Big Java Chapter 4

4.1 Number Types

- ทุกๆค่า(value)ในjava จะต้องเป็น obj. หรือไม่ก็จะต้องเป็น 1ใน8 ***primitive type***

Table 1 Primitive Types

Type	Description	Size
`int`	The integer type, with range –2,147,483,648 (`Integer.MIN_VALUE`) . . . 2,147,483,647 (`Integer.MAX_VALUE`, about 2.14 billion)	4 bytes
`byte`	The type describing a single byte, with range –128 . . . 127	1 byte
`short`	The short integer type, with range –32,768 . . . 32,767	2 bytes
`long`	The long integer type, with range –9,223,372,036,854,775,808 . . . 9,223,372,036,854,775,807	8 bytes
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme (see Special Topic 4.5 on page 153)	2 bytes
`boolean`	The type with the two truth values `false` and `true` (see Chapter 5)	1 bit

- แต่ล่ะประเภทจะมีขนาด(range) ไม่เท่ากัน ตามตารางข้างบน ซึ่งถ้าหากอยากรู้ค่าสูงสุดของ int หาได้จาก **Integer.MAX_VALUE** และค่าต่ำสุดหาได้จาก **Integer.MIN_VALUE**

- int เก็บได้ประมาน $2*10^9$ ถ้าเกิดเอาเลขสัก 10^{12} ไปเก็บไว้ใน int ค่าที่เก็บไว้จะกลายเป็น -727379968 ซึ่งมันไม่ใช่ตัวที่เราอยากจะเก็บไง ปัญหาแบบนี้เรียกว่า ***overflow***

- ปัญหาใหญ่กว่าคือ overflow จะไม่มี error แจ้งเตือนใดๆทั้งสิ้น ต้องรับรู้ได้ด้วยตัวเอง

- การใช้ floting-pont บางทีก็จะมีการแสดงผลที่ผล เช่น

```
f = 4.35
sysout(f*100)   >>      434.99999999
```

ปัญหาแบบนี้เรียกว่า ***rounding error***

- ปัญหานี้เพราะว่า computer ใช้เลขฐาน2ในการเก็บ ไม่ได้ใช้ฐาน10แบบเรา (รายละเอียดเคยเรียนไปใน intro แล้วเนอะ)

- และเพราะปัญหานี้ professional programs เลยไม่นิยมใช้ double และ float สำหรับการคำนวณเกี่ยวกับการเงิน

- ปัญหา overflow และ rounding error นั้นจะหมดไป เพียงคุณใช้ BigInteger และ BigDecimal

```
BigInteger n = new BigInteger(“10000000000”);
BigInteger r = n.multiply(n);
BigDecimal d = new BigDecimal(“4.35”)
```

แต่ว่า 2คลาสนี้จะใช้ + - * ไม่ได้ ต้องใช้เมธอด add, subtract และ multiply แทน

4.2 Constants

- ค่าที่ไม่สามารถเปลี่ยนแปลงได้ตลอดโปรแกรม เรียกว่า ***constants*** (เรียกว่าค่าคงที่ก็จบล่ะ)

ex. area = 3.14 * r * r

- จากตัวอย่างจะเห็นได้ว่าเป็นโค้ดที่มีการพิมพ์จำนวนเลขลงไปเอง ซึ่งเราเป็นคนพิมพ์เองก็คงไม่งงไร แต่ถ้าเป็นคนอื่นมาอ่านละแบบ ไม่รู้ว่า 3.14 แมร่งคือเลขไรว่ะ เสกมาจากไหน

- ดังนั้นควรเก็บค่าพวกนี้ไว้ในตัวแปร อย่างน้อยคนอ่านก็จะรู้ว่ามันคือค่าอะไร

ex. double pi = 3.14;

area = pi * r * r;

- แบบนี้คนอ่านก็จะรู้ล่ะว่ามันคือค่า π

- แต่มีอีก สมมติเราเห็นล่ะว่าตัวแปรตัวนี้ ไม่ว่าโปรแกรมจะทำงานรอบไหน ไม่ว่า r จะเป็นเท่าไหร่ มันก็จะยังค่าเท่าเดิม ถ้างั้นก็ประกาศเป็นค่าคงที่(constants) ไปเลยดิ่ จะได้ไม่เผลอไปแก้ค่าไรมัน

Syntax : Declare Constants

```
final dataTyp varName = value;
final dataType varName;
```

ex. final double pi = 3.14;

- มีต่ออีก การตั้งชื่อ constants ควรตั้งเป็นตัวใหญ่หมด และใช้ snakecase แทน camelcase

ex. final double PI = 3.14;

final double FEE_VALURE = 0.07;

- แถมๆ ตัวแปรที่ประกาศ final แล้ว จะใช้ private แทน public ก็ได้นะ เพราะยังไงก้แก้ไขค่าไม่ได้อยู่ล่ะ

ex. register.enterPayment(1, 2, 1);

- จากตัวอย่าง สมมติเราจะใช้เมธอดแบบนี้ ละก็ไม่รู้ว่าแต่ล่ะตัวคืออะไร? แล้วเมธอดนี้ทำไร? รีเทิร์นค่าไรอีกล่ะ? เพราะงั้นเราควรเขียนอธิบายเมธอดด้วย

Syntax

```
/**
        Method detail
        @param param1 detail
        @param param2 detail
        @return detail
*/
```

ex. /**

Enters the payment received from the customer.

@param dollars the number of dollars in the payment

@param quarters the number of quarters in the payment

@param dimes the number of dimes in the payment

@return the total number in the payment

*/

public int enterPayment(int dollars, int quarters, int dimes){

...

}

4.3 Arithmetic Operations and Mathematical Functions

4.3.1 Arithmetic Operation

- ไม่มีไรจะเขียน แค่บอกว่าใช้ * แทนคูณ / แทนหาร

4.3.2 Increment and Decrement

operator	code test	result
post increase	`System.out.println(i++);` `System.out.println(i);`	1 2
pre increase	`System.out.println(++i);` `System.out.println(i);`	2 2
post decrease	`System.out.println(i--);` `System.out.println(i);`	1 0
pre decrease	`System.out.println(--i);` `System.out.println(i);`	0 0

4.3.3 Integer Division

- ไม่ว่าจะ int/float, float/int, float/float จะได้ออกมาเป็น float หมด
- มีเฉพาะ int/int ถึงจะได้ออกมาเป็น int และการปัดเศษทิ้ง (เหมือน // ใน python)

4.3.4 Powers and Roots

- ใช้ Math.pow(x, n) กับ Math.sqrt(x)
- เมธอดอื่นๆในคลาส Math เช่น นส.หน้า 141 ไปเปิดเอง

4.3.5 Casting and Rounding

- cast รอสรุปละเอียดทีเดียวละกัน
- เวลาจะปัดเศษ 3.5 เป็น 4 งี้ ให้ใช้ Math.round(x) และจะรีเทิร์นออกมาเป็น long