Generics ใน Java: ผู้ใช้มีสิทธิเลือกชนิดข้อมูล

อ. เนรมิต ชุมสาย ณ อยุธยา*

Abstract

Those who have been programming for years with OOP in language such as C++ when moved to earlier version of Java often found themselves in need of ways to represent type of data only when user specified it. Luck is on our side, in Java 1.5 (Tiger), we now have an ability to use Generics where user can use any type of data they want in their programs. The aim of this article is to introduce readers to understand the concept of Generics and how to use it to benefit those who use their programs.

คำสำคัญ

Generics

บทนำ

Java เริ่มแนะนำเรื่องของ Generics มาตั้งแต่รุ่น 5.0 ซึ่งเป็นการตัดสินใจที่ทำให้เรา (ผู้ใช้ภาษา Java) มีอีกทางเลือกหนึ่งในการทำให้โปรแกรมมีความยืดหยุ่นเพิ่มขึ้น กล่าวคือเราสามารถประกาศใช้ชนิดของข้อมูลที่เราไม่อาจรู้ได้ในเวลาที่เขียน code แต่จะได้มาเมื่อผู้ใช้บอกถึงชนิดของข้อมูลนั้นเอง ซึ่งในการแนะนำในครั้งนั้น Java ก็ได้เพิ่มโครงสร้างใหม่ๆ ตามมาด้วย เช่น boxing, unboxing, method ที่รองรับ parameter ได้โดยไม่จำกัดจำนวน (Varargs) และ for-each/loop เราคงจะไม่พูดถึงโครงสร้างเหล่านี้อย่างละเอียด คงเป็นเพียงแค่การนำมาใช้กับตัวอย่างในเรื่องของ Generics เท่านั้น และเมื่อพูดถึง Generics แล้วเราก็คงต้องพูดถึง Java Collections ด้วยเหมือนกัน ทั้งนี้ก็เพราะว่า Generics และ Collections ทำงานควบคู่ไปด้วยกันได้เป็นอย่างดี โดยเฉพาะการนำ Generics มาใช้อย่างมากมายในการออกแบบ Collections ต่างๆ ของ Java

Generics

Generics คืออะไร? ในการเขียนโปรแกรมนั้นหลายๆ ครั้งที่เราต้องการให้การทำงานของโปรแกรมรองรับข้อมูลได้หลากหลายชนิด โดยไม่ต้องเขียน code รองรับข้อมูลชนิดนั้นๆ โดยเฉพาะ ในภาษาของการเขียนโปรแกรม (โดยเฉพาะ Java) เราเรียกว่า การสร้างความเป็น abstract ให้กับข้อมูล[1] ซึ่งถ้าจะพูดให้ง่ายต่อการเข้าใจก็คือ "การยอมให้ข้อมูลเป็นได้ตามความพอใจของผู้ใช้" เพื่อให้เห็นภาพของ Generics ใน Java เราจะเริ่มต้นด้วยการเรียกใช้ class List อย่างง่ายๆ ดังที่แสดงให้ดูในโปรแกรมที่ 1

```
 8: public class SumOf {
 9:     public static void main(String[] args) {
10:       List<Integer> ints = Arrays.asList(12, 13);
11:       int sum = 0;
12:       for(int ele : ints) {
13:         sum += ele;
14:       }
15:       System.out.printf("%d%n", sum);
16:
17:       List<Double> doubles = Arrays.asList(11.30, 13.45, 15.25);
18:       double result = 0;
19:       for(double ele : doubles) {
20:         result += ele;
21:       }
22:       System.out.printf("%.2f%n", result);
23:
24:     }
25: }
```

โปรแกรมตัวอย่างที่ 1: การใช้ class List อย่างง่ายๆ

* คณบดีคณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยฟาร์อีสเทอร์น

[1] ในภาษา C++ โครงสร้างที่ใกล้เคียงกับ Generics ใน Java ก็คือ templates

โปรแกรมตัวอย่างที่เราแสดงให้ดูเรียกใช้ Collection ชื่อ List ของ Java ในบรรทัดที่ 10 และ 17 ซึ่งเป็นการกำหนดให้ตัวแปร ints และตัวแปร doubles เป็นที่เก็บข้อมูลชนิด Integer และ Double ตามลำดับ ถ้าผู้อ่านจำได้ทั้งสองชนิดเป็น object (reference ที่เกิดจาก wrapper class) ดังนั้นเราจึงไม่สามารถใช้เครื่องหมายการประมวลผลเชิงคณิตศาสตร์ได้แต่เนื่องจากว่ากระบวนการเปลี่ยนให้ object เป็น primitive types[2] นั้นได้ถูกกำหนดโดยอัตโนมัติจาก Java ที่เราเรียกว่า unboxing (และเราเรียกกระบวนการเปลี่ยน primitive types ให้เป็น reference ว่า boxing) เราจึงสามารถทำการบวกเลขที่มีอยู่ใน List ได้ และการประกาศของทั้งสองบรรทัดยังได้แสดงถึงการใช้ Varargs โดยในบรรทัดที่ 10 เราเรียกใช้ method asList() ของ class Arrays ด้วยการส่งค่าที่เป็น int 2 ค่าและในบรรทัดที่ 17 เราส่งค่าที่เป็น double 3 ค่า ผู้อ่านไม่ต้องกังวลถึงการเปลี่ยนค่า และจำนวนของ parameter ที่ต้องส่งไปเพราะมันจะได้รับการสนองตอบแบบอัตโนมัติ ส่วน for/loop ที่เห็นทั้งสองตัวนั้นเป็น for-each/loop ที่ช่วยให้การเข้าหาข้อมูลแต่ละตัวทำได้ง่ายขึ้น (เมื่อเปรียบเทียบกับการเขียนแบบเก่า)

ย้อนกลับไปในบรรทัดที่ 10 และ 17 การประกาศตัวแปร ints และ doubles นั้นเรามีคำว่า Integer และ Double อยู่ในเครื่องหมาย <> ตามหลังคำว่า List เราเรียกการประกาศแบบนี้ว่าเป็นการประกาศแบบกำหนดชนิดข้อมูล ซึ่งจะทำได้ก็ต่อเมื่อ class ที่ใช้ได้ถูกสร้างขึ้นเพื่อรองรับชนิดข้อมูลในแบบที่เราเรียกว่า Generics เท่านั้น ทั้งนี้และทั้งนั้นผู้สร้าง class จะเป็นผู้กำหนดถึงชนิดของข้อมูลต่างๆ ที่ class สามารถรองรับได้

