การทำซ้ำ (Iteration)

วัตถุประสงค์

- เข้าใจว่าทำไมต้องมีการใช้ loop
- สามารถใช้และเข้าใจการควบคุม loop while, for และ do...while
- เข้าใจถึงความคล้ายและความแตกต่างระหว่าง while, for และ do...while
- สามารถเขียน nested loop ได้
- สามารถใช้ break และ continue ใน loop ได้

ทำไมต้องมีการใช้ loop

คอมพิวเตอร์ต้องดำเนินการซ้ำ ๆ บ่อยครั้ง เช่น ผู้ใช้อาจสั่งให้คอมพิวเตอร์ print ข้อความเดิม ๆ ออกหน้าจอ 100 บรรทัด หรือ เวลาผู้ใช้เปิด file คอมพิวเตอร์ต้องอ่านข้อมูลจาก hard drive จนกว่าจะสิ้นสุด file หรือ คอมพิวเตอร์ต้องดำเนินการรอรับข้อมูลจากผู้ใช้จนกว่าจะปิดเครื่อง ดังนั้นการทำชุดคำสั่งเดิมซ้ำจึงถูกพบเห็นได้บ่อยมาก

Java มีคำสั่งให้คอมพิวเตอร์ดำเนินการซ้ำ (หรือ วน loop) ดังนี้ while, for, และ do...while

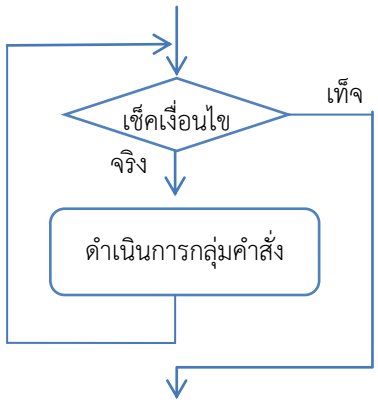
Statement While

เมื่อเราต้องการให้คอมพิวเตอร์ดำเนินการซ้ำ ๆ ในกรณีที่เงื่อนไขเป็นจริง เราอาจใช้ while ในการสร้าง loop โดยใช้ไวยากรณ์และ flow chart ดังนี้

while (เงื่อนไข) {

คำสั่งหรือกลุ่มคำสั่ง ;

}



แม้ในวงเล็บ ( ) ของ while จะเป็นเงื่อนไข เหมือนในวงเล็บ ( ) ของ if แต่เราจะใช้ if เมื่อเราต้องการเลือกทำกลุ่มคำสั่งใดคำสั่งหนึ่ง ในขณะที่เราจะใช้ while เมื่อเราต้องการสั่งให้ทำทุกคำสั่งใน { } ของ while ซ้ำ ๆ

Java จะดำเนินการคำสั่งหรือกลุ่มคำสั่งที่อยู่ถัดจาก while (เงื่อนไข) ก็ต่อเมื่อเงื่อนไขใน ( ) เป็นจริง และจะวนกลับมาทำซ้ำกลุ่มคำสั่งเดิมจนกว่าเงื่อนไขจะเป็นเท็จ ตัวอย่างเช่น

```
1:  int counter = 0;
2:  while ( counter < 100 ) {
3:        System.out.println("Welcome to Java!");
4:        counter++;
5:  }
```

โปรแกรมข้างต้นเป็นโปรแกรมที่ใช้พิมพ์ "Welcome to Java" 100 บรรทัดออกทางหน้าจอ โดยโปรแกรมจะวน loop 100 loop แต่ละ loop จะพิมพ์แค่หนึ่งบรรทัด

การวน loop ข้างต้นให้ครบ 100 loop เราจะต้องมี**ตัวแปรช่วยนับ**จำนวน loop (จำนวนการทำซ้ำ) ซึ่งในที่นี้ตัวแปรที่ช่วยนับชื่อ counter ที่ถูกประกาศและตั้งค่าเริ่มต้นให้เท่ากับ 0 ในบรรทัดที่ 1

