Java 언어에 structure type의 도입

이 호 석†

요 약

Java 프로그래밍 언어는 general-purpose concurrent object-oriented 언어로 알려져 있다. Java 언어는 개념과 구문 모두가 매우 간결하고 통일되어 있으며 인터넷 환경에서 최대한 활용되도록 하기 위하여 가상기계 개념을 도입하여 목적코드를 생성한다. 프로그래밍 언어에서 가장 중요한 부분이 data type 부분이다. Java 언어는 primitive type과 reference type을 지원한다. Primitive type에는 boolean type과 integral type이 있다. Integral type에는 character, byte, short integer, integer, long integer, single-precision 과 double-precision floating point number가 있다. Reference type에는 class type, interface type, array type이 있다. 그러나 Java 언어는 general-purpose 프로그래밍 언어가 일반적으로 지원하는 structure type을 지원하지 않는다. 대신에 class type이 structure type을 포함하여 지원하는 구조로 되어 있다. 그러나 class type과 structure type은 서로 상이한 data type으로 판단된다. 따라서 Java 언어가 general-purpose의 성격을 가지기 위해서는 structure type을 명시적으로 지원하는 것이 바람직하다고 생각된다. 이 논문은 structure type을 Java 언어에 포함시킬 것을 제안한다.

The structure type introduced in Java

Ho Suk Lee†

ABSTRACT

Java is considered a general-purpose concurrent object-oriented programming language. Its syntax is similar to C++, but it omits many of the features of C++. Java is a strongly typed language. The types of the Java language are divided into two categories : primitive types and reference types. Primitive types are boolean type and integral types. Integral types are byte, short integer, integer, long integer, single-precision and double-precision floating point numbers. Reference types are class types, interface types, array types. Java does not explicitly support structure type which general-purpose programming languages universally support. Instead the class type in itself conceptually incorporates the structure type. Therefore the programmer has to use class type whenever the structure type is considered to be more appropriate for the simple structuring of related data types. But the class type and the structure type are considered to be different data types. So this paper argues that Java must explicitly support the structure type so as to be necessarily considered as a general-purpose programming language.

1. 서 론

1.1 연구 배경

우리는 Java 프로그래밍 언어[1,2,3,4,5]를 읽으면서 대단한 흥미를 느낄 수 있다. 우리는 예전에 Fortran, Algol, Pascal과 같은 프로그래밍 언어에 대하여 읽을 때 Fortran으로부터 Algol이 개발되었고, 또한 Algol로부터 Pascal이 발전되어 나왔다는 설명을 읽을 수 있었다. 그러나 우리는 그러한 발전과정을 직접 경험할 수는 없었다. 하지만 우리는 이 시대에 이와 비슷한 프로그래밍 언어의 발전 현상을 직접 경험할 수

† 정 회 원 : 호서대학교 컴퓨터학부 교수
논문접수 : 1997년 12월 22일, 심사완료 : 1998년 5월 26일

있다. 이 글은 바로 Java 프로그래밍 언어의 점차적인 사용으로 인하여 C 언어에서[6] C++ 언어로[7,8,9,10,11] 그리고 C++ 언어에서 Java 언어[1,2,3,5]에 이르는 프로그래밍 언어의 발전과정을 직접 경험할 수 있기 때문이다.

여러분들이 이미 C 언어에서 C++ 언어로의 발전 과정에 대하여서는 많이 알고 있을 것으로 생각한다. C++ 언어는 C 언어를 포함하고 있다. C 언어에 객체지향 개념[7,8,12] 이 포함되고 이 것을 class라는 data type을 통하여 구현한 것이 C++ 언어라고 할 수 있다. 결과적으로 C++ 언어는 C 언어에 비하여 매우 커졌으며 객체지향 개념의 중요한 요소인 다형성(polymorphism), 연산자 중복정의(operator overloading), 다중상속(multiple inheritance)과 같은 특징들을 포함하고 있다 [7,9,10].

Java 프로그래밍 언어는 C 언어와 C++ 언어에 그 기반을 두고 있다. 그러나 C++ 언어가 가지고 있는 복잡하고 에러를 유발시키기 쉬운 특징들을 Java 언어는 포함하고 있지 않다. Java 언어의 구문은 C 언어나 C++ 언어에 비하여 비교적 간단하며 keyword의 사용도 자연스러운 면이 있다. Java 언어는 structure type, pointer type, template, operator overloading, multiple class inheritance등의 특징들을 가지고 있지 않다[1,2,3,5]. 대신에 reference type, graphics, multithread, multimedia handling, network 기능등을 가지고 있다. Graphics, multithread, multimedia handling, network 기능등은 언어 외적인 요소라기 보다는 거의 Java 언어의 언어 내적인 요소같은 느낌이 들 정도이다. 결국 언어 자체는 간결하여 졌으나 더욱 많은 기능을 포함하고 있다.

언어의 성공요소중에 언어를 설계하고 구현한 기관의 강력한 지원도 큰 역할을 한다[17,18]. 현재 Java 언어는 Java 언어의 개발자인 Sun Microsystems의 강력한 지원을 받고 있다. 뿐만 아니라 IBM & Lotus, HP, Netscape, Oracle, Apple, Digital [24,25] 등의 기관으로부터도 역시 강력한 지원을 받고 있다. 따라서 Java 언어의 미래는 상당히 낙관적이라고 할 수 있으며 더욱이 새로운 internet 응용을 Java 언어를 이용하여 구현하려고 하기 때문에 더욱 많은 응용 프로그램이 탄생할 것이다.