Interface List ได้ถูกออกแบบให้รองรับความหลากหลายของข้อมูลดังนี้ Interface List<E> { ... } เพราะฉะนั้น class ต่างๆ ที่ออกแบบด้วยการใช้ interface List จึงสามารถรองรับข้อมูลได้หลายชนิดดังที่เราแสดงให้ดูในโปรแกรมตัวอย่างก่อนหน้านี้ หรือเช่นตัวอย่างนี้

```
List<String> words = new ArrayList<String>();
words.add("Chiang Mai");
words.add("Lam Phun");
List<Character> alphabets = new ArrayList<Character>();
alphabets.add('Z');
alphabets.add('Q');
Character ch = alphabets.get(0);
```

แต่ถ้าเป็น Java รุ่นก่อนที่จะมี Generics เราไม่สามารถที่จะดึงออกได้ทันทีเราต้องทำการ cast ข้อมูลให้เป็นไปตามที่เราต้องการ ดังนี้

```
List alphabets = new ArrayList();
alphabets.add('Z');
alphabets.add('Q');
Character ch = (Character)alphabets.get(0);
```

ถ้าเรามองดูในระดับที่ลึกลงไปถึง byte code เราจะเห็นว่า code ทั้งสองตัวเหมือนกัน Java จะเปลี่ยนให้รูปแบบของ Generics มาเป็นการ cast กระบวนการดังกล่าวเราเรียกว่า erasure ซึ่งหมายถึงการที่โปรแกรมทำการลบ type parameters ออกและทำการ cast แทน เพราะฉะนั้น List<Integer>, List<Double>, List<List<String>> จะถูกเปลี่ยนให้เป็น List ในขณะที่กำลังถูกประมวลผล (run)

ในการประกาศตัวแปร alphabets นั้นเราได้กำหนดให้เป็น list ที่เก็บเฉพาะข้อมูลที่เป็น Character เท่านั้น (ด้วยการเขียนดังนี้ List<Character> alphabets) ดังนั้น complier ก็จะรู้ถึงชนิดของข้อมูลที่ได้ถูกกำหนดไว้ การตรวจสอบข้อมูลที่เกิดขึ้น (ตอน run time) ก็จะการันตีถึงความถูกต้อง ซึ่งต่างกับการ cast ที่บอกให้เรารู้ว่าโปรแกรมเมอร์คิดว่าชนิดของข้อมูลเป็นข้อมูลที่ถูกต้อง ณ

[2] Java มี primitive types อยู่ 8 ตัวคือ byte, short, int, long, float, double, bool, และ char ส่วน wrapper class ของทั้งแปดตัวคือ Byte, Short, Integer, Long, Float, Double, Boolean, และ Character

เวลานั้นเท่านั้น การประกาศในรูปแบบของ generic นั้นเมื่อมีการ compile แล้วก็จะไม่มีการ compile ซ้ำอีก หมายความว่าเราจะได้ไฟล์ที่มีนามสกุล .class เพียงตัวเดียว ถึงแม้จะมีการเรียกด้วย type parameter (อาจเรียกว่า type variable ก็ได้) ที่ต่างกันทั้งนี้ก็เพราะว่า compiler จะทำการแทน type parameter (เช่น T, E เป็นต้น) ด้วย type argument (เช่น String, Integer เป็นต้น) ที่ส่งเข้ามาทุกครั้งของการเรียกใช้

โปรแกรม BucketOfItems[3] เป็นตัวอย่างการใช้ Generic ในการออกแบบ class

```
 7: class BucketOfItems<T> {
 8:     private final int MAX = 3; //initial capacity
 9:     private T[] bucket; //generic container
10:     private int count; //number of items in container
11:
12:     public BucketOfItems() {
13:         bucket = (T[])new Object[MAX];
14:         count = 0;
15:     }
16:
17:     //getting container size
18:     public int size() {
19:         return count;
20:     }
21:
22:     //add item into container
23:     public void add(T value) {
24:         //double container size if necessary
25:         if(count > bucket.length - 1) {
26:             T[] temp = (T[])new Object[MAX * 2];
27:             //copy items from bucket to temp
28:             System.arraycopy(bucket, 0, temp, 0, bucket.length);
29:             //reassign bucket to temp
30:             bucket = temp;
31:         }
32:         //add new item
33:         bucket[count++] = value;
34:     }
35:
55:     //get middle item (the easy way)
56:     public <T> T getMiddleValue() {
57:         return (T) bucket[bucket.length / 2];
58:     }
59:
60:     //find largest item
61:     public <T extends Comparable> T largestItem() {
62:         if(bucket.length == 0)
63:             return null;
64:         T largest = (T)bucket[0];
65:         for(int i = 1; i < bucket.length; i++) {
66:             if(largest.compareTo(bucket[i]) < 0)
67:               largest = (T)bucket[i];
68:         }
69:         return largest;
70:     }
71:
72:     //add in ascending order
73:     public <T extends Comparable> void addInorder(T value) {
74:         int i = 0;
75:         for(; i < count; i++) {
76:             T item = (T)bucket[i];
77:             //found a spot for insertion
78:             if(item.compareTo(value) > 0)
79:               break;
80:         }
```

[3] code บางส่วนได้ถูกตัดออก - ดูเอกสารอ้างอิงที่ 9

```
81:         //move each object forward one position
82:         for(int j = count; j > i; j--)
83:             bucket[j] = bucket[j-1];
84:
85:         //insert new object
86:         Arrays.fill(bucket, i, i+1, value);
87:         count++;
88:     }
89:
```

โปรแกรมตัวอย่างที่ 2: การสร้าง Generic Array

Class BucketOfItems เมื่อมองผิวเผินแล้วอาจดูซับซ้อน แต่จริง ๆ แล้วไม่เป็นเช่นนั้น เราจะอธิบายถึง code ภายในอย่างละเอียด ถึงแม้ว่าการเขียน code ในหลายๆ ส่วนอาจดูแปลกตาไปบ้างแต่เมื่อทราบถึงที่มาที่ไปแล้วผู้อ่านหลายๆ ท่านก็จะถึงบางอ้อทันที

