Methods (หรือที่เรียกกันทั่ว ๆ ไปว่า Function)

วัตถุประสงค์

- เข้าใจว่าทำไมต้องมีการใช้ method
- เข้าใจถึง concept ของ modularity
- สามารถนิยาม method, เรียก method และส่งค่า argument ให้ method ได้
- เข้าใจถึง scope ของตัวแปร
- เข้าใจเรื่อง stack กับ method

ทำไมต้องมีการใช้ method

ปกติโปรแกรมจะทำงานตามลำดับคำสั่งที่เขียนในโปรแกรม โดยโปรแกรมทุกโปรแกรมจะเริ่มการทำงานที่ method หลัก (method main) ก่อนเสมอ แต่โปรแกรมเราใหญ่มาก ๆ การที่ code ทุกบรรทัดอยู่ใน method main จะไม่มีประสิทธิภาพ (เพราะ ทำให้เข้าใจยาก แก้ไขยาก ไม่สามารถนำ code มาใช้งานใหม่ได้)

method เป็นกลุ่มของ statement ย่อย ๆ ที่ถูกจัดมาอยู่รวมกันเพื่อทำงานหนึ่งให้สำเร็จ ผู้เขียนโปรแกรมรู้ว่า การทำงานนั้นจะถูกใช้บ่อย ๆ ทำให้ผู้พัฒนาโปรแกรมต้องการเขียน code ขึ้นมาแค่ครั้งเดียว แล้วนำ method มาเรียกใช้ใหม่ได้อีกเรื่อย ๆ โดยไม่ต้องเขียนรายละเอียดของ code ขึ้นมาใหม่ ยกตัวอย่างเช่น ถ้าเราต้องการหาค่า m ยกกำลัง n (m^n) ถ้าเราประกาศ method สำหรับ code ด้านล่างให้ถูกต้อง เราก็ไม่จำเป็นต้องเขียน code 4 บรรทัดด้านล่างทุก ๆ ครั้งที่เราอยากจะหาค่ายกกำลัง แต่เราเรียกใช้ method ได้เลย

```
int result = 1;
for(int count = 0; count < n; count++) {
    result = result * m;
}
```

หลักการของ Modularity

เมื่อโปรแกรมมีขนาดใหญ่ขึ้น เรามีความจำเป็นต้องแบ่งโปรแกรมออกเป็นส่วนย่อย (เรียกว่าโปรแกรมย่อย) เพื่อให้สามารถทำความเข้าใจได้ง่าย, พัฒนาได้ง่าย, และสามารถนำโปรแกรมย่อยที่พัฒนาแล้วไปใช้ในโปรแกรมอื่นต่อไปได้ โปรแกรม Java ประกอบด้วย

- method หลัก (main method) หนึ่ง โปรแกรมจะมีเพียง method หลัก method เดียวเท่านั้น และกำหนดให้ชื่อเป็น main ได้เท่านั้น ใน method main นี้จะบ่งบอกว่าโปรแกรมทั้งโปรแกรม "ทำอะไร" และ มีลำดับการทำงานอย่างไร
- method ย่อย มีจำนวนไม่จำกัด แต่ละmethod มีหน้าที่ในการทำงานเพียงอย่างเดียว และ มีรายละเอียดในการทำงานว่า "ทำอย่างไร"
- "โปรแกรมแต่ละโปรแกรมทำงานเพียงอย่างเดียว และทำงานนั้นให้ดีที่สุด"

ประโยชน์ของการแบ่งงานออกเป็น method หลัก-method ย่อย (Modularity)

- เขียนครั้งเดียว ใช้งานได้หลายครั้ง (Reusability)
- เป็นการรวมคำสั่งไว้ที่จุดเดียว
- แก้ไขง่าย เพราะแก้ไขเพียงจุดเดียว (Locality)
- Method ย่อยที่เขียนขึ้นสามารถนำไปใช้งานในโปรแกรมอื่นได้ (Generality)

หมายเหตุ: method ย่อย (subprogram) ในภาษารุ่นเก่าเช่นภาษา Fortran นิยมเรียกว่า subroutine สำหรับภาษาในกลุ่ม object-oriented เช่น C++ และ Java นิยมเรียกว่า method