본 논문은 우선 Java 언어의 특징들을 살펴본다. 프로그래밍 언어에서 가장 중요한 부분이 data type 부분이다. Java 언어는 primitive type과 reference type을 지원한다. Primitive type에는 boolean type과 numeric type이 있다. Numeric type에는 character, byte, short integer, integer, long integer, single-precision과 double-precision floating point number가 속한다. Reference type에는 class type, interface type, array type이 속한다. 그러나 Java 언어는 일반적으로 general-purpose 프로그래밍 언어가 지원하는 structure type을 지원하지 않는다. 대신에 class type이 structure type을 포함하여 지원하는 구조로 되어 있다. 그러나 class type과 structure type은 서로 상이한 data type으로 판단된다. 따라서 Java 언어가 진정한 general-purpose 프로그래밍 언어가 되기 위하여서는 반드시 structure type을 명시적으로 지원하는 것이 바람직하다고 생각된다. 이 논문은 structure type을 Java 언어에 포함시키는 것을 제안한다.

2. Java 언어의 특징

2.1 Java 언어의 간결성 특징

Java 언어는 전처리기를 가지고 있지 않다. C++ 언어는 #define, #inculde, #typedef 등의 컴파일러 명령을 가지고 있으며 이를 처리하기 위하여 언어외적인 요소인 전처리기를 가지고 있다. 그러나 Java는 #define 대신 상수 데이터 멤버를 가지고 있으며 #include 대신 import 가 사용되며 #typedef 대신 class가 정의되어 사용된다. 그 결과 Java source code는 C++ source code에 비하여 훨씬 일관성이 있으며 프로그램의 가독성(readability)이 높은 편이다. 더욱이 Java 언어에는 C++ 언어에 존재하는 header file이 존재하지 않는다. 모든 것은 API(Application Programming Interface) class에 정의되어 있으며 class로 부터의 상속(inheritance)을 통하여 프로그램을 구성한다.

Java 언어는 다중상속(multiple inheritance)을

지원하지 않는다. 다중상속은 강력한 경우도 있지만 사용하기가 어려우며 여러 가지 문제를 유발시킬 수 있다. 대신에 Java 언어는 interface type을 사용하여 다중상속의 효과를 얻고 있다. 참고문헌 [20]에는 다중상속이 유발시키는 문제들이 잘 설명되어 있다.

Java 언어는 연산자 중복정의(operator overloading)를 지원하지 않는다. 이 특징은 C++ 언어에서는 뛰어난 특징으로 여겨지고 있다. 그러나 Java 언어는 이를 지원하지 않는다. 이로 인하여 Java 언어는 C++ 언어에 비하여 간결하여졌다.

Java 언어는 function을 지원하지 않는다. Java 언어는 오직 method만을 지원한다. C++ 언어에는 function이 있으며 더욱이 Java의 method와 동일한 member function도 class에 존재한다. 이로 인하여 C++ 언어는 개념적으로 복잡하게 되었다. Java에는 function은 존재하지 않고 오직 method만이 존재한다. 또한 Java 언어는 default argument도 지원하지 않는다.

Java 언어는 template class와 template function을 지원하지 않는다. 이 기능들은 사용하기가 어려우며 이 기능들을 이용하여 작성한 프로그램은 읽기도 매우 어렵다.

2.2 Java 언어의 기능적 특징

Java 언어는 기본적으로 method들의 병렬 수행을 고려한다. Java 언어는 multithread를 지원하며 multithread에 대하여 priority, scheduling, synchronization을 제공함으로서 병렬 수행을 지원한다.

Java 언어는 다중상속을 지원하지 않는다. 그러나 Java 언어는 interface type을 통하여 다중상속과 비슷한 기능을 지원하고 있다. Interface type이란 상수(constant)와 abstract method로 구성된 type이다. Abstract method란 구현(implementation)이 없는 method를 말한다. 다른 class들은 이미 선언된 interface를 상속한 다음 method를 구현하여 사용할 수 있다. Interface type 개념은 object에 대하여 정의와 구현을 분리하여 사용하는 생각이라고 할 수 있다.

Java 언어는 C++ 언어에 존재하는 메모리 해제(free) 명령이 존재하지 않는다. 대신에 garbage collection을 프로그래머가 명시적으로 사용하게 함으로서 더욱 메모리 관리의 효율성을 증가시켰다.

2.3 Java 언어의 구문적 특징

Java 언어로 작성한 프로그램은 class로만 구성된다. 즉 Java 언어는 거의 완벽한 객체지향 프로그래밍 언어이다.

Java 언어는 C++ 언어의 구문을 상당부분 이어받았다. 그러나 다른 점도 많다. 우선 첫째로 C++ 언어에 존재하는 structure 구문이 사라졌다. C++ 언어에서는 structure와 friend function 모두 존재한다. 그러나 Java 언어에서는 structure와 friend function 모두가 사라졌으며 대신에 modifier를 붙이지 않는 - modifier는 Java 언어의 private, protected, public access specifier를 의미한다 - class에 대하여 friendly access를 허용함으로서 structure와 class를 함께 지원하고 있는 셈이다. 이 것은 C++ 언어와 Java 언어를 비교하였을 때 구문이 간결하게 정의되었다는 인상을 줄 수도 이다. 그러나 한편으로는 자료구조의 구축 측면에서 문제점을 내포하고 있다. 그 문제점이란 바로 class와 structure를 같은 의미로 혼동하여 사용하게 된다는 점이다[1,2,5]. 본 논문의 3장에서 이 문제에 대하여 심도있게 다루기로 한다. 다음은 friendly access를 허용하는 class의 선언이다.

```
class FriendlyData {
      int x ;
      String s ;

      public FriendlyData()
      {
        x = 0 ;
        s = new String ( "Hello" ) ;
      }
      public String toString()
      {
        return "x: " + x + "     s : " + s ;
      }
}
```

