Generic Classes and Methods

Compiled by Kanjana Eiamsaard, Version 1.0.1 2022-03-08

Agenda

- ความหมาย คุณสมบัติ และเหตุผลในการใช้ Generic class
- การลบล้างชนิดข้อมูล
- ชนิดข้อมูลดิบ
- เมทอดเจเนอริก
- Wildcard parameter

Generic class

- คือ คลาสที่ไม่เจาะจงชนิดข้อมูล
  - **ไม่ระบุชนิดตัวจัดเก็บข้อมูลโดยตรงในการสร้างคลาส**
  - **แต่จะรับพารามิเตอร์เพื่อนำไประบุชนิดข้อมูลในคลาสอีกทีหนึ่ง (type parameter)**
  - นำไปใช้กับข้อมูลชนิดใดก็ได้ที่เป็นตัวแปรอ้างอิง (Reference type) เช่น Integer, Double, Float, Character, array, enum, Object เป็นต้น
  - Primitive data type ไม่สามารถนำมาใช้ร่วมได้
- นำมาใช้เพื่อแก้ปัญหาคลาสที่ไม่มีความแตกต่างในแง่สาระสำคัญ แต่ต้องแยกคลาส เนื่องจากต้องรองรับข้อมูลต่างชนิดกัน
  - การแยกทำให้ระดับในการบำรุงรักษาต่ำ (Low maintainability)

ตัวอย่างคลาสที่ไม่มีความแตกต่างในแง่สาระสำคัญ

```java
public class PairInteger{
    private int x;
    private int y;
    public PairInteger(){…
    public PairInteger(int x, int y){
        this.x = x;
        this.y = y;
    }
    public int getX(){
        return this.x;
    }
    public int getY(){…
    public void swap(){
        int temp;
        temp = this.x;
        this.x = y;
        this.y = temp;
    }
    @Override
    public String toString(){
        return String.format(
            "X = %d and Y = %d",
            this.x,this.y);
    }
}
```

```java
public class PairDouble{
    private double x;
    private double y;
    public PairDouble(){…
    public PairDouble(Double x, Double y){
        this.x = x;
        this.y = y;
    }
    public double getX(){
        return this.x;
    }
    public double getY(){…
    public void swap(){
        double temp;
        temp = this.x;
        this.x = y;
        this.y = temp;
    }
    @Override
    public String toString(){
        return String.format(
            "X = %.2f and Y = %.2f",
            this.x,this.y);
    }
}
```

ตัวอย่างคลาสที่ไม่มีความแตกต่างในแง่สาระสำคัญ (ต่อ)

```
PairInteger d1 = new PairInteger(5,10);
System.out.println("Before swap: "+d1.toString());
d1.swap();
System.out.println("After swap: "+d1.toString());

PairDouble d2 = new PairDouble(5.0,10.0);
System.out.println("Before swap: "+d2.toString());
d2.swap();
System.out.println("After swap: "+d2.toString());
```

```
Before swap: X = 5 and Y = 10
After swap: X = 10 and Y = 5
Before swap: X = 5.00 and Y = 10.00
After swap: X = 10.00 and Y = 5.00
```

- คลาส PairInteger และคลาส PairDouble มีพฤติกรรมที่เหมือนกันมาก แต่ต้องสร้างคลาสแยกกัน

ตัวอย่างการแก้ปัญหาด้วยคลาสเจเนอริก

```java
public class Pair<T>{
    private T x;
    private T y;
    public Pair(){…
    public Pair(T x, T y){
        this.x = x;
        this.y = y;
    }
    public T getX(){
        return this.x;
    }
    public T getY(){…
    public void swap(){
        T temp;
        temp = this.x;
        this.x = y;
        this.y = temp;
    }
    @Override
    public String toString(){
        return String.format(
            "X = "+this.x+" Y = "+this.y);
    }
}
```

```java
Pair<Integer> d3 = new Pair<>(5, 10);
System.out.println("Before swap: "+d3.toString());
d3.swap();
System.out.println("After swap: "+d3.toString());

Pair<Double> d4 = new Pair<>(5.0, 10.0);
System.out.println("Before swap: "+d4.toString());
d4.swap();
System.out.println("After swap: "+d4.toString());
```