การนิยาม method ผู้เขียนโปรแกรมต้องนิยามส่วนประกอบดังนี้

```
access_modifier  return_value_type    methodName (parameter_list) {
        // body
}
```

1. ส่วนหัวของ method ประกอบไปด้วย
    a. คำขยายเกี่ยวกับการเข้าถึง (access modifier) เช่น public static (รายละเอียดจะกล่าวภายหลัง)

b. ชนิดของค่า return (return value type) เช่น int, double, boolean

c. ชื่อของ method

d. input ของ method (argument หรือบางครั้งก็เรียกว่า parameter) โดยจำนวน input จะเป็นกี่ตัวก็ได้ แต่ละตัวจะถูกคั่นด้วยเครื่องหมาย comma

2. ส่วนของตัวของ method (method body) ประกอบไปด้วยกลุ่มของ statement ใน block { .... } ถ้า method มีการคืนค่า (การ return ค่า) ผู้เขียนโปรแกรมต้องใช้คำสั่ง return เพื่อคืนค่าให้ถูกต้อง โดยชนิดของค่าที่ return ควรจะตรงกับ return value type ที่ประกาศไว้ที่ส่วนหัวของ method

code ด้านล่างแสดงตัวอย่างของการประกาศ method ของ code สำหรับการยกกำลัง

```
1. public static int power(int m, int n) {
2.     int result = 1;
3.     for(int count = 0; count < n; count++) {
4.         result = result * m;
5.     }
6.     return result;
7. }
```

บรรทัดที่ 1: ส่วนหัวของ method มีส่วนประกอบดังนี้

a. public และ static เป็นคำขยาย

b. int บ่งบอกว่า method นี้จะ return ค่าให้ “ผู้เรียก” โดยชนิดของค่าที่ถูก return เป็น int

c. method นี้ชื่อ power

d. method นี้รับ input 2 ตัว ชื่อ m และ n และชนิดของ input ทั้งสองเป็น int

**** ชนิดของ input ไม่จำเป็นต้องเหมือนกับชนิดของค่าที่ถูก return

**** ชนิดของ input ทุกตัวไม่จำเป็นต้องเหมือนกัน

บรรทัดที่ 2-6: ส่วนตัวของ method หรือ กลุ่มของ statements ที่รวมเข้าไว้ด้วยกันเพื่อหาค่า m ยกกำลัง n ข้อสังเกตุเกี่ยวกับ คำสั่ง return ในบรรทัดที่ 6 มีดังนี้

- ถ้า method มีการ return ค่า ผู้เขียนต้องใส่คำสั่ง return ในตัวของ method ทุกครั้ง

- ชนิดของค่าที่จะ return ในบรรทัดที่ 6 ต้องสอดคล้องกับค่าที่ประกาศไว้ในส่วนหัวของ method (ถ้าเราเปลี่ยนบรรทัดที่ 2 เป็น double result = 1.0 เราจะพบ compile error)
- ในกรณีที่ method ไม่ได้ return ค่าใด ให้ใส่คำว่า void หลังคำขยายในส่วนหัวของ method

บรรทัดที่ 7: วงเล็บปีกกาปิด บ่งบอกว่าส่วนตัวของ method สิ้นสุดที่บรรทัดนี้ ซึ่งบอกถึงขอบเขต (scope) ของตัวแปรที่ประกาศใน method นี้ทั้งสามตัว ซึ่งได้แก่ m, n, และ result

**** ในตัวอย่างข้างต้น **scope** ของตัวแปร m n และ result คือบรรทัดที่ 1-7 หมายความว่า การเข้าถึงตัวแปรทั้งสามไม่ว่าจะเป็นการอ่านค่าจากตัวแปรหรือการเขียนค่าเข้าไปในตัวแปรนั้นทำได้แคใน scope นี้

**** ส่วนตัวแปรชื่อ count มี scope ตั้งแต่บรรทัดที่ 3 – 5 (ระหว่างวงเล็บปีกกาเปิดที่บรรทัดที่ 3 และปีกกาปิดที่บรรทัดที่ 5) method power จะใช้ตัวแปร count ที่บรรทัดอื่นที่ไม่ใช่บรรทัดที่ 3 4 และ 5 ไม่ได้ (ถ้าใช้ จะได้ error: count cannot be resolved)