(예 1) Java class의 friendly access
(example 1) Friendly access of Java class

두 번째로는 명시적인 pointer 개념이 사라졌다. 이것은 매우 주목할 만한 일이다. Java 언어는 class에 대하여 reference에 의한 참조를 함으로서 pointer 변수를 사용할 필요가 없도록 만들었다. 이러한 생각은 graphics data 그리고 multimedia data를 다루기 위하여서는 reference에 의한 참조가 기본적이며 이것을 Java 언어에 도입하는 과정에서 pointer에 의한 참조와 reference에 의한 참조로 나누어져 있는 것을 reference에 의한 참조로 통일하기 위하여 고안한 생각이라고 판단된다. Reference에 의한 참조 자체가 새로운 개념은 아니다. 기존의 언어에도, 예를 들면 C 언어, function에 parameter를 전달하는 과정에서 call-by-reference로 parameter를 전달할 수 있다. 그러나 Java 언어에서는 class, **this** pointer, pointer variable, graphics data, multimedia data 등을 고려하는 과정에서 class 자체에 대하여 reference에 의한 참조를 하게 함으로서 reference에 의한 참조 하나의 개념으로 class, **this** pointer, pointer variable, graphics data, multimedia data 등을 취급할 수 있도록 하였다. Class의 보편성과 reference에 의한 참조의 보편성으로 인하여 Java 언어는 C++ 언어에 비하여 더욱 간결함과 개념의 일관성을 가지게 되었다고 볼 수 있다.

다음은 Java 언어로 작성한 self-referential class Node에 대한 선언의 예이다.

```
// self-referential Node declaration //
class Node {
    int data ;
    Node next ;
    Node ( int v ) { data = v ; next =
    null ; }
    Node ( int v, Node nextNode ) { data =
    v ; next = nextNode ; }
    int getData( ) { return data ; }
    Node getNext( ) { return next ; }
}
```

(예 2) Java 언어의 재귀 참조 class Node 선언
(example 2) Self-referential class Node declaration of Java

이 선언을 살펴보면, class Node는 modifier가 없는 class로 선언되어 있다. 이로 인하여 friendly access가 가능한 class임을 알 수 있다. 즉, class를 이용하여 C++ 언어의 structure와 그리고 관련된 function을 구현한 것이다. 그리고 Node에 대한 선언에서 class에 대하여서는 reference에 의한 참조를 하기 때문에 Node next라고 선언한 것을 알 수 있다.

다음은 C++ 언어를 이용하여 위에서 설명한 self-referential node 구조를 선언한 예이다. Pointer variable에 대한 차이를 알 수 있을 것이다.

```
// C++ 언어를 사용한 self-referential Node 자
   료구조 선언 //
class Node {
   private :
      int data ;
      Node *next ;
   public :
      Node ( int d ) { data = d ; next =
      NULL ; }
      Node ( int d, Node *n ) { data = d ;
      next = n ; }
      int getData ( ) { return data ; }
      Node *getNext ( ) { return next } ;
} ;
```

(예 3) C++ 언어의 재귀 참조 class Node 선언
(example 3) Self-referential class Node declaration of C++

결론적으로 C++ 언어의 방대한 구문 중 꼭 필요한 부분만을 남겨놓고 사용하기 어렵거나 에러를 유발하기 쉬운 부분은 모두 없애 버린 셈이 된다. 또 다른 예로 C나 C++ 언어에는 엄연히 goto 문이 존재하며 사용할 수 있다. 그러나 Java 언어는 goto 문을 지원하지 않는다. Java에 goto라는 keyword는 존재하지만 사용되지는 않는다.

셋째로, Java 언어에서는 character string이 C++ 언어와는 다른 모습으로 처리된다. Java 언어는 character string을 하나의 class로 파악하여 나타내고 처리한다. 이 덕택에 C나 C++ 언어에서 여러

library routine을 사용하여 character string을 처리할 때 느낄 수 있었던 다소의 혼란을 해소하였다고 생각한다. 이 부분은 Java 언어가 거의 완전한 객체지향 언어임을 나타내는 하나의 예라고 할 수 있다.

2.4 Java Runtime Storage

2.4.1 Java Virtual Machine

Java 언어는 보완(protection)과 이식성(portability)을 최대한 보장하기 위하여 virtual machine 개념을 사용하였다[3]. Java 언어로 작성된 프로그램은 특정한 file format인 object class file format에 맞게 bytecode로 번역된다. Java virtual machine은 class file을 입력으로 읽어 프로그램을 수행한다.[3]

2.4.2 Java Runtime Data Area

Java 언어의 runtime data area는 4 부분으로 구성된다[3]. 이 4 부분은 stack area, heap area, method area, constant pool 이다.

Java 프로그램의 runtime data area는 stack (Java Stack)을 중심으로 구성된다. 이 Java stack은 일반 절차적(procedural) 프로그래밍 언어를 구현할 때 사용하는 runtime stack (혹은 central stack) [17,18]과 동일한 것으로 보인다. 각 thread는 Java stack을 하나씩 할당받는다. 각 Java stack은 frame을 저장한다. 이 frame은 일반적인 activation record[17,18]와 동일한 성격을 가진 것으로 보인다. 이 frame은 local variable, partial result, method invocation 그리고 return value 등을 저장한다. 그리고 Java 언어의 경우는 이 frame이 heap으로 구성되어 할당된다.

Heap area에는 class instance와 array등이 할당되며, heap area는 concurrent하게 수행되는 모든 thread에 의하여 공유된다. Heap area에 할당된 class instance와 array는 더 이상의 참조가 이루어지지 않으면 garbage collector에 의하여 자동적으로 회수된다. 이 heap area는 concurrent access를 위하여 monitor[21,22] 속에 위치하게 된다.

Method area는 heap area의 한 부분으로서 역시 concurrent하게 수행되는 thread에 의하여 공유된다. 각 class가 가지고 있는 constant, field, method data, method code를 저장한다.