บรรทัดที่สองเป็นการเช็คว่าค่าในตัวแปร counter น้อยกว่า 100 หรือไม่ ถ้าน้อยกว่าร้อย กลุ่มคำสั่งบรรทัดที่ 3 และ 4 จะถูกดำเนินการ นั่นคือ "Welocome to Java!" จะถูกพิมพ์ออกทางหน้าจอหนึ่งบรรทัด และ ค่าของ counter จะถูกเพิ่มขึ้นหนึ่ง

*** ให้สังเกต การตั้งค่าตั้งต้นให้ counter เท่ากับ 0 ในบรรทัดแรก การตั้งเงื่อนไขเช็ค counter น้อยกว่า 100 ในบรรทัดที่ 2 การเพิ่มค่า counter ทีละหนึ่งในบรรทัดที่ 4 จะทำให้ โปรแกรมรัน loop ข้างต้น 100 loop พอดี ซึ่งมีรายละเอียดดังนี้

- counter มีค่าเป็น 0 ที่จุดเริ่มต้นของ loop รอบที่ 1 และมีค่าเป็น 1 ที่ท้าย loop
- counter มีค่าเป็น 1 ที่จุดเริ่มต้นของ loop รอบที่ 2 และมีค่าเป็น 2 ที่ท้าย loop
- เป็นอย่างนี้เรื่อยไปจนถึง loop รอบที่ 100 ซึ่ง counter มีค่าเป็น 99 ที่จุดเริ่มต้นของ loop ที่ 100 และมีค่าเป็น 100 ที่ท้าย loop ซึ่ง loop รอบที่ 100 นี้จะเป็น loop สุดท้ายเนื่องจาก เวลาโปรแกรมกลับไป check เงื่อนไขอีกที และพบว่า เงื่อนไขเป็นเท็จ ทำให้ต้องออกจาก loop

*** ถ้าเรากำหนดค่า counter เท่ากับ 1 ในบรรทัดแรก loop และ เงื่อนไข counter < 100 ในบรรทัดที่ 2 โปรแกรมจะวนทำซ้ำ 99 รอบ แต่ถ้าเงื่อนไขในบรรทัดที่ 2 เป็น counter <= 100 โปรแกรมจะวนทำซ้ำ 100 รอบ (เปรียบเหมือนกรณีที่เรานับ 1 ถึง 99 กับ 1 ถึง 100)

บรรทัดที่ 5 เป็นการกำหนดขอบเขตสิ้นสุดของ block ของ statement while ซึ่งขอบเขตของ block เป็นเรื่องสำคัญ เพราะเราจะต้องกำหนดบอก Java ว่า คำสั่งใดอยู่ใน block (ให้ทำซ้ำ) คำสั่งใดที่อยู่นอก block ให้ทำครั้งเดียว

เงื่อนไขใน ( ) ของ while คือเงื่อนไขที่ให้ while ทำงานต่อไปเรื่อย ๆ คือ เงื่อนไขที่ระบุว่ายังทำงานไม่เสร็จ นิสิตต้องตั้งเงื่อนไขในนี้ให้ดี มิฉะนั้นโปรแกรมอาจทำงานติดลูป และทำงานไม่รู้จบ ซึ่งรูปแบบในการตั้งเงื่อนไขส่วนใหญ่จะประกอบไปด้วย 1) ตัวแปรที่มีค่าเปลี่ยนไปเรื่อย ๆ เมื่ออยู่ในลูป (เช่นค่าของตัวแปร counter ที่ถูกเพิ่มไปเรื่อย) ซึ่งจะถูกนำไปเปรียบเทียบกับ 2) ค่าเป้าหมาย โดยใช้ 3) ตัวดำเนินการเปรียบเทียบ หรือ comparison operator

ข้อควรระวัง 1 การใช้ while loop มีข้อควรระวังที่คล้ายกับของการใช้ if นั่นคือ การใช้ ; หลัง (เงื่อนไข) ทันทีจะทำให้โปรแกรมมีข้อผิดพลาด กล่าวคือ

```
int counter = 0;
while ( counter < 100 ) ; {
        System.out.println(“Welcome to Java!”);
        counter++;
}
```