เราจะเริ่มด้วยการประกาศของตัว class เอง

```
 7: class BucketOfItems<T> {
 8:     private final int MAX = 3;  //initial capacity
 9:     private T[] bucket;         //generic container
10:     private int count;          //number of items in container
```

เรากำหนดให้ class BucketOfItems มีตัวแปรที่เรียกว่า type variable[4] ด้วยการใช้ตัวอักษร T (ตัวใหญ่) เป็นตัวกำหนด ซึ่งเราสามารถใช้ได้ทุกที่ภายใน class ถ้าจะเปรียบเทียบแบบง่ายๆ T ก็หมายถึงชนิดของข้อมูลที่เราจะนำมาใช้ภายใน class เมื่อโปรแกรมถูกประมวลผล (run time execution) ซึ่งโดยการออกแบบแล้วเราไม่สามารถส่ง primitive types ไปให้ class BucketOfItems ได้จะเป็นได้เฉพาะ class type, interface type, หรือ type variable เท่านั้น ดังนั้นถ้าเราเรียกใช้ด้วย

```
BucketOfItems<int> bucket = new BucketOfItems<int>();
```

เราก็จะได้รับการฟ้องจาก Java ทันที ดังนี้

```
TestGenericArray.java:8: unexpected type
found   : int
required: reference
        BucketOfItems<int> bucket = new BucketOfItems<int>();
                      ^
TestGenericArray.java:8: unexpected type
found   : int
required: reference
        BucketOfItems<int> bucket = new BucketOfItems<int>();
                                                      ^
Note: .\BucketOfItems.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
2 errors
```

[4] Type variable ถ้าจะแปลให้ได้ความหมายก็น่าจะแปลว่า "ตัวแปรที่บอกถึงชนิดข้อมูลที่นำมาใช้"

ถ้าเรากำหนดให้ BucketOfItems รองรับข้อมูลชนิดใดเราก็ไม่สามารถเปลี่ยนแปลงให้เป็นชนิดอื่นได้ดังตัวอย่างนี้

```
BucketOfItems<Integer> bucket1 = new BucketOfItems<Integer>();
bucket1.add(34);
bucket1.add(45.45); //ERROR!
```

บรรทัดสุดท้ายของ code ด้านบนนี้ทำการนำข้อมูลที่เป็น double เข้าสู่ bucket1 ซึ่งประกาศให้รองรับข้อมูลที่เป็น int เท่านั้น ดังนั้นจึงเกิด error ขึ้น (compile ไม่ผ่าน)

ในบรรทัดที่ 9 ผู้อ่านจะเห็นการประกาศตัวแปร bucket ดังนี้ private T[] bucket; ซึ่งเมื่อมองดูแล้วก็เป็นการประกาศให้ bucket เป็น array ที่ใช้เก็บข้อมูลชนิด T ซึ่งก็ไม่น่าจะมีปัญหาอะไร แต่เมื่อเราทำการจองเนื้อที่ให้กับ bucket ในบรรทัดที่ 13 ดังนี้

```
13:  bucket = (T[ ])new Object[MAX];
```

ผู้อ่านอาจถามว่าทำไมเราถึงไม่กำหนดด้วยประโยค bucket = new T[MAX]; เหมือนกับที่เราคุ้นเคย สาเหตุก็เพราะว่า Java ไม่ยอมให้เราสร้าง array ที่ใช้เก็บ Generic type ทั้งนี้ก็อาจมาจากการที่ Java ต้องรู้ถึงชนิดของข้อมูลก่อนที่จะจองเนื้อที่ให้กับ array และเมื่อเรา compile ด้วย option -Xlint:unchecked เราก็จะเห็นการเตือนของ Java ดังนี้

```
BucketOfItems.java:13: warning: [unchecked] unchecked cast
found   : java.lang.Object[]
required: T[]
        bucket = (T[])new Object[MAX];
                      ^
```

อย่างไรก็ตามเราก็หลีกเลี่ยงด้วยการจองเนื้อที่ให้กับ array ที่ใช้เก็บ Object ก่อนแล้วจึงทำการ cast กลับมาให้เป็น T[ ]

ข้อกำหนดการใช้ type parameter

โดยทั่วไปแล้วเราจะใช้อักษรตัวใหญ่ตัวเดียวเป็นตัวกำหนดชนิดข้อมูล ซึ่งอาจดูแล้วไม่เป็นไปตามที่เรียนมาว่าชื่อตัวแปรหรือการกำหนดสัญลักษณ์แทนข้อมูลควรเป็นคำที่สื่อถึงข้อมูลนั้นๆ อย่างไรก็ตามการใช้อักษรตัวเดียวทำให้ง่ายต่อการแยกแยะว่าสิ่งที่ส่งมาให้ Generic class เป็น type variable, class, หรือเป็น interface โดยทั่วไปแล้วเราใช้ตัวอักษรต่อไปนี้

E หมายถึง element (ใช้มากใน Java Collection Framework)

K หมายถึง key

N หมายถึง number

T หมายถึง type

V หมายถึง value

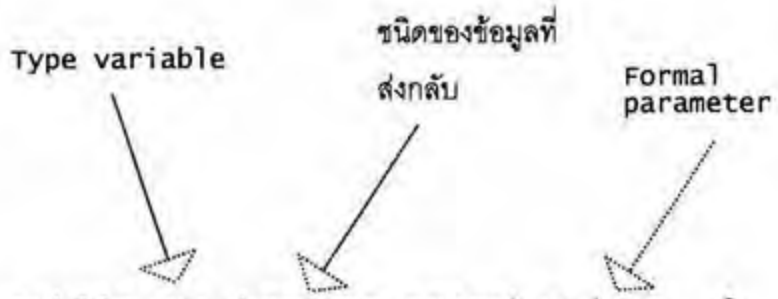
S, U, V หรืออื่น ๆ หมายถึงชนิดอื่น ๆ ที่นำมาใช้

Generic methods

ภายใน class BucketOfItems ของเรามี method หลายตัวที่มีหน้าตาไม่เหมือนกับ method ที่เราคุ้นเคยกัน เช่น

```
56:     public <T> T getMiddleValue() {
57:       return (T) bucket[bucket.length / 2];
58:     }
```

