 목적을 가지며, 유사하게 동작할 필요가 없다.

```
int    max(int    x, int    y) // max#1
{ return x>y ? x : y; }
double max(double x, double y) // max#2
{ return x>y ? x : y; }

... max(...) ...
```

※ max#1과 max#2는 서로 독립적임

Parametric Polymorphism(인자 다형성)

Parametric Polymorphic Subprogram(인자 다형 부프로그램)

- 개념 Type 이름을 parameter(인자, 매개타입)화한 것이다.

```
max(int   a, int   b) { int   tmp; ... }
max(float a, float b) { float tmp; ... }
max(char  a, char  b) { char  tmp; ... }
```

- Formal parameter의 type을 기술하는 자리에 generic type 이름을 쓴다.

```
generic
    type g_type;     // generic type 인자
    max(g_type a, g_type b) { g_type tmp; ... }
```

- 특정 type의 parameter를 다루는 subprogram이 필요할 때,
  generic type 이름을 원하는 type 이름으로 지정할 수 있다.

```
Int_max : max(g_type=>int)
```

- 새로운 subprogram이 생성됨

```
Int_max(int a, int b) { int tmp; ... }
```

→ 이와 같이 생성된 subprogram들은 모두 동일하게 행동한다.

- Generic subprogram(포괄형 부프로그램)이라고도 불린다.

Ada의 Parametric Polymorphism - 1

Generic Parameter를 이용한 Parametric Polymorphism

```
Generic   // Template for a Sort procedure
  type NDX is (<>);      // NDX, ELT, VEC: generic types
  type ELT is private;
  type VEC is array (Integer range <>) of ELT;
  procedure G_Sort(List: in out VEC);
  procedure G_Sort(List: in out VEC) is
    Temp: ELT;       /* local variable */
  begin
    ...
  end G_Sort;
```

Generic G_Sort 이용하기

- Generic type에 특정 type 지정

```
procedure Int_Sort is
  new G_Sort(NDX=>Integer; ELT=>Integer; VEC=>Int_Array);
```

  - Compiler는 G_Sort와 실행 방법이 동일한 subprogram Int_Sort를 생성한다.

- 특정 type이 지정된 subprogram 호출

```
Int_Sort(a_list);
   - a_list의 type이 Int_Array이어야 함
```

Ada의 Parametric Polymorphism - 2

Generic Formal Subprogram(포괄형 형식 부프로그램)을 이용한 Para… Poly…

```
Generic    // Template
  with function Fun(X:Float) return Float;
                // Fun: generic formal subprogram
  procedure Integrate(Low: in Float;  Up: in Float;
                      Result: out Float) is
    Funval: Float;    // local variable
  begin
     ...   Funval := Fun(Low);   ...
  end;
```

Generic Integrate 이용하기

- Generic formal subprogram에 특정 함수(function) 지정

```
Integ_sin is new Integrate(Fun => sin);
```

  - Compiler는 `Integrate`와 실행 방법이 동일한 subprogram `Integ_sin`을 생성한다.

- 특정 함수(function)가 지정된 subprogram 호출

```
Integ_sin(a, b, r);
   - a, b, r의 type이 Float이어야 함
```

C++의 Parametric Polymorphism

Generic Function(포괄형 함수)을 Template Function이라고 한다.

Generic Parameter를 이용한 Parametric Polymorphism

```
template <class Gtype>      // Gtype: generic type
Gtype max(Gtype a, Gtype b) {
  return  a > b ? a : b;
}
```

- Generic type 인자 `Gtype`은 actual parameter의 type으로 bind된다.

Generic Subprogram 이용하기

- Generic type parameter `Gtype`은 actual parameter의 type으로 bind된다.

```
int  x, y;
...
...  max(x, y); ...
```

- Compiler는 parameter의 type이 `int`인 함수를 생성한다.

```
int max(int a, int b) {
  return  a > b ? a : b;
}
```

C의 Macro

Generic Function은 macro를 사용하여 표현할 수 있다.