จะให้ผลเหมือนกับผลของ code ข้างล่าง เป็นผลให้ โปรแกรมเข้าไปใน loop แล้วไม่สามารถออกจาก loop ได้เนื่องจากการดำเนินการ Null statement ใน block ของ while ไม่ทำให้ค่าของ counter เพิ่ม และเงื่อนไขจะไม่มีทางเป็นเท็จได้

```
int counter = 0;
while ( counter < 100 ) {
        ;       // <-- Null Statement
}
{
        System.out.println(“Welcome to Java!”);
        counter++;
}
```

ข้อควรระวัง 2 ถ้าเราต้องการให้ โปรแกรมวนทำซ้ำคำสั่งทั้งหมดในกลุ่มคำสั่ง เราต้องใส่คำสั่งทั้งหมดที่อยากให้ Java ดำเนินการซ้ำไว้ในเครื่องหมาย { } ถัดจาก (เงื่อนไข) ของ while นั่นคือ

```
int counter = 0;
while ( counter < 100 )
        System.out.println(“Welcome to Java!”);
        counter++;
```

จะให้ผลเหมือนกับผลของ code ข้างล่าง เป็นผลให้ โปรแกรมเข้าไปใน loop แล้วพิมพ์ “Welcome to Java!” ไปเรื่อย ๆ ไม่รู้จบ ไม่สามารถออกจาก loop ได้เนื่องจากค่าของ counter ไม่ได้ถูกทำให้เพิ่ม ทำให้เงื่อนไขเป็นจริงตลอดเวลา

```
int counter = 0;
while ( counter < 100 ) {
        System.out.println(“Welcome to Java!”);
}
counter++;
```

บางครั้ง เราต้องการเขียนโปรแกรมให้ทำซ้ำ แต่ไม่รู้จำนวนที่จะทำซ้ำแน่นอน เช่น บางครั้งเราอาจวน loop รับ input จากผู้ใช้ไปเรื่อย ๆ จนกว่าผู้ใช้จะใส่ค่าที่เป็น sentinel (ค่าที่รู้กันระหว่างผู้เขียนโปรแกรมกับผู้ใช้ว่า ผู้ใช้ต้องการให้โปรแกรมเลิกรอรับค่าได้) ดังตัวอย่างต่อไปนี้

```
0:  import java.util.Scanner;
1:  Scanner my_obj = new Scanner(System.in);
2:  int counter = my_obj.nextInt();
3:  while ( counter != -1 ) {
4:          System.out.println(“Welcome to Java!”);
5:          counter = my_obj.nextInt();
6:  }
```

บรรทัดที่ 2 แสดงการรับค่าจำนวนเต็มชนิด int จาก keyboard มาตั้งค่าเริ่มต้นให้ตัวแปร counter ซึ่งจะถูกเช็คเงื่อนไขในบรรทัดที่ 3 ถ้าค่าใน counter ไม่เท่ากับ -1 โปรแกรมจะทำการดำเนินการคำสั่งทั้งหมดใน block { } ของ while (สังเกตว่า เป็นเงื่อนไขตรวจสอบว่า ยังไม่ถึงเป้าหมายใช่มั้ย คือ input ยังไม่เท่ากับ 1 ในทำต่อไปเรื่อย ๆ) ในบรรทัดที่ 4 และ 5 นั่นคือ print “Welcome to Java!” ออกทางหน้าจอที่ต้น loop และรอรับค่าจาก Keyboard แล้วนำค่ามาใส่ในตัวแปร counter ใหม่ที่ท้าย loop

สังเกตว่า ทุก ๆ ครั้งที่โปรแกรมจะดำเนินการกลุ่มคำสั่งใน loop จะ check ค่าในตัวแปร counter ก่อน ดังนั้น ถ้าเราอยากจะให้ โปรแกรมรันจบไม่ติด loop เราต้องมั่นใจว่า เงื่อนไขใน ( ) มีโอกาสเป็นเท็จ ซึ่งใน code ข้างต้น ผู้ใช้มีโอกาสเปลี่ยนค่าใน counter ได้ที่ท้ายของ loop ทุก ๆ loop ช่วยให้โปรแกรมออกจาก loop ได้ถ้าผู้ใช้ใส่ค่า -1 จาก key board