ผู้อ่านจะเห็นว่าในบรรทัดที่ 56 เรามี <T> อยู่ด้านหน้า T ซึ่งอยู่ด้านหน้าชื่อ method getMiddleValue() อีกที ซึ่งถ้าเราลองตรวจดูเราก็คงจะเดาได้ว่าตัว T โดด ๆ นั้นเป็นชนิดของข้อมูลที่ method ต้องส่งกลับออกไปยังผู้ที่เรียกใช้ (return type) แต่ตัว <T> เองนั้นหมายถึงอะไร ? จะพูดง่าย ๆ <T> หมายถึง type variable ที่เราต้องกำหนดให้มีถ้าเราต้องการให้ method ที่เราสร้างขึ้นเป็น generic method รูปที่ 1 แสดงโครงสร้างของ generic method



รูปที่ 1 โครงสร้างของ Generic method

เราลองมาดู method อีกตัวหนึ่งที่เป็น generic method

```
61:     public <T extends Comparable> T largestItem() {
62:       if(bucket.length == 0)
63:           return null;
64:       T largest = (T)bucket[0];
65:       for(int i = 1; i < bucket.length; i++) {
66:           if(largest.compareTo(bucket[i]) < 0)
67:               largest = (T)bucket[i];
68:       }
69:       return largest;
70:     }
```

Method largestItem() เป็น method ที่มี type variable เป็น <T extends Comparable> ซึ่งก็หมายความว่า ชนิดข้อมูลจะต้องเป็นชนิด T ที่เป็น sub-type ของ Comparable เท่านั้น ส่วน T ที่ตามมาเป็นชนิดของข้อมูลที่ต้องส่งกลับ เรามาลองดูตัวอย่างอีกตัวหนึ่งที่เกี่ยวกับ generic method

```
 7: public class GenericMethodDemo4 {
 8:     //generic method returning List<T> for any type T
 9:     public static <T> List<T> toList(T[] array) {
10:        List<T> list = new ArrayList<T>();
11:        for(T item : array) {
12:          list.add(item);
13:        }
14:        return list;
15:     }
16:
17:     public static void main(String[] args) {
18:        List<String> list1 = toList(new String[] {"Generic", "methods"});
19:        System.out.println(list1);
20:        List<Double> list2 = toList(new Double[] {23.45, 3.45, 2.11});
21:        System.out.println(list2);
22:     }
23: }
```

โปรแกรมตัวอย่างที่ 3: Generic Method

โปรแกรม GenericMethodDemo4.java แสดงการกำหนดให้ method toList() เป็น Generic method ที่ทำการดึงเอาข้อมูลที่มีอยู่ใน array ทั้งหมดเข้าสู่ list โดยกำหนดให้ list เป็น list ที่เก็บข้อมูลชนิด T (ใช้ค่าส่งกลับเป็น List<T>) พร้อมทั้งกำหนดให้ T เป็นอะไรก็ได้ด้วยการกำหนด <T> ที่อยู่ด้านหน้าของ List<T>

มาถึงตรงนี้ผู้อ่านที่มีคำถามอยู่ในใจว่าทำไมถึงต้องเป็น Generic method ก็คงหาคำตอบได้ด้วยตัวเองแล้ว ทั้งนี้ก็เพราะว่าถ้าเราไม่กำหนดชนิดของข้อมูลด้วยการนำ <T> มาใส่ไว้ด้านหน้า List<T> โปรแกรมที่เราเขียนขึ้นก็จะฟ้องเราว่าไม่รู้จักว่า T คืออะไร ดังที่แสดงให้ดูด้วย error นี้

```
GenericMethodDemo4.java:9: cannot find symbol
symbol  : class T
location: class GenericMethodDemo4
    public static  List<T> toList(T[] array) {
                           ^
GenericMethodDemo4.java:9: cannot find symbol
symbol  : class T
location: class GenericMethodDemo4
    public static  List<T> toList(T[] array) {
                            ^
GenericMethodDemo4.java:10: cannot find symbol
symbol  : class T
location: class GenericMethodDemo4
        List<T> list = new ArrayList<T>();
                ^
GenericMethodDemo4.java:10: cannot find symbol
symbol  : class T
location: class GenericMethodDemo4
        List<T> list = new ArrayList<T>();
                                     ^
GenericMethodDemo4.java:11: cannot find symbol
symbol  : class T
location: class GenericMethodDemo4
        for(T item : array) {
```

การกำหนดแบบนี้ของ method toList() ทำให้ scope ของ T อยู่ภายใน method toList() เท่านั้นเราไม่สามารถใช้ T ในที่อื่นๆ ได้ เพราะฉะนั้นถ้าเมื่อใดก็ตามที่เราต้องเขียน method ที่ทำงานกับข้อมูลที่เราต้องการให้เป็น generic เราก็ต้องกำหนด type variable ให้กับ method เหล่านั้นด้วย

Generic method กับ Varargs

Varargs เป็นโครงสร้างที่ยอมให้จำนวนของ parameter ที่อยู่ใน method เป็นไปได้ตามใจของผู้เรียกใช้ เราสามารถที่จะนำ varargs มาใช้กับ generic method ได้ดังตัวอย่างที่เราได้ปรับปรุงจากโปรแกรมก่อนหน้านี้ด้วยการเปลี่ยน parameter ใน method toList() ให้เป็น T... array ดังที่เห็นนี้

```
 9:     public static <T> List<T> toList(T... array) {
10:        List<T> list = new ArrayList<T>();
11:        for(T item : array) {
12:           list.add(item);
13:        }
14:        return list;
15:     }
```

การเรียกใช้ method ก็ทำได้ง่ายขึ้น เช่น

```
18:        List<String> list1 = toList("Generic", "methods");
19:        System.out.println(list1);
20:        List<Double> list2 = toList(23.45, 3.45, 2.11);
21:        System.out.println(list2);
```

ผลลัพธ์ที่เราได้จากการ run โปรแกรมตัวอย่างนี้คือ

```
[Generic, methods]
[23.45, 3.45, 2.11]
```

ผู้อ่านจะเห็นว่าในบรรทัดที่ 18 เราส่ง parameter 2 ตัวให้กับ toList() แต่ในบรรทัดที่ 20 เราส่งไปให้ 3 ตัวซึ่งเป็นการแสดงให้เห็นถึงความสะดวกสบายของการใช้ method ที่เขียนแค่ตัวเดียวแต่รองรับการเรียกใช้หลายรูปแบบ ซึ่งถ้าไม่มี varargs เราก็ต้องใช้วิธีการที่เรียกว่า method overloading แทน