Constant pool은 일반 컴파일러의 symbol table과 비슷한 것이다. 각 class마다 그리고 각 interface마다 할당되며 프로그램에서 정의한 constant들과 numeric literal들이 저장된다.

2.4.3 Frame

Java frame은 일반 activation record [17,18]와 비슷하다. Frame은 Java stack에 할당되며 각 frame은 method에서 정의된 local variable, operand stack, partial result를 위한 temporary area들을 저장한다. Operand stack은 method에 argument를 전달하고 method 수행 결과 return value를 저장한다.

2.4.4 Object Representation

Class instance에 대한 참조는 object handle에 대한 pointer를 통하여 이루어진다. Object handle은 pointer의 짝으로서 그 내용은 다음과 같다.

- Object handle
  - pointer 1
    - Object의 method들에 대한 pointer
    - Object의 type을 나타내는 pointer
  - pointer 2
    - Object를 저장하기 위하여 할당된 heap area에 대한 pointer

3. 프로그래밍 언어에서 structure type의 분석

이 절에서는 Pascal, C, Common LISP, C++ 언어에서 어떻게 structure type을 정의하고 있는지 살펴보고, 마지막으로 structure type에 대한 형식적 정의(formal definition)에 대하여 살펴봄으로서 일반 프로그래밍 언어에서 structure type이 반드시 필요하다는 점을 설명한다.

3.1 Pascal 언어의 record data type

Pascal 언어의 참고문헌 [13]에서 record 자료구조에 대하여 다음과 같이 설명하고 있다. PASCAL

includes a structured type called a record in which the elements can have different types. A record has a fixed number of components, and each component has a name, called the field identifier. We declare a record structure by using a type declaration of this form :

```
TYPE record-identifier = RECORD
        field-identifier-1 : data type ;
        field-identifier-2 : data type ;
                .
                .
        field-identifier-n : data type ;
    END ;
```

In this declaration, "record-identifier" represents the name we are giving to the entire structure. Each "field-identifier" and its associated data type is listed between the reserved identifiers RECORD and END.

Example 1. Inventory information

```
TYPE INVENTORY = RECORD
    PARTNUMBER : INTEGER ;
    PARTNAME   : PACKED ARRAY
                 [1..20] OF CHAR ;
    PRICE      : REAL ;
    QUANTITY   : INTEGER ;
  END ;
```

이러한 record 자료구조에 대하여 또 다음과 같이 그 목적을 설명하고 있다. 다음은 참고문헌 [13]에서 인용한 것이다. "We have seen that an array can be a useful data structure because it allows us to form a structure in which all of the items have the same type. There are times, however, when we want to deal with a structure that contains elements of different types. For example, we might want to maintain inventory information in the following form.

```
Part number : 43754
  Part name : brake shoe
      Price : 15.75
   Quantity : 12
```

We want to keep track of four different pieces of information, and there are three different data types involded : integer, real, and char (actually an array of characters). Multidimensional array will not be helpful in this case, because the individual objects have different types.

이상의 설명에서 Pascal의 record type은 상이한 data type의 element들로 구성되는 구조를 정의하고 다룰 필요가 있을 때 사용하는 것을 알 수있다.

3.2 C 언어의 structure data type

C 언어의 참고문헌 [6]에서 structure 자료구조에 대하여 다음과 같이 설명하고 있다. "**A structure is a collection of one or more variables, possibly of different types grouped together under a single name for convenient handling.**

그리고 structure의 목적에 대하여서는 다음과 같이 설명하고 있다. "**Structures help to organize complicated data, particularly in large programs, because in many situations they permit a group of related variables to be treated as a unit instead of as separate entities.**"

Structure의 예

```
struct date {
    int   day ;
    int   month ;
    int   year ;
} ;
```

"Keyword **struct** introduces a structure

declaration, which is a list of declaration enclosed in braces. An optional name called a structure tag may follow the word struct(as with date here). The tag names this kind of structure, and can be used subsequently as a shorthand for the detailed declaration."

C 언어에서는 더욱 분명히 structure type의 필요성에 대하여 설명하고 있다.

3.3 Common LISP 언어의 structure data type

Common LISP [15]에서는 structure를 다음과 같이 설명하고 있다. "All structures are defined through the **defstruct** construct

defstruct name-and-options [doc-string] {slot-descriptions}$^{+}$

This defines a record-structure data type. A general call to **defstruct** looks like the following example.

```
(defstruct (name option-1 option-2 ...)
           doc-string
           slot-description-1
           slot-description-2
             .
             .
)
```

The name must be a symbol; it becomes the name of a new data type consisting of all instances of the structure. The name is returned as the value of the **defstruct** form."

그리고 간단한 실제의 예는 다음과 같다.

```
(defstruct ship
           x-position    0
           y-position    0
           x-velocity    nil
           y-velocity    nil
           mass          17000.0
)
```

LISP 언어는 함수 프로그래밍 언어(functional programming language)로서 property list를 structure와 비슷한 목적으로 사용하였다. 그러나 절차적 특징(procedural feature)이 많이 포함된 Common LISP에서 structure type이 존재하며 그 이유는 새로운 data type을 정의하기 위한 것이라고 설명하고 있다.

3.4 C++ 언어의 structure data type

참고문헌 [7,10]에 C++ 언어의 structure가 소개되어 있다. C++ 언어는 C를 확장한 언어이며 특히 언어의 구현적인 관점에서 structure를 확대하여 class를 구현하였다. 따라서 C++에서는 structure와 class는 거의 동일하게 사용된다. C++ 언어에 있어서 structure와 class의 차이는 다음 두가지에 있다. 첫째는 structure는 기본적으로(by default) public accessibility를 가지는 반면 class는 기본적으로(by default) private accessibility를 가진다는 점이다. 다른 하나는 structure는 data로만 구성되는 반면 class는 data와 function (C++ 에서는 data member와 member function)으로 구성된다는 점이다.