การเรียกใช้ method และส่งค่า argument ให้ method

หลังจากนิยามหรือประกาศ method แล้ว method หลัก (หรือ method อื่น ๆ) สามารถเรียกใช้ method ที่ถูกนิยามได้โดยการใช้ชื่อของมันแล้วมีวงเล็บ ( ) ตามหลัง

<u>ในกรณีที่ method ต้องการ input</u> ผู้เขียนโปรแกรมต้องส่งค่าให้ตัวแปร input ทุกตัวที่ประกาศไว้ที่ส่วนหัวของ method โดยผู้เขียนจะต้องใส่ค่าที่จะส่งไว้ใน ( ) ที่อยู่หลังชื่อของ method

นอกจากนี้ชนิดของค่าที่ส่งจะต้องสอดคล้องกับชนิดของ input ของ method ที่นิยามไว้ที่ส่วนหัวของ method ด้วย

method main อาจเรียก method power โดยใช้ statement ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 1: power(2, 3);

ตัวอย่างที่ 2:
```
int a = 2, b = 3;
power(a, b);
```

**** ค่าจาก a และ b จะถูกคัดลอก (ส่ง) ไปให้ตัวแปร m และ n เพื่อให้ method power นำไปใช้ในการคำนวณต่อไป (กระบวนการคัดลอกค่านี้ เรียก Pass-by-value, ค่าของตัวแปร m จะเท่ากับค่าของตัวแปร a และ ค่าของตัวแปร n จะเท่ากับค่าของตัวแปร b แต่ a ไม่ใช่ตัวแปรตัวเดียวกับ m และ b ไม่ใช่ตัวแปรเดียวกับ n)

ในกรณีที่ method มีการ return ค่า (ในส่วนหัวของการนิยาม method ไม่ได้ใช้ void หลัง modifier) ไม่มี error เกิดขึ้นถ้าเราเรียก method power ตามแบบตัวอย่างที่ 1 และ 2 ข้างต้น แต่ method ที่เรียก method power นี้ ไม่ได้นำค่าที่ method return ให้มาใช้ ถ้าผู้เขียนต้องการนำค่าดังกล่าวมาใช้ต่อ ผู้เขียนต้องนำตัวแปรมารับค่า return นั้น ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 3:
```
int a = 2, b = 3, c;
c = power(a, b);
```

ค่า 2 และ 3 ของ a และ b จะถูกคัดลอกไปใส่ไว้ในตัวแปร m และ n จากนั้น method power จะทำการประมวลผลเพื่อให้ได้ค่า m ยกกำลัง n และ return ค่านั้นออกมา เนื่องจาก มีเครื่องหมาย = หน้า method power ค่าที่ถูก return ออกมาจาก method power จะถูกอ่าน และ นำไปเขียนในตัวแปร c (เครื่องหมาย = เป็น operator ที่นำค่าทางขวามือของเครื่องหมาย ไปใส่ไว้ในตัวแปรด้านซ้ายมือ)

ขอบเขตของตัวแปร (Variable's Scope)

- Local variables (ตัวแปรท้องถิ่น) คือ ตัวแปรที่ถูกประกาศเพื่อใช้งานภายใน method ใด method หนึ่งเท่านั้น ตัวแปรท้องถิ่นจะต้องถูกประกาศก่อนที่จะถูกใช้
- Variable's Scope (ขอบเขตของตัวแปร) คือ ส่วนของโปรแกรมที่สามารถเข้าถึงหรืออ้างถึงตัวแปรได้ ขอบเขตของตัวแปรจะเริ่มจากจุดที่มันถูกประกาศต่อไปจนถึงสิ้นสุด block ที่ตัวแปรนั้นอยู่ (เราใช้ { } บอกจุดเริ่มต้นและจุดสิ้นสุดของ block)
- เราไม่สามารถประกาศชื่อตัวแปรซ้ำกันใน block เดียวกันได้ เพราะ compiler จะสับสนไม่รู้ว่าตัวไหนเป็นตัวไหน แต่เราสามารถประกาศชื่อตัวแปรเหมือนกันกี่ตัวก็ได้ถ้าตัวแปรเหล่านั้นอยู่คนละ block กัน ตัวอย่างเช่น code ด้านล่าง ใน method 1 ตัวแปร i สามารถถูกประกาศสองครั้งได้ เพราะ อยู่คนละ block แต่ใน method 2 มี error เป็นเพราะ เราได้ประกาศ i ที่บรรทัดที่ 2 แล้ว ซึ่งมี scope