- Generic class ในที่นี้คือ คลาส Pair
  - **ไม่ระบุชนิดข้อมูล แต่รับชนิดข้อมูลผ่านพารามิเตอร์ที่ส่งมาจากผู้เรียก**
- ผู้เรียกส่งพารามิเตอร์เป็น Wrapper class ของ int คือ Integer (สืบทอดจากคลาส Object)
- ใช้ diamond operator (<>) ได้ทำให้ไม่ต้องระบุชนิดข้อมูลซ้ำอีกรอบ

ตัวอย่างการแก้ปัญหาด้วยคลาสเจเนอริก (ต่อ)

```
Pair<Integer> d3 = new Pair<>(5, 10);
```

1

```
public Pair(T first, T second) {
    this.first = first;
    this.second = second;
}
```

2

```
public Pair(Integer first, Integer second) {
    this.first = first;
    this.second = second;
}
```

ตัวอย่างการแก้ปัญหาด้วยคลาสเจเนอริกที่ไม่ถูกต้อง

```
Pair<int> d5 = new Pair<>(5, 10);
```

```
MainDriver.java:24: error: unexpected type
        Pair<int> d5 = new Pair<>(5, 10);
                 ^
  required: reference
  found:    int
1 error
```

- ไม่สามารถระบุพารามิเตอร์เป็น primitive type ได้

การคลาสเจเนอริกที่มีข้อมูลมากกว่าหนึ่งชนิด

```java
public class MultiPair <A,B>{
    private A x;
    private B y;
    public MultiPair(){
        this(null,null);
    }
    public MultiPair(A x, B y){
        this.x = x;
        this.y = y;
    }
    public A getX(){
        return this.x;
    }
    public B getY(){
        return this.y;
    }

    @Override
    public String toString(){
        return String.format(
            "X = "+this.x+" Y = "+this.y);
    }
}
```

```java
System.out.println("----Multi generic class----");
MultiPair<Integer,Double> d5 = new MultiPair<>(5, 10.0);
System.out.println("Before swap: "+d5.toString());
```

```
----Multi generic class----
Before swap: X = 5 Y = 10.0
```

- A และ B ถูกนำมาใช้เป็นตัวกำหนดชนิดข้อมูลที่แตกต่างกัน
- **ที่มี 2 ตัว ไม่ใช่เพราะต้องการรับค่าสองค่า แต่เป็นเพราะมีข้อมูลสองชนิด**
- Type parameter มักนิยมใช้เป็นตัวพิมพ์ใหญ่ เช่น T (จากคำว่า type), E (จากคำว่า element), K (จากคำว่า key) และ V (จากคำว่า value)

การลบล้างชนิดข้อมูล (Type erasure)

```java
public class Pair {
    private Object first;
    private Object second;

    public Pair(Object first, Object second) {
        this.first = first;
        this.second = second;
    }

    public Object getFirst() {
        return first;
    }

    public Object getSecond() {
        return second;
    }

    public void swap() {
        Object temp = first;
        first = second;
        second = temp;
    }

    @Override
    public String toString() {
        return "(" + first + ", " + second + ")";
    }
}
```

- เบื้องหลังคลาสเจเนอริกจะแทนที่พารามิเตอร์ระบุชนิดด้วยคลาส Object ทั้งหมด
  - ด้วยคุณสมบัติ Polymorphism
- เมื่อคอมไพล์โค้ดไปเป็น bytecode แล้ว ข้อมูลชนิดของคลาส Pair<Integer> จะไม่เหลืออีก ดังนั้น เมื่อโปรแกรมรัน เราจะไม่สามารถตรวจสอบย้อนหลังได้ว่า Pair ตัวนี้ถูกประกาศเป็น Pair<Integer> เรียกข้อจำกัดนี้ว่า “**type erasure**”

ชนิดข้อมูลดิบ (Raw type)

```
ArrayList scores = new ArrayList();

scores.add(new Double(28.5));
scores.add(new Integer(15));
scores.add("John");
scores.add(new Boolean(false));
System.out.printf("scores: %s%n", scores);

Double first = (Double)scores.get(0);
Integer second = (Integer)scores.get(1);
String third = (String)scores.get(2);
Boolean fourth = (Boolean)scores.get(3);
```

```
scores: [28.5, 15, John, false]
```