ตัวอย่างการตั้งเงื่อนไขของ while loop ที่มีปัญหา

ตัวอย่างที่ 1

```
1:  i = 5;
2:  while ( i > 0 ) {
4:        System.out.println(i + " ");
5:        i++;
6:  }
```

เนื่องจาก กำหนดค่า i เริ่มต้นที่ 5 ซึ่งทำให้เงื่อนไขในบรรทัดที่ 2 เป็นจริง เมื่อเข้ามาในลูป บรรทัดที่ 5 เพิ่มค่า i มากขึ้นไปอีก i จึงไม่เคยน้อยกว่าหรือเท่ากับ 0 ซะที ทำให้ทำงานไม่รู้จบ

ตัวอย่างที่ 2

```
1:  i = 5;
2:  while ( i > 5 ) {
4:        System.out.println(i + " ");
5:        i--;
6:  }
```

เนื่องจาก กำหนดค่า i เริ่มต้นที่ 5 ซึ่งทำให้เงื่อนไขในบรรทัดที่ 2 เป็นเท็จ คำสั่งในลูปก็จะไม่เคยถูกเรียกทำงาน ดังนั้นมีบรรทัดที่ 2-6 ก็เหมือนไม่มี ไม่ต้องใส่มาให้สับสนก็ได้

ตัวอย่างที่ 3

```
1:  i = 5;
2:  while ( i < 0 ) {
4:        System.out.println(i + " ");
5:        i--;
6:  }
```

เหมือนกับตัวอย่างที่ 3 เนื่องจาก กำหนดค่า i เริ่มต้นที่ 5 ซึ่งทำให้เงื่อนไขในบรรทัดที่ 2 เป็นเท็จ คำสั่งในลูปก็จะไม่เคยถูกเรียกทำงาน ดังนั้นมีบรรทัดที่ 2-6 ก็เหมือนไม่มี

ตัวอย่างที่ 4

```
1:  i = 1;
2:  while ( i < 20 ) {
4:        System.out.println(i + " ");
5:  }
```

เนื่องจาก กำหนดค่า i เริ่มต้นที่ 1 ซึ่งทำให้เงื่อนไขในบรรทัดที่ 2 เป็นจริง คำสั่งในลูปถูกเรียกทำงาน แต่ค่า i ไม่เคยถูกเปลี่ยนเลย !!!! นั่นคือ ค่าใน i จะเป็นหนึ่งตลอดกาล เงื่อนไขในบรรทัดที่ 2 ไม่เคยเป็นเท็จ ทำให้โปรแกรมทำงานไม่รู้จบ

จากตัวอย่างข้างต้น ขออธิบายซ้ำอีกทีว่า เงื่อนไขใน ( ) ของ while คือเงื่อนไขที่ให้ while ทำงานต่อไปเรื่อย ๆ จนกว่าเงื่อนไขจะเป็นเท็จ นิสิตต้องตั้งเงื่อนไขในนี้ให้ดี มิฉะนั้นโปรแกรมอาจทำงานไม่รู้จบเพราะติดลูป ซึ่งรูปแบบในการตั้งเงื่อนไขส่วนใหญ่ จะประกอบไปด้วย 1) ตัวแปรที่มีค่าเปลี่ยนไปเรื่อย ๆ เมื่ออยู่ในลูป ซึ่งจะถูกนำไปเปรียบเทียบค่ากับ 2) ค่าเป้าหมาย โดยใช้ 3) ตัวดำเนินการเปรียบเทียบ หรือ comparison operator