ตั้งแต่บรรทัดที่ 1 – 6 ดังนั้นการประกาศ i อีกครั้งในบรรทัดที่ 3 เป็นการประกาศซ้ำซ้อนใน block เดียวกันทำให้เกิด error ขึ้น

```
1.  public static void method1() {
2.      int x = 1;
3.      int y = 1;
4.      for(int i = 1; i < 10; i++) {
5.              x += I;
6.      }
7.      for(int i = 1; I < 10; i++) {
8.              y += I;
9.      }
10. }
```

```
1.  public static void method2() {
2.      int i = 1, sum = 0;
3.      for(int i = 1; i < 10; i++) {
4.              suj += I;
5.      }
6. }
```

- ในกรณีที่มี nested block (block ซ้อนกัน) scope ของตัวแปรที่ถูกประกาศใน block ภายนอก จะรวมถึง block ที่อยู่ภายในด้วย แต่ scope ของตัวแปรที่ถูกประกาศอยู่ใน block ภายใน จะไม่รวมถึง block ภายนอก ดังตัวอย่างต่อไปนี้

```
1.  for(int i = 0; i < 10; i++) {
2.
3.      // only i  can be referenced here
4.
5.      for( int j = 0; j < 10; j++) {
6.          // i and j can be referenced here
7.      }
8.
9.      // only i  can be referenced here
10. }
```

ตัวแปร i ถูกประกาศในบรรทัดที่ 1 มี scope ตั้งแต่บรรทัดที่ 1 – 10 ส่วน ตัวแปร j ถูกประกาศในบรรทัดที่ 5 มี scope ตั้งแต่บรรทัดที่ 5 – 7 ข้อสังเกต: ตัวแปร i สามารถถูกอ้างอิงได้ในบรรทัดที่ 5 -7 แม้ว่าจะเป็น block ที่อยู่ภายใน แต่ตัวแปร j จะไม่สามารถถูกอ้างอิงนอกบรรทัดที่ 5 – 7 ได้

Method Call กับ Stack (การเรียก method กับ โครงสร้างกองซ้อน)

เมื่อมีการเรียก method (invoke หรือ call method) โปรแกรมจะย้ายการทำงานไปที่ method ที่ถูกเรียกจนเสร็จแล้วจึงกลับมาทำงานต่อจากคำสั่งเดิมก่อนจะมีการเรียก method นี้

method ที่กำลังทำงานจะได้รับการควบคุมไป ซึ่งหากมีการเรียกใช้ method อื่น ๆ การควบคุมจะถูกโอนไปยังmethod ที่ถูกเรียกนั้น เมื่อทำเสร็จแล้วอำนาจการควบคุมก็จะกลับมายัง method ที่เรียก กลไกดังกล่าวถูกทำให้สำเร็จได้โดยใช้โครงสร้างชนิดกองซ้อน หรือ stack

พิจารณา code ต่อไปนี้:-

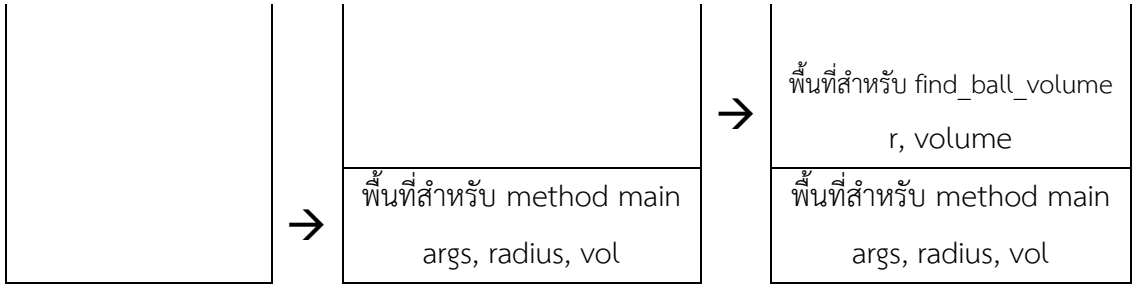
```
1.  public static int power(int m, int n) {
2.      int result = 1;
3.      for(int count = 0; count < n; count++) {
4.          result = result * m;
5.      }
6.      return result;
7.  }
8.
9.  public static double find_ball_volume(int r) {
10.     double volume = 0.0;
11.     volume = 4.0 / 3.0 * power(r, 3);
12.      return volume;
13. }
14.
15. public static void main(String [] args) {
16.      int radius = 10;
17.      double vol = find_ball_volume(radius);
18. }
```