즉, C++ 언어는 structure와 class간의 개념적인 차이를 거의 나타내지 않고 있다. 실제로 C++의 reference manual[7]에서는 다음과 같이 설명하고 있다. "A structure is a class declared with the class-key struct; its members and base classes are public by default.

그러나 그럼에도 불구하고 C++ 언어는 structure와 class 두가지 모두를 지원하고 있다. 이 것은 묵시적으로 structure와 class간의 차이 점을 인정하는 것이라고 할 수 있다.

이상에서, 일반 절차적 프로그래밍 언어에서 structure type을 정의하여 사용하는 이유는 서로 관련있는 data type을 하나의 단위로 구성하여 새로운 data type을 정의하기 위한 것이라는 것을 알 수 있다.

3.5 Structure data type에 대한 formal definition

참고문헌 [14]에서는 structure data type(혹은

record type)에 대하여 다음과 같이 형식적으로 (formal) 정의하고 있다.

. Definition

Records :

Let s_1, s_2, ..., s_m be distinct identifiers, and T_1, T_2, ..., T_m type identifiers. Then

type T = **records** s_1:T_1; s_2:T_2; s_3:T_3; . . . s_m:T_m **end**

define a new type T. The type T is a set of m-tuples (x_1, x_2, ... , x_m) where x_i is of type T_i, for i = 1, ... , m. **Employing the usual terminology of mathematics, we say that T is the cartesian product of the sets T_1, T_2, . . . , T_m.** Components of the record-type T are called fields and the s_i (i = 1, . . . , m) are called field identifiers. A variable x of type T is an m-tuple of variables, where the ith component variable is of type T_i and is denoted as x . s_i. Thus we can manipulate the whole structured value of x, or select a particular component of it, and manipulate the component.

이 정의는 structure가 cartesian product로 설명될 수 있음을 보여주고 있다. 즉, structure는 함께 고려하여야 하는 상이한 data type들의 cartesian product를 하나의 단위로 묶어서 나타내는 것이라고 결론지을 수 있다. 주목할 점은 이 정의는 cartesian product와 관련된 operation은 포함하고 있지 않다는 것이다.

그러므로 structure는 cartesian product이며 structure를 정의할 때, 처음부터 cartesian product와 그리고 관련된 operation까지 함께 미리 결정지워서 생각하는 것은, 구체적인 응용과 관련짓기 전에, 응용과 관련된 모든 경우의 수를 미리 고려하려고 시도하는 것이라고 할 수 있다. 물론 고려하여야 하는 경우의 수가 적은 cartesian product인 경우에는 가능할 수도 있겠지만, 일반적으로 고려하여야 하는 경우의 수가 많은 cartesian product인 경우 그리고 structure를 정의하는 시점에서 응용의 모든 경우가 파악되지 않은 경우에는 어려운 상황이 존재할 수 있다.

3.6 Object-oriented 언어의 두가지 의미 모델

참고문헌 [23]에서는 object-oriented 언어에 대하여 두가지 의미모델을 제시하고 있다. 하나는 closure model이고 다른 하나는 data structure model이다. Closure model이란 functional programming에서 operation 실행의 "side effect"를 encapsulate하기 위하여 오랜동안 사용된 모델이다. Data structure model이란 data와 operation을 함께 묶어서 생각하는 모델로서 우리가 알고 있는 abstract data type의 개념과 비슷하다. 여기서 주목할 점은 이 두 가지 모델 모두 data와 operation을 함께 묶어서 고려한다는 점이다. 그리고 이 두가지 모델은 서로 상대방 모델을 근사하는 것으로 증명되어 있다.

그러면 class의 개념은 무엇인가? Class는 기본적으로 class instance(object)들의 집단을 의미한다. 그리고 object-oriented 언어의 의미모델을 살펴볼 때, class type은 data structure model에 가까운 것을 알 수 있다. 그렇다면 class는 이러한 class instance를 생성시키고 다룰 수 있는 기능을 기본적으로 가지고 있어야 한다. 당연히 class는 data와 operation을 함께 정의할 수 있는 것이어야 한다.

그러나 structure는 위 3.5절의 정의에서 살펴 보았듯이 cartesian product이며, class와는 달리 오히려 data와 operation을 함께 정의할 수 없는 경우에 사용된다고 할 수 있다.

3.7 Java 언어에 structure type의 도입

Java 언어에서 structure type을 삭제한 이유는 다음과 같다고 생각한다. 즉, class type과 structure type의 비슷한 성격을 너무 확대하여 해석한 것으로 판단된다. 그래서 Java 언어에서 structure type을 없애고 대신 class type이 완전히 structure type을 대신할 수 있다고 판단한 것 으로 생각된다. 다시 말하

별 modifier가 붙지않은 class를 friendly class로 하여 structure와 같은 의미로 사용할 수 있다고 생각한 것 같다. 그러나 이러한 접근 방식은 궁극적으로 class type 속에 structure type을 부리하게 포함시키는 결과를 낳은 것으로 보인다.

Class는 object(class instance)를 구성하기 위한 것이다. Class는 data와 method를 함께 정의하고 encapsulate하여 object를 정의하기 위한 것이다. 그러나 class type속에 structure type을 포함시킨 것은 class를 두가지 목적 즉, object를 정의하기 위한 목적과, 단순히 관계있는 data type을 하나의 단위로 구성하여 새로운 data type을 정의하기 위한 두가지 목적으로 사용하는 것이다. 즉, class라는 하나의 syntactic construct가 두가지 의미로 사용되는 것이다. 이 것은 분명히 class와 structure 사이에 혼란을 초래한다. 이 혼란을 friendly class라는 개념으로 피하고는 있으나 바람직하다고는 생각되지 않는다. 왜냐하면 object를 생성하기 위한 class type과 서로 관련있는 data type들을 서로 묶어서 하나의 새로운 data type을 정의하기 위한 structure type은 분명히 서로 다른 data type이라고 생각되기 때문이다. Structure type에 대한 설명 및 예는 3.1절에서 3.6절까지 자세히 살펴보았다. 따라서 Java 언어에서 class의 개념과 structure type의 개념은 분리하여 structure type을 다시 정의하여 도입하는 것이 바람직 할 것이다. 즉, class type과 structure type은 서로 다른 data type이며 class가 structure를 포함하는 포함 관계에 있는 data type이 아니라 서로 보완 관계에 있는 data type이라고 할 수 있다.