Sub-typing กับ Generics

เรารู้อยู่แล้วว่าเราสามารถที่จะกำหนดให้ object ชนิดหนึ่งเป็น object อีกชนิดหนึ่งได้ถ้าเราได้กำหนดให้ชนิดทั้งสองของ object เป็นชนิดที่สามารถใช้ร่วมกันได้ (compatible) เช่น เราสามารถกำหนดค่า Integer เข้าสู่ Object ได้ทั้งนี้ก็เพราะว่า Object เป็น super-class ของ Integer ดังตัวอย่างนี้

```
Object x = new Object();
Integer y = 23;
x = y;
```

sub-typing เป็นคุณสมบัติที่เป็นจุดเด่นของการเขียนโปรแกรมในรูปแบบที่เราเรียกว่า Object-Oriented Programming ในภาษา Java นั้นเรากำหนดให้ชนิดของข้อมูล (type) เป็น sub-type ของข้อมูลตัวอื่นด้วยการใช้ keyword: extends หรือ implements เช่น

Integer เป็น sub-type ของ Number
ArrayList<E> เป็น sub-type ของ List<E>
List<E> เป็น sub-type ของ Collection<E>

ถ้าเรามองย้อนไปดู method largestItem() ของ class BucketOfItems เราะเห็นถึงการใช้ sub-type ที่ว่า <T extends Comparable> ซึ่งให้ความหมายว่า T จะเป็นข้อมูลชนิดใดก็ได้ที่เป็น sub-type ของ Comparable

เราอาจออกแบบให้ class ของเรารองรับข้อมูลที่เป็น sub-type ของอะไรก็ได้ที่เราต้องการ เช่น เราอาจออกแบบให้ class Exchange<Number> เป็น class ที่รองรับข้อมูลที่เป็น sub-type ของ Number ซึ่งทำให้เราสามารถสร้าง object ได้หลายชนิด เช่น

```
Exchange<Number> exchange = new Exchange<Number>();
Exchange.add(new Double(244.0));
exchange.add(new Integer(244));
```

ตัวอย่างด้านบนนี้จะไม่สร้างปัญหาให้เราถ้าเราสร้าง method add() ที่มี parameter ที่เป็น T เช่น add(T someNumber) แต่อาจทำให้ผู้อ่านคิดว่าเมื่อ Integer เป็น sub-type ของ Number แล้วจะทำให้ Exchange<Integer> เป็น sub-type ของ Exchange <Number> ด้วย แต่จริง ๆ แล้วไม่ได้เป็นอย่างนั้น สมมติว่าเรามี method นี้

```
public void addMore(Exchange<Number> number) { … }
```

method ที่เห็นนี้รับข้อมูลที่เป็น Exchange<Number> แต่ไม่ได้บอกว่าจะรับข้อมูลที่เป็น Exchange<Integer> หรือ Exchange <Double> ดังนั้นเราจึงไม่สามารถส่ง parameter อื่น ๆ ที่ไม่ใช่ Exchange<Number> ไปให้ method ตัวนี้ได้ เพื่อให้เห็นภาพได้ชัดเจนยิ่งขึ้นผู้อ่านควรทดลองด้วยโปรแกรมต่อไปนี้

```
 7: public class SubtypeDemo1 {
 8:     public static void main(String[] args) {
 9:         //these are OK
10:         ArrayList<Integer> ints = new ArrayList<Integer>();
11:         ArrayList<String> str = new ArrayList<String>();
12:         ArrayList<Object> obj1 = new ArrayList<Object>();
13:
14:         //these are NOT OK
15:         ArrayList<Object> obj2 = new ArrayList<String>();
16:         ArrayList<Object> obj3 = new ArrayList<Integer>();
17:
18:         List<String> strs = new ArrayList<String>();
19:         //this is NOT OK
20:         List<Object> objs = new ArrayList<String>();
21:
22:         List<Number> nums1 = new Vector<Number>();
23:         //these are NOT OK
24:         List<Number> nums2 = new Vector<Integer>();
25:         List<Number> nums3 = new ArrayList<Long>();
26:     }
27: }
```

โปรแกรมตัวอย่างที่ 4: Subtyping กับ Generic

หลังจากที่เรา compile, error ที่เราได้คือ

```
SubtypeDemo1.java:15: incompatible types
found   : java.util.ArrayList<java.lang.String>
required: java.util.ArrayList<java.lang.Object>
      ArrayList<Object> obj2 = new ArrayList<String>();
                               ^
SubtypeDemo1.java:16: incompatible types
found   : java.util.ArrayList<java.lang.Integer>
required: java.util.ArrayList<java.lang.Object>
      ArrayList<Object> obj3 = new ArrayList<Integer>();
                               ^
SubtypeDemo1.java:20: incompatible types
found   : java.util.ArrayList<java.lang.String>
required: java.util.List<java.lang.Object>
      List<Object> objs = new ArrayList<String>();
                          ^
SubtypeDemo1.java:24: incompatible types
found   : java.util.Vector<java.lang.Integer>
required: java.util.List<java.lang.Number>
      List<Number> nums2 = new Vector<Integer>();
                           ^
SubtypeDemo1.java:25: incompatible types
found   : java.util.ArrayList<java.lang.Long>
required: java.util.List<java.lang.Number>
      List<Number> nums3 = new ArrayList<Long>();
                           ^
5 errors
```

ผู้อ่านจะเห็นว่า error ที่ได้จะบอกถึงชนิดของข้อมูลที่ไม่สามารถใช้ร่วมกันได้ (incompatible types) ซึ่งก็น่าจะบอกให้เรารู้ว่าเราไม่อาจตั้งสมมติฐานได้ว่า ถ้า X เป็น sub-type ของ Y จะทำให้ A<X> เป็น sub-type ของ A<Y> ได้ วิธีการที่จะทำให้การเลือกใช้ชนิดของข้อมูลเป็นไปได้กว้าง (ยืดหยุ่น) ยิ่งขึ้นก็ต้องใช้ wildcards เข้ามาช่วย