บรรทัดที่ 1-7 ของโปรแกรมข้างต้น แสดงรายละเอียดของ method ชื่อ power method นี้ return ค่าจำนวนเต็มชนิด int และ มีตัวแปรที่เกี่ยวข้อง 4 ตัว (ตัวแปร m n และ result มี scope ตั้งแต่บรรทัดที่ 1-7 ส่วนตัวแปร count มี scope ตั้งแต่บรรทัดที่ 3-5

บรรทัดที่ 9-13 ของโปรแกรมข้างต้น แสดงรายละเอียดของ method ชื่อ find_ball_volume method นี้ return ค่าทศนิยมชนิด double และ มีตัวแปรที่เกี่ยวข้อง 2 ตัว (r และ volume มี scope ตั้งแต่บรรทัดที่ 9-13)

บรรทัดที่ 15-19 ของโปรแกรมข้างต้น แสดงรายละเอียดของ method หลัก ซึ่งไม่ return ค่าใด ๆ และ method นี้มีตัวแปร ที่เกี่ยวข้อง 3 ตัว (ตัวแปร args radius และ vol มี scope ตั้งแต่บรรทัดที่ 15-18)

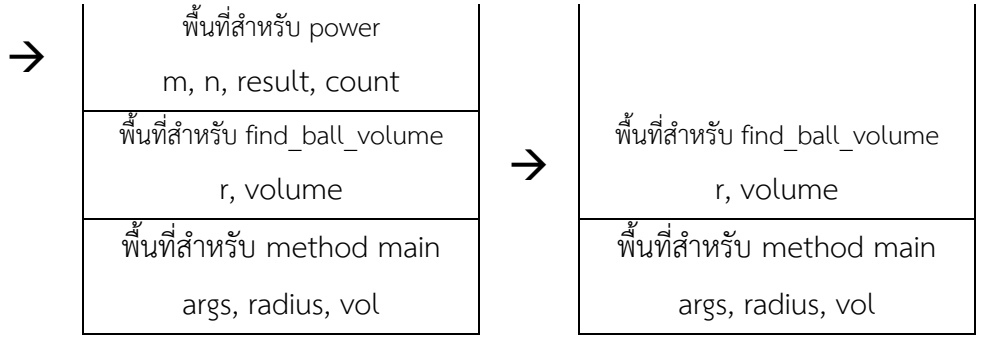
หลังจากที่เราเข้าใจ method และ scope ของตัวแปรใน code ข้างต้นแล้ว เรามาดูว่าขณะที่เรารันโปรแกรม call stack จะเป็นอย่างไรเมื่อ method ต่าง ๆ ถูกเรียกไปอยู่บน stack และ ค่าของตัวแปรถูกส่งข้าม stack อย่างไร