즉, Java 언어에서는 modifier를 붙이지 않은 class에 대하여 public access를 하는 방법을 허용함으로서 class가 마치 일종의 structure와 같은 형태로 사용될 수 있도록 하였다. 그러나 Pascal, Common Lisp, C, C++ 그리고 3.5절에서 structure에 대한 형식적인 정의를 살펴보면 structure는 각기 다른 data type을 지닌 여러 data들을 하나의 단위로 구성하여 새로운 data type을 정의하는 역할을 하는 것을 알 수 있다. 이 경우 structure는 data로만 구성되는 것이 타당하다 하겠다.

여기서 class의 개념과 entity[19]의 개념에 대하여 생각하여 보자.

(1) Class는 data와 method를 encapsulate하여 구성한 것이다. 여기서 data와 관련된 method에 대하여 생각하여 보자. 우리가 어떻게 data와 그리고 관련된 method를 동시에 명확히 파악할 수 있는가? 이 질문에 대한 해답은 만약 class가 추상적이고 인위적으로 정의한 것이라면 method도 추상적이고 인위적으로 정의될 수 있기 때문에, data와 method를 함께 고려하여 encapsulate하여 궁극적으로 class를 구성할 수 있을 것이다.

여기서 class와 비슷한 개념으로 entity라는 개념을 생각할 수 있다. 이 두 개념 모두 어떤 대상을 가리키는 것이다. 그러면 그 차이는 무엇이라고 할수 있는가? Class를 생각할 때 method도 함께 고려한다면 entity를 생각할 때도 method를 함께 고려할 수 있는가?

(2) 그러면 entity란 무엇인가? Entity는 일반적으로 실제 세계에 존재하는 대상을 가리키며 그 개념은 entity/relationship model로 잘 알려져 있다. Entity와 relationship은 참고문헌 [19]에 다음과 같이 정의되어 있다 - entity as a thing which can be distinctly identified and relationship as an association among entities. 이 경우에 어떻게 entity와 relationship을 data와 method로 파악할 수 있는가? 우선 entity가 실제세계에 존재하는 대상이기 때문에 그 속성을 인위적으로 규정하는 것이 쉽지 않다. 더욱이 규정한다고 해도 완전하지 않을 수 있으며 시간이 흐름에 따라 그리고 상황이 바뀜에 따라 변할 수 있다. 따라서 이 경우 data와 method를 함께 encapsulate하여 사용하려고 하는 것은 미리 너무 앞서서 모든 경우의 수를 한정지으려고 하는 것이며 시간이 흐름에 따라 그리고 관련된 응용에 따라 entity의 속성이 다르게 파악되어 method가 변할 수 있기 때문에 바람직하다고 생각되지 않는다. 그러므로 이 경우 class처럼 완전한 encapsulation보다는 structure를 이용하여 entity를 나타내는 것이 더 바람직할 것으로 생각된다.

Pascal, Common Lisp, C, C++가 제공하는 structure는 바로 이런 경우에 적용하여 사용할 수 있

다. Java 언어에서는 이런 경우를 배제하였다. 그러나 위에서 설명한 이유와 같이 structure가 필요한 경우도 많이 존재하기 때문에 Java 언어에서 class가 structure를 흡수한 것은 재고되어야 할 것이다. Java 언어의 class는 항상 method까지 포함시켜야 하므로 응용에 따라 쉽게 적용하기가 어려운 상황이 발생할 수도 있다. 즉, class type과 structure type은 서로 다른 data type이며 class가 structure를 포함하는 포함 관계에 있는 data type이 아니라 서로 보완 관계에 있는 data type이라고 할 수 있다. 그러므로 Java 언어에서 class를 이용하여 structure를 나타내려고 한 것은 잘 못이라고 생각하며 structure type를 다시 도입해야 한다고 믿는다. 다시 도입하는 경우 structure type은 reference type에 포함시키는 것이 타당할 것이다.

이제 다음 예제 프로그램 (예 4) (예 5) (예 6)을 살펴보자.

```
//      C++ program segment

#include <iostream.h>

struct Time {
        int hour ;
        int minute ;
        int second ;
} ;
main()
{
        Time dinnerTime ;

        dinnerTime.hour = 18 ;
        dinnerTime.minute = 30 ;
        dinnerTime.second = 0 ;
                .
                .
                .
}
```

(예 4) C++ 프로그램 세그먼트
(example 4) C++ program segment

```
//      Java program segment

public class Time {
   private int hour ;
   private int minute ;
   private int second ;
   public Time( )
                    { setTime ( 0, 0, 0 ) ; }
   public Time( int h )
                    { setTime ( h, 0, 0 ) ; }
   public Time( int h, int m )
                    { setTime ( h, m, 0 ) ; }
   public Time( int h, int m, int s )
                    { setTime ( h, m, s ) ; }

   public void setTime ( int h, int m, int s)
   {
       hour = ( ( h >= 0 && h < 24 ) ? h
              : 0 ) ;
       minute = ( ( m >= 0 && m < 60 ) ?
                m : 0 ) ;
       second = ( ( s >= 0 && s < 60 ) ? s
                : 0 ) ;
   }
   public String getTime( )
   {
       return  ( ( ( hour <== 12 || hour
                  == 0 ) ? 12 : hour %
                  12 ) + ":" +
                ( minute < 10 ? "0" : "" )
                  + minute + ":" +
                ( second < 10 ? "0" : "" )
                  + second +
                ( hour < 12 ? " AM" : "
                  PM") ) ;
   }
            .
            .
}
```