การใช้ Wildcards กับ Generics

การออกแบบ method (หรือส่วนอื่น ๆ) ที่ยอมให้มีการเรียกใช้ได้กับข้อมูลหลายชนิดนั้นทำได้หลายวิธี และวิธีหนึ่งที่ทำได้ง่ายๆ คือ การใช้ wildcards ลองมาดูตัวอย่างกัน สมมติว่าเราต้องการแสดงผลข้อมูลที่มีอยู่ใน Collection โดยต้องการให้แสดงได้หลายชนิด เราอาจใช้ Object เป็นชนิดข้อมูลที่ Collection เก็บอยู่ทั้งนี้ก็เพราะว่า Object เป็น class ที่อยู่บนสุดของทุก class ซึ่งน่าจะยอมให้เราเรียกใช้ข้อมูลชนิดอื่นที่อยู่ภายใต้ Object ด้วย ดังเช่น method print() ที่เห็นนี้

```
void print(Collection<Object> col) {
    for(Object obj : col) {
        System.out.println(obj);
    }
}
```

เมื่อเราลองเรียกใช้ด้วยประโยคต่อไปนี้

```
LinkedList<String> list = new LinkedList<String>();
list.add("Chiang Mai");
list.add("Chaing Rai");
list.add("Lampoon");

print(list);
```

เราก็จะได้รับการฟ้องจาก compiler ถึงการใช้ Object กับ Sting ซึ่งเป็นสิ่งที่ Java ไม่ยอม (ถึงตอนนี้ผู้อ่านคงเข้าใจมากแล้วว่า ถึงแม้ว่า X เป็น sub-type ของ Y ก็ไม่ได้บอกว่า A<X> เป็น sub-type ของ A<Y>) วิธีการหนึ่งที่เราสามารถนำมาใช้ได้คือ เปลี่ยนให้ parameter ใน method print() เป็น Collection<String> ซึ่งเป็นวิธีการที่ง่ายที่สุด แต่เราก็จะใช้ได้เฉพาะข้อมูลที่เป็น String เท่านั้น เราไม่สามารถใช้กับข้อมูลชนิดอื่นได้ วิธีการที่สามารถนำมาใช้ได้อีกวิธีการหนึ่ง คือ การสร้าง method มารองรับข้อมูลทุกชนิด (overloading) แต่ก็ทำให้เราต้องเขียน code เยอะขึ้น ดังนั้นเราต้องหันมาใช้สิ่งที่ดีกว่าที่ Java มีให้ นั่นก็คือการใช้ wildcard วิธีการก็ง่ายมาก เราเพียงแค่เปลี่ยน method print() ให้ใช้เครื่องหมาย "?" แทนชนิดข้อมูล ดังที่แสดงให้ดูนี้

```
void print(Collection<?> col) {
    for(Object obj : col) {
        System.out.println(obj);
    }
}
```

เท่านี้เราก็สามารถเรียกใช้ method print() ด้วยข้อมูลหลาย ๆ ชนิดได้ เช่น

```
ArrayList<Integer> ints = new ArrayList<Integer>();
ints.add(34);
ints.add(23);
print(ints);

ArrayList<Double> doubles = new ArrayList<Double>();
doubles.add(34.5);
doubles.add(23.5);
print(doubles);
```

ในภาษาอังกฤษเราเรียก Collection<?> ว่า "Collection of unknown" ซึ่งอาจแปลเป็นไทยได้ว่า "กลุ่มของข้อมูลที่ยังไม่บ่งบอกชนิด" เพราะฉะนั้นเมื่อเราส่งข้อมูลชนิดต่าง ๆ ไปให้ compiler ก็จะใช้ข้อมูลชนิดนั้น ๆ ได้โดยไม่เกิดปัญหาอะไร ผู้อ่านจะเห็นว่าภายใน method print() เราก็ยังคงใช้ Object ใน for/loop เหมือนเดิม ทั้งนี้ก็เพราะว่าไม่ว่า collection จะเก็บอะไรมันก็ยังเป็นการเก็บ Object อยู่ดี สิ่งที่เราทำไม่ได้ก็คือ นำ Object ที่เราไม่รู้ว่าเป็นชนิดอะไรเข้าสู่ collection เช่น

```
Collection<?> list = new ArrayList<Number>();
list.add(new Object());
list.add(new Integer());
```

ผู้อ่านจะเห็นว่า list เป็นที่เก็บข้อมูลที่ยังไม่บ่งบอกชนิด เราก็ไม่สามารถนำ Object เข้าสู่ที่เก็บตัวนี้ได้ ข้อมูลที่ส่งเข้าไปยัง method add() ต้องเป็น sub-type ของข้อมูลชนิดนี้ แต่เมื่อเราไม่รู้ถึงชนิดของข้อมูลเราก็ไม่สามารถส่ง Object ไปให้ add() ได้ สิ่งที่เราส่งได้ก็คือ null เท่านั้น (เช่น list.add(null);) อย่างไรก็ตามเราสามารถที่จะดึงข้อมูลที่ไม่บ่งบอกชนิดออกมาจากที่เก็บได้ แต่จะต้องนำเข้าสู่ตัวแปรที่เป็น Object หรือส่งไปให้กับ method ที่มี parameter เป็น Object เท่านั้น เช่น

```
Object item = list.get(0);
boolean result = isMemberOf(list.get(1));
```

ทั้งนี้ก็เพราะว่าข้อมูลที่อยู่ภายในต้องเป็น Object อยู่ดีนั่นเอง

การใช้ wildcards ที่มีขอบเขต (Bounded Wildcards)

เราสามารถที่จะกำหนดชนิดของข้อมูลให้อยู่ในกลุ่มเดียวกันได้ เช่น ข้อมูลที่เป็น sub-type ของ Number หรือข้อมูลที่เป็น sub-type ของ type อื่น ๆ ด้วยการใช้ wildcard ร่วมกับ extends ดังตัวอย่างต่อไปนี้

```
void print(Collection<? Extends Number> col) {
    for(Object obj : col) {
        System.out.println(obj);
    }
}
```

เมื่อเราเปลี่ยน method print() ให้เป็นดังที่เห็นด้านบนนี้ เราก็สามารถเรียกใช้ print() ด้วยข้อมูลชนิดใดก็ได้ที่เป็น sub-type ของ Number (ตัวแปร ints และ doubles ที่เห็นจากตัวอย่างก่อนหน้านี้ใช้ได้ แต่ตัวแปร list ไม่สามารถใช้ได้เพราะ list ที่เราสร้างขึ้นนั้นไม่อยู่ในกลุ่มของ Number แต่เป็น String)