Statement For

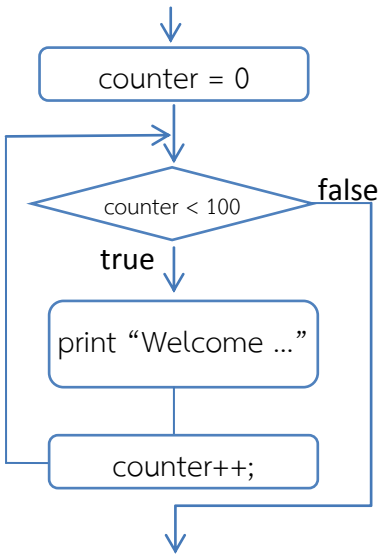
พิจารณา code while loop ด้านล่างต่อไปนี้:-

```
1:  int counter = 0;
2:  while ( counter < 100 ) {
3:        System.out.println("Welcome to Java!");
4:        counter++;
5:  }
```

code while loop ข้างต้น (บรรทัดแรก ตั้งค่าตั้งต้นให้ counter, บรรทัดที่ 2 check condition ก่อนเข้า loop, และ บรรทัดที่ 4 เพิ่มค่าใน counter 1 ค่า) สามารถนำมาเขียนเป็น for loop ที่ให้ผลในการดำเนินการเหมือนกันได้ดังนี้

```
1:  int counter;
2:  for (counter = 0; counter < 100; counter++ ) {
3:        System.out.println("Welcome to Java!");
4:  }
```

ซึ่งสามารถเขียน flow chart ได้ดังนี้



การทำงานของ for statement ข้างต้นเริ่มจากตั้งค่าตั้งต้นให้ตัวแปร counter เท่ากับ 0 จากนั้นโปรแกรมจะ check เงื่อนไขว่า counter น้อยกว่า 100 หรือไม่ ซึ่ง 0 น้อยกว่า 100 โปรแกรมจะเข้า loop มาเพื่อดำเนินการพิมพ์ “Welcome to Java!” จากนั้นค่า counter จะถูกเพิ่มขึ้นเป็น 1 แล้วโปรแกรมจะวนกลับไป check เงื่อนไขเดิมว่า counter น้อยกว่า 100 หรือไม่ ซึ่ง 1 น้อยกว่า 100 โปรแกรมก็จะเข้า loop เป็นครั้งที่ 2 เพื่อมาดำเนินการพิมพ์ “Welcome to Java!” จากนั้นค่า counter จะถูกเพิ่มขึ้นเป็น 2 โปรแกรมจะทำซ้ำเรื่อยไปจนค่าของ counter เท่ากับ 100 จึงจะออกจาก loop

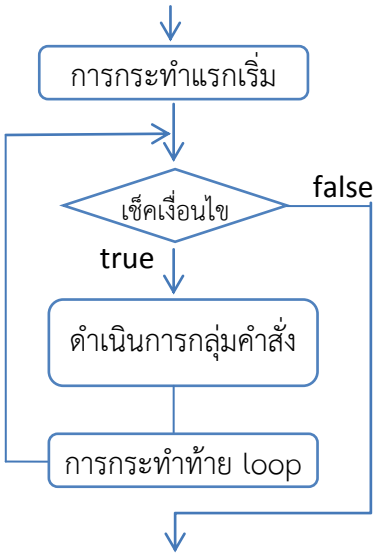
รูปแบบการเขียน loop (มีการตั้งค่าตั้งต้น, มีการเช็คเงื่อนไขก่อนเข้า loop อีกครั้ง, และ มีการดำเนินการอย่างหนึ่งซ้ำ ๆ กันทุกครั้งที่ท้าย loop) ข้างต้นพบได้บ่อยมาก ซึ่งบางครั้ง เราสามารถใช้ for loop ในการเขียน loop ลักษณะนี้ได้ ซึ่งข้อดีคือ for statement จะรวมการตั้งค่าตั้งต้น การเช็คเงื่อนไขก่อนเข้า loop และ การดำเนินการท้าย loop ไว้อยู่ในบรรทัดเดียว (คั่นด้วย ;) โดยใส่ไว้ใน ( ) หลัง for ทำให้ผู้อ่าน code สามารถทำความเข้าใจการทำงานของ loop นั้น ๆ ได้มากยิ่งขึ้น <u>โดยส่วนใหญ่ for loop จะถูกใช้เมื่อเรารู้จำนวนของการวน loop แน่นอน</u>