- Java 1.0 ยังไม่มีคุณสมบัติเจเนอริก มีแต่แบบที่รับข้อมูลชนิด Object จึงทำให้นำคลาสชนิดใดมาใช้ก็ได้
  - แต่ไม่สะดวก เพราะตอนเรียกใช้ต้องทำการ casting และเสี่ยงที่จะ cast ผิดชนิด
  - การใช้งานคลาสแบบนี้เรียก “Raw type”
- ArrayList ใน Java 1.0 มีความแตกต่างจากปัจจุบัน (ArrayList<>)
- Java รุ่นปัจจุบันยังคงอนุญาตให้ใช้แบบ raw type เพื่อให้โปรแกรมเก่าใช้งานได้ **แต่คนรุ่นใหม่ควรหลีกเลี่ยง**
- Autoboxing ทำให้ใช้คลาสแบบ raw type ได้สะดวกยิ่งขึ้น แต่ก็ยังเสี่ยงที่จะ casting ผิดอยู่ดี

เมทอดเจเนอริก

- กำหนดเมทอดให้เป็นเจเนอริกได้ โดยที่คลาสของเมทอดนั้นไม่จำเป็นต้องเป็นเจเนอริกด้วย
- ตัวอย่างนี้เป็นการทำ overloading method โดย method เหมือนกัน แต่พารามิเตอร์ที่มีชนิดต่างกัน

```
integerArray contains:
1 2 3 4 5 6
doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
characterArray contains:
H E L L O
```

```java
public class OverloadedMethods {
    public static void main(String[] args) {
        Integer[] integerArray = { 1, 2, 3, 4, 5, 6 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
        Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };

        System.out.println("integerArray contains: ");
        printArray(integerArray);
        System.out.println("doubleArray contains: ");
        printArray(doubleArray);
        System.out.println("characterArray contains: ");
        printArray(characterArray);
    }

    public static void printArray(Integer[] inputArray) {…

    public static void printArray(Double[] inputArray) {…

    public static void printArray(Character[] inputArray) {
        for (Character element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }
}
```

เมทอดเจเนอริก (ต่อ)

```
public static <T> void printArray(T[] inputArray)
```

```java
public class GenericMethod {
    public static void main(String[] args) {
        Integer[] integerArray = { 1, 2, 3, 4, 5, 6 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
        Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };

        System.out.println("integerArray contains: ");
        printArray(integerArray);
        System.out.println("doubleArray contains: ");
        printArray(doubleArray);
        System.out.println("characterArray contains: ");
        printArray(characterArray);
    }

    public static <T> void printArray(T[] inputArray) {
        for (T element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }
}
```

- รูปแบบการประกาศเมทอดเจเนอริก คือ
  - ประกาศพารามิเตอร์ระบุชนิด (type parameter) อยู่ในเครื่องหมาย <> โดยอยู่หน้า return type

```
integerArray contains:
1 2 3 4 5 6
doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
characterArray contains:
H E L L O
```

เมทอดเจเนอริก (ต่อ)

```
15      public static <T> void printArray(T[] inputArray) {
16          for (T element : inputArray) {
17              System.out.printf("%s ", element);
18          }
19          System.out.println();
20      }
21  }
```

```
public static void printArray(Object[] inputArray) {
    for (Object element : inputArray) {
        System.out.printf("%s ", element);
    }
    System.out.println();
}
```

- เบื้องหลังของเมทอดเจเนอริก ในตัวอย่าง printArray() เป็นเมทอดเดียวกันคือ ข้อมูลชนิด Object

เมทอดเจเนอริก (ต่อ)

```
7           System.out.println("integerArray contains: ");
8           printArray(integerArray);
9           System.out.println("doubleArray contains: ");
10          printArray(doubleArray);
11          System.out.println("characterArray contains: ");
12          printArray(characterArray);
13      }
14
15      public static <T> void printArray(T[] inputArray) {
16          for (T element : inputArray) {
17              System.out.printf("%s ", element);
18          }
19          System.out.println();
20      }
21  }
```

≠

```
System.out.println("----Multi generic class----");
MultiPair<Integer,Double> d5 = new MultiPair<>(5, 10.0);
System.out.println("Before swap: "+d5.toString());
```

- **ข้อแตกต่าง**ระหว่างเมทอดเจเนอริกกับคลาสเจอเนอริก คือ เมทอดเจอเนอริกไม่ต้องระบุชนิดข้อมูลเวลาเรียกใช้
- คอมไพเลอร์จะดูเองจากชนิดของอาร์กิวเมนต์ที่ผ่านให้กับเมทอด