เราไม่ได้ถูกจำกัดการใช้ wildcards ใน method เท่านั้นเราสามารถที่จะใช้ wildcards ในการประกาศตัวแปรได้เช่นกัน ดังตัวอย่างต่อไปนี้

```
List<? extends Number> numbers1 = Arrays.asList(1, 2, 3);
List<? extends Number> numbers2 = Arrays.asList(1.5, 2.5, 3.5);
```

ประโยคตัวอย่างที่เห็นไม่สร้างปัญหาให้เราเพราะว่าคำสั่ง Arrays.asList(1, 2, 3) ส่ง list ที่มี Integer ไปให้ numbers1 ซึ่งเป็น list ที่เป็น sub-type ของ Number เช่นเดียวกันกับประโยคที่ตามมา Arrays.asList(1.5, 2.5, 3.5) ส่ง list ที่มี Double ไปให้ numbers2 ซึ่งก็เป็น list ที่เป็น sub-type ของ Number เช่นกัน

เรารู้ว่าเครื่องหมาย "?" หมายถึง ข้อมูลที่ยังไม่บ่งบอกชนิด แต่เมื่อนำมาใช้กับคำว่า extends เช่น List<? extends Number> ก็ให้ความหมายถึงข้อมูลอะไรก็ได้ที่เป็น sub-type ของ Number (ในทางกลับกันเราเรียก Number ว่าเป็น super-type) ทำให้ขอบเขตของการใช้ข้อมูลอยู่ภายใน Number เท่านั้น หรือพูดง่าย ๆ ว่า Number เป็น upper bound ของการใช้ข้อมูล

```
 7: class Wildcard2 {
 8:     //generic method with bounded wildcard
 9:     static void print(Collection<? extends Number> col) {
10:         for(Object obj : col) {
11:             System.out.println(obj);
12:         }
13:     }
14:
15:     public static void main(String[] args) {
16:         ArrayList<Integer> ints = new ArrayList<Integer>();
17:         ints.add(34);
18:         ints.add(23);
19:         print(ints);
20:         ArrayList<Double> doubles = new ArrayList<Double>();
21:         doubles.add(34.5);
22:         doubles.add(23.5);
23:         print(doubles);
24:
25:         List<? extends Number> numbers1 = Arrays.asList(1, 2, 3);
26:         List<? extends Number> numbers2 = Arrays.asList(1.5, 2.5, 3.5);
27:         print(numbers1);
28:         print(numbers2);
29:     }
30: }
```

โปรแกรมตัวอย่างที่ 5: Wildcard

โปรแกรม Wildcard2.java ที่เห็นด้านบนนี้แสดงการใช้ wildcard ที่มีขอบเขตอยู่เฉพาะ class Number หรือ class อื่นๆ ที่ extends มาจาก class Number (เท่านั้น) เราไม่สามารถที่จะใช้ข้อมูลชนิดอื่น ๆ ได้ เช่น ถ้าเราใส่ประโยค ArrayList<String> str = new ArrayList<String>(); และเรียกใช้ method print() เราก็จะได้รับการฟ้องจาก Java ทันทีดังนี้

```
Wildcard2.java:31: print(java.util.Collection<? extends java.lang.Number>) in
Wildcard2 cannot be applied to (java.util.ArrayList<java.lang.String>)
        print(str);
        ^
1 error
```

ข้อผิดพลาดที่เราได้รับเป็นสิ่งที่มองเห็นได้ชัดเจน คือเราไม่สามารถรองรับข้อมูลที่ไม่อยู่ในกลุ่มที่เราได้กำหนดไว้ สมมติว่าเราอยากที่จะกำหนดให้ list ของเรารองรับข้อมูลได้หลากหลาย เราก็อาจคิดว่าทำไมเราไม่ประกาศให้ list ของเรารองรับข้อมูลที่เป็น "?" ไปเลยดังที่เราได้ทำมาก่อนหน้านี้ คำตอบก็เหมือนเดิมคือ ทำไม่ได้ เช่นตัวอย่างจากโปรแกรมที่เราได้เพิ่ม code เข้าไปนี้

```
 7: class Wildcard2 {
 8:     //generic method with bounded wildcard
 9:     static void print(Collection<? extends Number> col) {
10:         for(Object obj : col) {
11:             System.out.println(obj);
12:         }
13:     }
14:
15:     public static void main(String[] args) {
```

```
16:         ArrayList<Integer> ints = new ArrayList<Integer>();
17:         ints.add(34);
18:         ints.add(23);
19:         print(ints);
20:         ArrayList<Double> doubles = new ArrayList<Double>();
21:         doubles.add(34.5);
22:         doubles.add(23.5);
23:         print(doubles);
24:
25:         List<? extends Number> numbers1 = Arrays.asList(1, 2, 3);
26:         List<? extends Number> numbers2 = Arrays.asList(1.5, 2.5, 3.5);
27:         print(numbers1);
28:         print(numbers2);
29:
30:         ArrayList<?> list = new ArrayList<Number>(5);
31:         list.add(new Integer(1));
32:         list.add(new Long(2L));
33:         list.add(new Object());
34:         list.add(null);
35:
36:         Object obj = list.get(0);
37:         Number num = list.get(0);
38:         Integer integer = list.get(0);
39:     }
40: }
```

โปรแกรมตัวอย่างที่ 6: Wildcard

หลังจากที่เรา compile สิ่งที่เราได้ก็คือ error ในบรรทัดที่ 31, 32, 33, 37, และ 38 สาเหตุก็คือ เราไม่รู้ว่า list เก็บข้อมูลชนิดอะไรก่อนที่เราจะจองเนื้อที่ เรารู้ว่าเราได้ส่งข้อมูลที่เป็น sub-type ของอะไรที่เราไม่รู้ไปให้ method add() ซึ่งตัวมันเองต้องการข้อมูลชนิด E ที่เป็น sub-type ของ ArrayList ส่วนในบรรทัดที่ 34 นั้นเราทำได้เพราะ null เป็นทุกสิ่งทุกอย่างที่มีอยู่ในโลกของ Java (ทุกสิ่งเป็น Object) และในบรรทัดที่ 36 เราสามารถดึงข้อมูลออกได้ด้วย method get() เพราะว่าเราประกาศให้เป็น Object (เหตุผลเดียวกันกับการใช้ method add() ก่อนหน้านี้)