พิจารณาจาก code for loop ข้างต้น เราจะเห็นได้ว่า for statement มี syntax และ flow chart ดังนี้

for (คำสั่งแรกและทำเพียงครั้งเดียว **;** เงื่อนไขเพื่อเข้า **loop;** คำสั่งท้าย **loop ทำก่อน** ขึ้น **loop** ใหม่**) {**

คำสั่งหรือกลุ่มคำสั่ง;

}



การกระทำเริ่มแรก เงื่อนไขเพื่อเข้า loop และ การกระทำท้าย loop ก่อนขึ้น loop ใหม่ จะอยู่ในวงเล็บเดียวกันหลังคำสั่ง for และมีเครื่องหมาย ; คั่น

ให้สังเกตุ flow chart ว่าการกระทำแรกเริ่มถูกดำเนินการครั้งเดียวเท่านั้น (ก่อนที่จะ check เงื่อนไขเข้า loop แรก) แม้ว่าโปรแกรมจะวน loop มาใหม่อีกกี่ครั้ง โปรแกรมจะไม่ดำเนินการการกระทำนี้อีก ส่วนใหญ่การกระทำแรกเริ่มจะเป็นการประกาศและตั้งค่าตั้งต้นตัวแปรที่ใช้ในการนับจำนวน loop เช่น counter = 0;

*** บางครั้ง ผู้เขียนอาจไม่ใส่อะไรเลยหรือใส่ comma เข้าไปในส่วนของ การกระทำแรกเริ่ม และ การกระทำท้าย loop ก็ได้ เช่น

// ตัวอย่างที่ 1

```
1:  int counter = 0;
2:  for (  ; counter < 100;  ) {
3:        System.out.println("Welcome to Java!");
4:        counter++;
5:  }
```

( ) หลัง for ใน code ข้างต้น มี ; 2 ตัว และมีแต่ส่วนเงื่อนไข นั่นคือ ผู้เขียนบอก Java ว่าไม่ต้องดำเนินการในส่วนการกระทำแรกเริ่มและในส่วนการกระทำท้าย loopในวงเล็บ

// ตัวอย่างที่ 2

```
1:  for ( int i = 0, j = 0 ; (i + j) < 100;  i++, j+=2 ) {
2:        System.out.println("Welcome to Java!");
3:  }
```

ในตัวอย่างที่ 2 ของ code ข้างต้น เราได้ประกาศตัวแปร i กับ j และตั้งค่า 0 ให้ทั้ง i และ j ในส่วนการกระทำแรกเริ่มใน ( ) หลัง for สำหรับการกระทำท้าย loop ค่า i ถูกเพิ่มทีละ 1 ขณะที่ j ถูกเพิ่มทีละ 2 loop จะจบก็ต่อเมื่อ ผลรวมค่า i และ j มีค่ามากกว่าหรือเท่ากับ 100

// ตัวอย่างที่ 3

```
for (int i = 0; i < 100; System.out.println(i++));
```

code ในตัวอย่างที่ 3 สามารถถูก run ได้โดยไม่มี error โดยมันจะตั้งค่าตั้งต้นให้ i เท่ากับ 0 จากนั้นจะเข้าไปใน loop และพิมพ์ค่า i ออกมาก่อนค่อยทำการเพิ่มค่าของ i ทีละหนึ่งค่า loop รันจนกว่าค่า i จะเท่ากับ 100 นั่นคือ code ข้างต้นจะพิมพ์ค่า i ตั้งแต่ 0 ถึง 99 (ทั้งหมด 100 บรรทัด) ออกทางหน้าจอ

// ตัวอย่างที่ 4

```
1:  for ( int i = 10, i >= 0;  i--) {
2:        System.out.println(“Welcome to Java!”);
3:  }
```

code ในตัวอย่างที่ 4 จะทำ println ซ้ำ 11 ครั้ง ค่า i ค่อย ๆ ลดทีละหนึ่ง (เพราะส่วนที่สามใน วงเล็บของ for loop เป็น i--) จาก 10 ไปถึง 0