(예 5) Java 프로그램 세그먼트
(example 5) Java program segment

```
// structure를 포함한 경우의 Java program
   segment

public class Time {

   struct time {
      int hour ;
      int minute ;
      int second ;
   }

   public Time( )   { setTime ( 0, 0, 0 ) ; }
   public Time( int h )
                  { setTime ( h, 0, 0 ) ; }
   public Time( int h, int m )
                 { setTime ( h, m, 0 ) ; }
   public Time( int h, int m, int s )
                  { setTime ( h, m, s ) ; }
   public void setTime ( int h, int m, int s)
   {
      time.hour = ( ( h )= 0 && h < 24 )
                  ? h : 0 ) :
      time.minute = ( ( m )= 0 && m <
                    60 ) ? m : 0 ) :
      time.second = ( ( s )= 0 && s < 60
                    ) ? s : 0 ) :
   }

   public time getTime( )
   {
      return time ;
   }
         .
         .
}
```

(예 6) Structure를 포함한 경우의 Java 프로그램 세그먼트
(example 6) Java program segment with structure type

예제 프로그램 (예 4) (예 5) (예 6)을 비교하여 보자. 세 프로그램 모두 time 이라는 자료구조를 다루고 있다. Time 자료구조는 정수 변수 hour, minute, 그리고 second로 구성된다. (예 4)은 C++ 언어를 이용하여 프로그램을 작성한 것이다. C++ 언어의 structure data type을 이용하여 hour, minute, 그리고 second 가 하나의 Time 이라는 structure로 구성되어 있다. (예 5)는 현재의 Java 언어를 이용하여 작성한 프로그램이다. Time 자료구조를 class를 이용하여 정의하였으며 정수 변수 hour, minute, 그리고 second 가 하나로 구성되어 있지 않고 개별적으로 구성되어 있다. 이들 정수 변수 hour, minute, 그리고 second가 서로 관계가 있다는 것은 오직 이들 변수의 이름만이 나타내고 있다. 만약 이들 세변수의 이름를 모두 바꾼다면 이들이 서로 관계가 있는 정수 변수라는 것을 파악하는 것은 전체 프로그램의 code를 살펴보지 않고서는 어렵게 된다. 이 것은 분명히 프로그램의 가독성(readability)을 저해하는 요인이 된다. Java 언어를 살펴보면 프로그램의 가독성을 향상시키기 위하여 노력한 흔적을 여러군데에서 발견할 수 있다. C 나 C++ 언어의 구문을 이어 받았으나 사용하기 어렵고 혼란을 야기시키는 구문을 삭제한 것이라든지, 의미의 전달이 매우 분명한 keyword의 사용이라든지, multiple inheritance의 삭제라든지등이 대표적인 것이다. 그러나 class type을 이용하여 structure type을 나타내려고 시도함으로서 프로그램의 가독성에 역행하는 결과를 초래한 것은 Java 언어의 일반적인 특징하고는 어울리지 않는 것이라고 생각한다.

더우기 본 논문 3.1절에서 3.5절까지에서 보여 주었듯이 structure type은 general-purpose 프로그래밍 언어에 있어서는 자료구조의 구축 측면에서 반드시 필요한 data type이다.

(예 6)은 structure type을 지원하는 Java 언어로 작성한 프로그램이다. 세 개의 정수 변수 hour, minute, 그리고 second가 structure로 구성되어 time이라는 이름으로 선언되었다. 그 결과 프로그램의 가독성이 훨씬 증가하였으며 자료구조의 구축이 훨씬 용이하여졌음을 알 수 있다. 더욱이 method를 작성할 때도 structure time을 직접 사용할 수 있기 때문에 프로그램의 구조도 훨씬 간결하여 졌음을 알 수 있다.

이상 3.1절에서 3.4절까지 일반 프로그래밍 언어에서 structure type을 성의하여 사용하는 이유에 대하

아 살펴보았으며 3.5절에서 structure type의 형식적 정의에 대하여서도 설명하였다. 3.6절에서 object-oriented 언어의 의미 모델을 고려하여 class type의 개념과 structure type의 개념은 다르다는 것을 확인하였다. 그리고 이절에서 Java 언어 class type의 문제점, class 개념과 entity 개념의 차이점, 그리고 entity 개념과 structure 개념의 유사점 등을 설명하였다. 그리고 Java 언어를 확장하여 structure를 정의하여 사용하는 예(예 6)를 들어 structure type을 사용할 경우 프로그램을 작성하기가 용이하여지며 프로그램의 구조도 명확하여짐을 보였다.

이상의 논의로부터 Java 언어에 class type 이외에 structure type을 별도로 정의하여 도입하는 것이 바람직하다는 결론에 도달하였다.

4. 결 론

Java 언어는 C++ 언어에 비하여 언어의 개념이 분명하고, 간결하며, 그리고 통일되어 있다. Java 언어의 구문도 간결하고 통일되어 있으며 keyword의 사용도 자연스러운 편이다. Java 언어의 연산자(operator)도 C++ 언어에 비하여 수가 적으며 분명하다. Java 프로그래밍 언어는 object-orientedness 개념을 최대한 적용한 언어이다. 모든 프로그램은 class로 구성된다. Class의 구조는 C++ 언어에 비하여 단순하며 class와 class를 구성하는 method들은 이전의 block-structured 프로그래밍 언어의 block의 모양을 많이 닮았다.