การใช้ wildcard กับ super

ใน class Collections เรามี method ตัวหนึ่งที่ทำหน้าที่ในการ copy ข้อมูลจาก list หนึ่งไปยังอีก list หนึ่งมีหน้าตาดังนี้

```
public static <T> void copy(List<? super T> dst, List<? extends T> src) {
    for(int i = 0; i < src.size(); i++)
        dst.set(i, src.get(i));
}
```

เครื่องหมายที่มีอยู่ใน parameter list ของ method copy() ที่ว่า "? super T" บ่งบอกถึงชนิดของข้อมูลปลายทางของการ copy ว่าเป็นชนิดอะไรก็ได้ที่เป็น super-type ของ T (ส่วนแหล่งที่มาของการ copy ก็เป็นข้อมูลชนิดอะไรก็ได้ที่เป็น sub-type ของ T)

โปรแกรม Wildcard3.java แสดงถึงการเรียกใช้ method copy() นี้

```
 7: class Wildcard3 {
 8:     //method to copy from one list to another
 9:     public static <T> void copy(List<? super T> dst, List<? extends T> src) {
10:         for(int i = 0; i < src.size(); i++)
11:             dst.set(i, src.get(i));
12:     }
13:
14:     public static void main(String[] args) {
15:         List<Object> objs = Arrays.<Object>asList("obj", 281, "another");
16:         List<Long> longs = Arrays.<Long>asList(69L, 938L);
17:         copy(objs, longs);
18:         System.out.println(objs);
19:     }
20: }
```

โปรแกรมตัวอย่างที่ 7: Wildcard

ผลลัพธ์ที่เราได้จากการ run คือ

[69, 938, another]

ในบรรทัดที่ 15 และ 16 ผู้อ่านจะสังเกตเห็นถึงการเรียกใช้ method asList() ที่มีการกำหนดชนิดของข้อมูลเข้าไปด้วยว่าจะให้เป็น <Object> หรือ <Long> (ซึ่งก่อนหน้านี้เราไม่มีการกำหนดดังที่เห็น) สาเหตุก็เพราะว่า Java ยอมให้เราเรียกทั้งสองแบบได้ แต่ถ้าเรารู้ว่าข้อมูลที่เราต้องการเก็บมีชนิดเป็นอะไรเราก็ควรที่จะบอกให้ Java รู้ได้เลย อย่างไรก็ตามเราไม่สามารถไม่กำหนดชนิดให้ได้ในบรรทัดที่ 15 ทั้งนี้ก็เพราะว่าเรามีข้อมูลที่ต่างชนิดกัน เราจึงคงต้องมีคำว่า <Object> อยู่ส่วนในบรรทัดที่ 16 นั้นเราไม่จำเป็นต้องมีคำว่า <Long> ก็ได้เพราะเรามีข้อมูลเป็น Long ทั้งสองตัว (ถ้าเราเปลี่ยนให้ข้อมูลในบรรทัดที่ 15 เป็น Object ทั้งหมดเราก็ไม่ต้องใส่คำว่า <Object> ก็ได้)

สรุป

การใช้ Generics ในการเขียนโปรแกรมนั้นทำให้โปรแกรมมีความยืดหยุ่นสูง ในเรื่องของการเลือกใช้ชนิดของข้อมูล ณ เวลาที่ผู้ใช้ต้องการ ถึงแม้ว่า Java จะมองข้ามองค์ประกอบนี้ไปเมื่อเริ่มแนะนำภาษานี้ใหม่ๆ แต่เมื่อนำมาใช้ก็ทำให้พวกเราหลายๆ คนมีช่องทางในการทำให้โปรแกรมเอื้อความสะดวกให้กับผู้ใช้มากขึ้น อย่างไรก็ตาม Generics ใน Java ที่จริงแล้วคือการเปลี่ยน (type erasure) ชนิดของข้อมูลให้เป็น Object โดยอัตโนมัติผ่านทาง compiler เช่นถ้าเรามี List<T> และเราเรียกใช้ List<Integer> compiler ก็จะทำการปรับเปลี่ยนชนิดของข้อมูลให้เป็น Integer ทุกที่ที่มีการเรียกใช้ และถ้าหากเราเรียก List<int> compiler ก็จะทำการ box ให้เรา ซึ่งเป็นค่า overhead ที่มากมายในมุมมองของการ implement ถ้าจะพูดแบบภาษาชาวบ้านแล้วกระบวนการนี้เป็นการ cast ข้อมูลให้กับเราโดยอัตโนมัติ (ผ่านทาง compiler) นั่นเอง

ถึงแม้ว่าบทความนี้ไม่ได้พูดถึงองค์ประกอบต่างๆ ที่เกี่ยวข้องกับ Generics ทั้งหมดอย่างละเอียด ประกอบกับเนื้อที่ที่จำกัด เราก็เพียงแต่หวังว่าผู้อ่านจะเล็งเห็นถึงความยืดหยุ่นที่ผู้ใช้สามารถเลือกได้ถ้าเราใช้ Generics ในการออกแบบโปรแกรม

เอกสารอ้างอิง

Naftalin, Maurice and Wadler, Philip. (2007). **Java Generics and Collections**. Beijing: O'Reilly.

McLaughli, Brett and Flanagan. (2004). **Java 1.5 Tiger A Developer's Notebook**. Beijing: O'Reilly.

Schildt, Herbert. (2007). **Java : The Complete Reference**. (7th ed.). New York: McGraw-Hill.

เนรมิต ชุมสาย ณ อยุธยา. (2006). เริ่มต้นด้วย Java. ค้นเมื่อ 5 พฤศจิกายน 2550, จาก:
http://sci.feu.ac.th/faa/Java.intro/introToJava.html

Christy, Joy. (2007). **Generics in Java 5.0**. Retrieved
November 6, 2007, from: http://www.javabeat.net/articles/java-5-0/2007/08/generics/

Eckel, Bruce. (2007). **3-10-04 Generics Aren't**. Retrieved November
6, 2007, from: http://www.mindview.net/WebLog/log-0050

Generics. (2007). Retrieved November 5, 2007, from: http://java.sun.com/j2se/1.5.0/docs/guide/language/generics.html

Hejlsberg, Anders. (2007). **Generics in C#, C++, and Java**. Retrieved November
6, 2007, from: http://www.artima.com/intv/generics2.html

Langer, Angelika. (2007). **Java Generic FAQ - Frequently Asked Questions**. Retrieved
November 5, 2007, from: http://www.angelikalanger.com/GenericsFAQ/JavaGenericsFAQ.html