Generic Function)

```
template <class Gtype>      // Gtype: generic type
Gtype max(Gtype a, Gtype b) {
  return  a > b ? a : b;
}
```

Macro

```
#define max(a, b) ((a) > (b)) ? (a) : (b)
```

➔ **크기 비교가 가능한 모든 type에 대해서 가능하다.**

Macro의 문제점

- Type 검사를 하지 않는다.
- 각 parameter는 body에 나타날 때마다 항상 값이 계산된다.
- Actual parameter가 side-effect를 가지면 결과 예측이 어렵다.

```
max(k++,n) → ((k++) > (n)) ? (k++) : (n)
```

- **k>n이면, k가 두 번 증가.**
- **k<=n이면, k가 한 번 증가**

Java의 Generic Method

Interface를 이용하는 경우

```
public static void foo(Comparable[] fa)
{ ... }
```

➔ fa는 배열로서, 요소가 Comparable interface를 implement한 class의 객체이어야 한다.

Generic type을 이용하는 경우

```
public static <T> T goo(T[] ga)  // T: generic parameter
{ ... }
```

- 호출하기

```
String[] names = {"Kim", "Lee", "Park"};
String result = <String> goo(names);
```

Interface를 implement한 generic type을 이용하는 경우

```
public static <T extends Comparable> T hoo(T[] ha)
{ ... }
```

➔ ha는 배열로서, 요소가 Comparable interface를 implement한 class의 객체이어야 한다.

Java의 Generic Method의 특징

Generic parameter는 class이어야 한다.

- Primitive type은 허용되지 않는다.
- Ada, C++: primitive type도 허용됨

**Generic method는 다양한 type으로 여러 번 호출될 수 있지만,
단 하나의 code만이 생성된다.**

- Ada, C++
  : Generic subprogram의 각 호출에 대해서 별개의 subprogram이 생성된다.

Generic method에 전달될 수 있는 class의 범위를 제한 할 수 있다.

- Generic parameter를 이용해서

```
public static <T extends Comparable> T hoo(T[] ha)
{ ... }
```

C#의 Parametric Polymorphism

Generic parameter를 이용한 Parametric polymorphism

```
class MyClass {
  ...
  public static T foo<T>(T fp) { ... }
}
```

- generic type parameter T는 Actual parameter의 type으로 bind된다

Generic subprogram 이용하기

- Actual parameter type을 지정하여 generic parameter를 대치한다.

```
int  myInt = MyClass.foo<int>(37);
String myStr = MyClass.foo<string>('Hello');
```

- Actual parameter type을 생략할 수 있다.
  - Actual parameter의 type에 따라 generic parameter를 대치한다.

```
int  myInt = MyClass.foo(37);
String myStr = MyClass.foo('Hello');
```

평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?

총 2문제가 있습니다.

START

평가하기 1

1. 다음의 overloaded subprogram에 대한 설명 중 올바르지 않은 것은?

a. 같은 참조 환경에서 이름이 같은 subprogram을 말한다.

b. Overloaded subprogram 각각은 고유한 protocol을 갖는다.

c. Overloaded subprogram은 actual parameter의 개수 및 type과 return 값의 type에 의해서 호출할 subprogram이 결정된다.

d. Overloaded subprogram은 반드시 유사하게 행동해야 한다.

확인

평가하기 2

2. 다음의 generic subprogram에 대한 설명 중 올바르지 않은 것은?

a. Generic subprogram은 소프트웨어의 생산성을 향상시킨다.

b. Generic subprogram은 호출될 때마다 다르게 행동할 수 있다.

c. Generic subprogram의 다른 사례화시에 다른 type parameter가 제공될 수 있다.

d. Java에서는 generic parameter로 `int`와 같은 기본 타입을 사용할 수 없다.

확인

정리하기

Overloaded Subprogram(중복 부프로그램)

- 한 참조 환경에서 이름이 같은 여러 개의 subprogram들

Generic Subprogram(포괄형 부프로그램)

- 여러 가지 type의 actual parameter를 전달 받아 실행할 수 있다.
- 여러 가지 type의 data를 하나의 실행 code로 실행할 수 있다.

Parametric Polymorphism(인자 다형성)

- Generic subprogram을 작성할 수 있다.

"강의를 마치겠습니다. 수고하셨습니다."