Java의 언어 개념(language concept)은 이 시대의 graphics, multimedia, internet 응용, client/server 응용을 효율적으로 지원할 수 있도록 설계되어 있다. 그리고 이를 위하여 Java 언어는 concurrency도 지원하고 있다. 이러한 여러 가지 응용들을 위한 API(Application Programming Interface)는 거의 Java 언어 내부요소로서 인식될 만큼 자연스럽게 사용할 수 있도록 지원되고 있다.

그리고 Java 언어는 보완(protection)과 이식성(portability)을 위하여 Java virtual machine 개념을 이용하여 목적코드를 생성한다. Java 언어를 이용한 프로그램 개발 비용은 C 언어나 C++ 언어에 비하여 큰 차이는 없으나 virtual machine을 대상으로 목적코드를 생성하기 때문에 프로그램의 수행에는 다소 긴 시간이 더 소요될 수 있다. 그러나 Java chip이 제작되어 사용되면 이 문제는 해결될 것으로 기대된다.

Java 언어는 JavaScript [16]와 같은 internet 응용을 위하여 설계된 언어에 비하면 일반 응용 프로그램도 개발할 수 있도록 설계되어 있다. 따라서 C 언어나 C++ 언어와 같이 사용될 수도 있다. 그러므로 Java 언어는 언어 개념의 간결성과 통일성 그리고 폭넓은 응용 프로그램 개발 환경의 지원으로 인하여 앞으로 폭넓게 사용될 수 있을 것으로 기대된다.

그러나 Java 언어에서 structure type과 class type 개념을 동일시 하여 structure type을 class type 속에 완전히 포함시킨 생각은 잘못된 생각이라고 판단한다. 본론에서 structure type에 대한 설명과 class type에 대한 설명에서 살펴보았듯이, class type과 strucutre type은 class type이 structure type을 포함하는 포함 관계에 있는 것이 아니라, 서로의 역할이 다르기 때문에 서로 보완 관계에 있는 개념이라는 것이 판명되었다. 따라서 Java 언어가 진정한 general-purpose 프로그래밍 언어가 되기 위하여서는 structure type을 class type에서 분리하여 새롭게 정의하여 지원하여야 한다고 생각한다.

참 고 문 헌

[1] Ken Arnold, James Gosling, The Java Programming Language, Addison-Wesley, 1996.

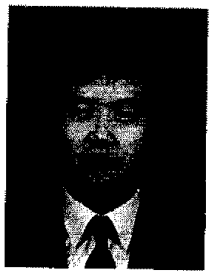
[2] James Gosling, Bill Joy, and Guy Steele, The Java Language Specification, Addison-Wesley, 1996.

[3] Tim Lindholm, Frank Yellin, The Java Virtual Machine Specification, Addison-Wesley, 1997.

[4] James Gosling, Frank Yellin, The Java Application Programming Interface, Volume 1, Core Package, Addison-Wesley, 1997.

[5] Harvey. M. Deital, Paul. J. Deitel, Java How to Program, Prentice-Hall Inc., 1997.

[6] Brian Kernighan, Dennis Ritchie, The C Programming Language, Prentice-Hall Inc., 1978.
[7] Bjarne Stroustrup, The C++ Programming Language (2nd edition), Addison-Wesley, 1991.
[8] Ira Pohl, Object-Oriented Programming Using C++ (2nd edition), Addison-Wesley, 1997.
[9] Stanley B. Lippman, C++ Primer (2nd edition), Addison-Wesley, 1995.
[10] Harvey M. Deitel, Paul J. Deitel, C++ How to Program, Prentice-Hall Inc., 1994.
[11] Richard Wiener, Lewis Pinson, An Introduction to Objected-Oriented Programming and C++, Addison-Wesley, 1988.
[12] James Martin, Principles of Object-Oriented Analysis and Design, Prentice-Hall Inc., 1993.
[13] G. Schneider, Steven Weingart, and David Perlman, An Introduction to Programming and Problem Solving with Pascal, John Wiley and Sons, 1978.
[14] Suad Alagic, Michael Arbib, The Design of Well-Structured and Correct Programs, Springer-Verlag, 1978.
[15] Guy Steele, et al, Common Lisp The Language, Digital Press, 1984.
[16] Arman Danesh, JavaScript, Sams.net Publishing, 1996.
[17] Terrence Pratt, Marvin Zelkowitz, Programming Languages Design and Implementation (3rd edition), Prentice-Hall Inc., 1996.
[18] Robert Sebesta, Concepts of Programming Languages (3rd edition), Addison-Wesley, 1996.
[19] Chris Date, An Introduction to Database Systems (6th edition), Addison-Wesley, 1996.
[20] Alfons Kemper, Guido Moerkotte, Object-Oriented Database Management : Applications in Engineering and Computer Science, Prentice-Hall Inc., 1994.
[21] Harvey M. Deitel, An Introduction to Operating Systems (2nd edition), Addison-Wesley, 1990.
[22] Abraham Silberschatz, Paul Galvin, Operating System Concepts (4th edition), Addison-Wesley, 1994.
[23] Carl Gunter, John Mitchell, Theoretical Aspects of Object-Oriented Programming, The MIT Press, 1994.
[24] Robert Orfali, Dan Harkey, Jeri Edwards, Instant CORBA, John Wiley & Sons, Inc., 1997.
[25] Robert Orfali, Dan Harkey, Client/Server Programming with Java and CORBA, John Wiley & Sons, Inc., 1997.



이 호 석

1983년 2월 서울대학교 전자계산학과 졸업 (공학 학사)
1985년 2월 서울대학교 컴퓨터공학과 졸업 (공학 석사)
1993년 8월 서울대학교 컴퓨터공학과 졸업 (공학 박사)
1994년 3월~현재 호서대학교 전자계산학부 조교수
관심분야 : 멀티미디어 시스템, 컴퓨터 그래픽스, 프로그래밍 언어론, Visual 프로그래밍, 데이터베이스 시스템