// ตัวอย่างที่ 5

```
1:  for ( int i = 0, i !=  9;  i+2) {
2:        System.out.println(i+" ");
3:  }
```

code ในตัวอย่างที่ 5 จะทำ println ค่า i ดังนี้ 0, 2, 4, 6, 8 10, 12, ... ไปเรื่อย ๆ ไม่รู้จบเพราะ i ไม่เท่ากับ 9

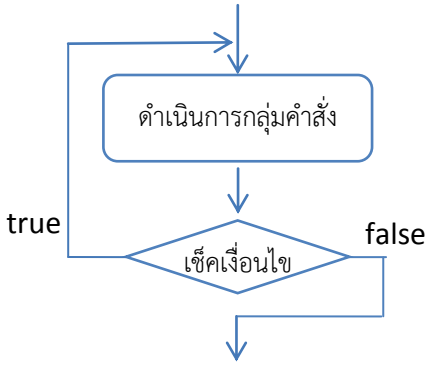
*** บางครั้งถ้าเราต้องการเขียนโปรแกรมให้ loop ถูกรันตลอดไป เราอาจใช้ for หรือ while ดังนี้

```
for ( ; ; ) {
        // Do something
}
```

(ซึ่งเหมือนกับ)

```
while (true) {
      // Do something
}
```

Statement Do...While



บางครั้งเราต้องการเขียนโปรแกรมเพื่อดำเนินการซ้ำ แต่เราต้องการให้ดำเนินการกับกลุ่มคำสั่งก่อนที่จะตรวจสอบเงื่อนไขดังแสดงใน flow chart ข้างต้น (ตัวอย่างเช่น เราสั่งให้โปรแกรมวนรับ input จากผู้ใช้ไปเรื่อย ๆ

จนกว่า ผู้ใช้จะป้อน input ที่ถูกต้อง) Java มีคำสั่ง do...while เพื่อรองรับรูปแบบการทำซ้ำแบบนี้ โดย syntax ของ statement do..while มีดังนี้

```
do {
        คำสั่งหรือกลุ่มคำสั่ง ;
}  while ( เงื่อนไข );
```

สังเกตุในกรณี do...while เราจะมี ; อยู่หลัง (เงื่อนไข) ได้และจะไม่ทำให้เกิดความผิดพลาดในการใช้ loop เหมือนในกรณีของ while อย่างเดียว

นอกจากนี้ ให้สังเกตุความแตกต่างระหว่าง flow chart ของ while และของ do...while คำสั่งหรือกลุ่มคำสั่งใน { } หลัง do...while จะถูกดำเนินการก่อนไม่ว่าเงื่อนไขจะเป็นจริงหรือเท็จก็ตาม (ต่างจาก statement while ที่เงื่อนไขต้องเป็นจริงเท่านั้น) ดังนั้นคำสั่งหรือกลุ่มคำสั่งใน { } หลัง do จะถูกดำเนินการแน่ ๆ อย่างน้อยหนึ่งครั้ง ดังตัวอย่าง

```
1:  int counter = 100;
2:  do {
3:        System.out.println("Welcome to Java!");
4:        counter++;
5:  } while ( count < 100 );
```

ในตัวอย่างข้างต้น แม้ในบรรทัดที่ 1 counter จะถูกตั้งค่าไว้ที่ 100 แล้ว แต่โปรแกรมก็ยังจะพิมพ์ "Welcome to Java!" ออกมาหนึ่งครั้ง ก่อนที่จะเพิ่มค่า counter ให้เป็น 101 และออกจาก loop เนื่องจากเงื่อนไขเป็นเท็จ

การเลือกชนิดของ loop

โดยปกติแล้ว loop ทั้ง 3 ชนิด (while, for, do...while) จะสามารถเขียนแทนกันได้ โดยให้ผลการดำเนินการคำสั่งเท่ากัน เช่น

```
while ( เงื่อนไข ) {
        คำสั่งหรือกลุ่มคำสั่ง ;
```

จะให้ผลลัพธ์เหมือนกับ

```
for ( ; เงื่อนไข;  ) {
        คำสั่งหรือกลุ่มคำสั่ง ;}
}
```