ปัญหาในการใช้เมทอดเจเนอริก

```java
public class MaximumTest {
    public static void main(String[] args) {
        System.out.printf("Maximum of %d, %d, and %d is %d%n",
                          3, 4, 5,
                          maximum(3, 4, 5));
        System.out.printf("Maximum of %.1f, %.1f, and %.1f is %.1f%n",
                          6.6, 8.8, 7.7,
                          maximum(6.6, 8.8, 7.7));
        System.out.printf("Maximum of %s, %s, and %s is %s%n",
                          "pear", "apple", "orange",
                          maximum("pear", "apple", "orange"));

        System.out.println(MaximumTest.maximum(1, 2, 3).getClass());
    }

    public static <T extends Comparable<T>> T maximum(T x, T y, T z) {
        T max = x;
        if (y.compareTo(max) > 0)
            max = y;
        if (z.compareTo(max) > 0)
            max = z;
        return max;
    }
}
```

- ในตัวอย่าง มีการรับค่ามา 3 ตัว ซึ่งไม่ระบุชนิด และคืนกลับเป็นชนิดที่รับมา
  - ในช่วงที่เมทอดรับค่า Java มีคุณสมบัติ autoboxing ทำให้เกิดการ Casting ชนิดข้อมูลที่เหมาะสมที่สุด (ในที่นี้แปลงจาก primitive -> wrapper class)
  - ชนิดของคลาสที่คืนกลับคือ Wrapper class
  - Wrapper class ทุกชนิดสร้างขึ้นจากการ implement interface Comparable<T> เอาไว้แล้ว
  - **T ในที่นี้ จึงไม่ใช่คลาสใดก็ได้ แต่ต้องเป็นคลาสที่ implement interface Comparable<T> เท่านั้น**
  - **หากไม่กำหนดชนิดของค่าคืนกลับ Java จะกำหนดคลาสปริยายเป็น Object ซึ่งไม่สามารถใช้เมทอด compareTo() ได้ (ในที่นี้คือ <T extends Comparable<T>>)**
- การเปรียบเทียบค่าไม่ใช้ Relational Operators เพราะเป็นการเปรียบเทียบ ระหว่าง Object (จากคลาสเดียวกัน)
  - นำ compareTo() มาใช้เพื่อการเปรียบเทียบ ซึ่งเป็นความสามารถของคลาสที่อิมพลีเมนต์อินเทอร์เฟซ Comparable<T>

การใช้ wildcard ในพารามิเตอร์ของเมทอด

```java
public class WildcardTest {
    public static void main(String[] args) {
        // Integer-only list
        ArrayList<Integer> integerList = new ArrayList<>();
        Collections.addAll(integerList, 1, 2, 3, 4, 5);

        System.out.printf("integerList contains: %s%n", integerList);
        System.out.printf("Total of the elements in integerList: %.1f%n", sum(integerList));

        // Double-only list
        ArrayList<Double> doubleList = new ArrayList<>();
        Collections.addAll(doubleList, 1.1, 3.3, 5.5);

        System.out.printf("doubleList contains: %s%n", doubleList);
        System.out.printf("Total of the elements in doubleList: %.1f%n", sum(doubleList));

        // Mixed-type list
        ArrayList<Number> numberList = new ArrayList<>();
        Collections.addAll(numberList, 1, 2.4, 3, 4.1);

        System.out.printf("numberList contains: %s%n", numberList);
        System.out.printf("Total of the elements in numberList: %.1f%n", sum(numberList));
    }

    public static double sum(ArrayList<? extends Number> list) {
        double total = 0;

        for (Number element : list)
            total += element.doubleValue();

        return total;
    }
}
```

- เมทอด sum() มีการใช้ wildcard เพื่อให้สามารถรับพารามิเตอร์ได้หลายชนิด แต่มีข้อแม้ว่าต้องเป็นชนิดที่เป็นซับคลาสของ Number และชนิด Number เอง
  - ArrayList<Integer>, ArrayList<Double> และ ArrayList<Number>
- **การสร้าง wildcard ให้พารามิเตอร์เปรียบเทียบได้กับการสร้างเมทอดเจเนอริก**

เปรียบเทียบ wildcard parameter กับ Generic method

```
public static double sum(ArrayList<? extends Number> list)
```

=

```
public static <T extends Number> double sum(ArrayList<T> list)
```

อ้างอิง

- https://nbviewer.jupyter.org/github/Poonna/java-book/