- ก่อนที่เราจะ run program call stack จะไม่มีข้อมูลใด ๆ ดังแสดงในรูปที่ 1 ด้านบน
- ครั้งแรกที่เรารันโปรแกรม, <u>method main (ระหว่างบรรทัดที่ 15-18) จะถูกเรียกก่อนเสมอ</u> ข้อมูลเกี่ยวกับ main (body ใน method main และ ตัวแปรที่เกี่ยวข้อง) จะถูกเก็บอยู่ใน call stack โดย CPU จะใช้เนื้อที่ใน stack ในรูปที่ 2 ในการดำเนินการต่าง ๆ
- ในบรรทัดที่ 17 เมื่อ main เรียก method find_ball_volume ระบบจะสร้างพื้นที่บน call stack เพิ่มขึ้นสำหรับ method find_ball_volume (ใช้เก็บข้อมูลที่เกี่ยวข้อง เช่น ตัวแปร r, volume ดังแสดงในรูปที่ 3) จากนั้น main จะทำการคัดลอกค่า 10 จากตัวแปร radius ที่อยู่ในพื้นที่ของ main ไปใส่ไว้ในตัวแปร r ในพื้นที่ของ stack สำหรับ find_ball_volume จากนั้น CPU จะไปทำงานใน method find_ball_volume (โดยใช้เนื้อที่

ของ stack สำหรับ find_ball_volume ในการดำเนินงาน) จากบรรทัดที่ 10 ไปจนถึงบรรทัดที่ 13 ให้เสร็จก่อน จึงจะกลับมาที่ บรรทัดที่ 17

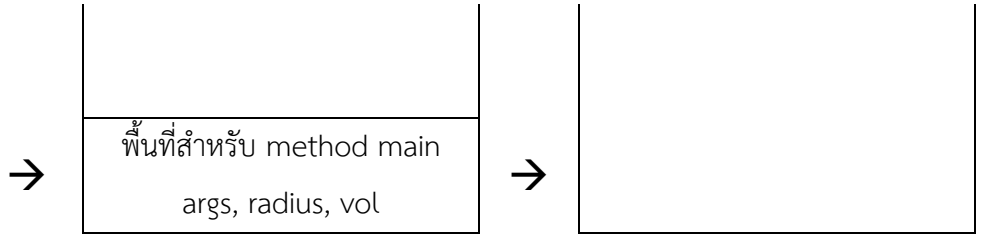
- ก่อนที่จะถึงบรรทัดที่ 17 เมื่อโปรแกรมทำงานถึงบรรทัดที่ 11 method find_ball_volume ได้เรียก method power ระบบจะสร้างพื้นที่บน call stack สำหรับ method power (ใช้เก็บข้อมูลที่เกี่ยวข้อง เช่น ตัวแปร m, n result, และ count ดังแสดงในรูปที่ 4) จากนั้น โปรแกรมจะทำการคัดลอกค่า 10 จากตัวแปร r ที่อยู่ในพื้นที่ของ find_ball_volume และ นำค่า 3 ไปใส่ไว้ในตัวแปร m และ n ในพื้นที่ของ stack สำหรับ method power จากนั้น CPU จะไปทำงานใน method power (โดยใช้เนื้อที่ของ stack สำหรับ power ในการดำเนินงาน) จากบรรทัดที่ 1 ไปจนถึงบรรทัดที่ 7 ให้เสร็จก่อน จึงจะกลับมาที่ บรรทัดที่ 11



4 5

- ในบรรทัดที่ 6 method power return ค่า 1000 ที่อยู่ใน ตัวแปร result ไปให้ method find_ball_volume เพื่อให้นำไปใช้ต่อ ระบบจะทำการคืนพื้นที่หน่วยความจำของ stack ของ method power แล้วกลับไปทำงานที่บรรทัด 11 โดยใช้พื้นที่ของ find_ball_volume ดังแสดงในรูปที่ 5

- โปรแกรมจะกลับมาทำงานที่บรรทัดที่ 11 โดยใช้ค่าที่ถูก return มาจาก method power ในการคำนวณค่า จากนั้นในบรรทัดที่ 12 find_ball_volume จะ return ค่าที่อยู่ในตัวแปร volume (ประมาณ 1333.33) ไปให้ main เพื่อให้ main นำไปใช้ต่อ ระบบจะทำการคืนพื้นที่หน่วยความจำของ stack ของ find_ball_volume แล้วกลับไปทำงานที่บรรทัด 17 โดยใช้พื้นที่ของ main ดังแสดงในรูปที่ 6 ในท้ายที่สุด main จะนำค่าที่ถูก return จาก find_ball_volume ไปใส่ไว้ในตัวแปร vol แล้วสิ้นสุดการทำงาน ระบบจะทำการคืนพื้นที่ของ stack ทั้งหมดดังรูปที่ 7



6 7