หรือ

```
for ( คำสั่งเริ่มต้น; เงื่อนไข ; คำสั่งท้าย loop ) {
        คำสั่งหรือกลุ่มคำสั่ง ;
}
```

จะให้ผลลัพธ์เหมือนกับ

```
คำสั่งเริ่มต้น ;
while ( เงื่อนไข ) {
        คำสั่งหรือกลุ่มคำสั่ง ;
        คำสั่งท้าย loop;
}
```

ดังนั้น เราจึงเลือกใช้รูปแบบ loop ใดก็ได้ที่ทำให้เราคิดว่า จะทำให้การเขียน code และ การทำความเข้าใจโปรแกรมของเราง่ายที่สุด ซึ่งโดยปกติ for loop จะถูกใช้เมื่อเรารู้จำนวนของการวน loop แน่นอน เช่น ต้องการพิมพ์ข้อความ 100 บรรทัด while loop จะถูกใช้เวลาจำนวนของการวนไม่แน่นอน เช่นในกรณีของโปรแกรมรับค่า input จากผู้ใช้ไปเรื่อย ๆ จนกว่าผู้ใช้จะใส่ 0

ส่วน do...while จะถูกใช้เวลาเราจะดำเนินการคำสั่งหรือกลุ่มคำสั่งก่อนตรวจสอบเงื่อนไข

Break และ Continue

เราเคยเรียนรู้การใช้ break ใน statement switch มาแล้ว ซึ่งการใช้ break ใน statement switch เป็นการบอกให้โปรแกรมหยุดดำเนินการคำสั่งใด ๆ ใน statement switch และให้ออกจาก statement switch ได้ การใช้ break ใน loop for, while, do...while ก็เหมือนกัน มันเป็นการบอกโปรแกรมของเราให้หยุดทำงานใน loop และ ออกจาก loop ได้ ดังตัวอย่างต่อไปนี้

```
for ( int i = 0; i < 10000; i++ ) {
        if ( i == 2000 ) {
                break;
        }
        System.outprintln(i);
}
```

โปรแกรมข้างต้นจะพิมพ์ค่า i ตั้งแต่ค่า 0 ถึง 1999 ออกทางหน้าจอก่อนจะออกจาก loop ด้วยคำสั่ง break

การใช้ continue ใน loop for, while, do...while เป็นการบอกโปรแกรมของเราให้ละการทำงานที่เหลือใน loop รอบปัจจุบัน แล้วให้ไปขึ้นรอบใหม่เลย ดังตัวอย่างต่อไปนี้

```
for ( int i = 0; i < 10000; i++ ) {
        if ( i == 2000 ) {
                continue;
        }
        System.out.println(i);
}
```

โปรแกรมข้างต้นจะพิมพ์ค่า i ตั้งแต่ค่า 0 ถึง 1999 ออกทางหน้าจอ จากนั้นพอ i มีค่าเท่ากับ 2000 คำสั่ง continue จะถูกดำเนินการ ทำให้โปรแกรมหยุดดำเนินการใน loop รอบปัจจุบัน (ไม่พิมพ์ค่า 2000 ออกมา) และขึ้น loop รอบใหม่ แล้วทำการพิมพ์ค่า 2001 จากนั้นวน loop เรื่อยไปจนพิมพ์ 9999 ออกทางหน้าจอ โปรแกรมจึงจะออกจาก loop

Nested Loop

บางครั้ง ในแต่ละการทำซ้ำ (ในแต่ละ loop) เราก็ต้องการให้โปรแกรมวนซ้ำทำงานอีกอย่างหนึ่งให้เรา ลักษณะนี้เป็นการทำ loop ซ้อน loop (Nested loop) ยกตัวอย่างเช่น ถ้าเราต้องการพิมพ์เลข 1 ถึง 12 ให้อยู่ในบรรทัดเดียวกันออกทางหน้าจอ เราอาจใช้แค่ loop เดียว แต่ถ้าเราต้องการพิมพ์บรรทัดดังกล่าว 20 บรรทัด เราอาจใช้ loop ซ้อน loop ในการดำเนินการ ดังนี้

```
1: int i, j;
2: for ( i = 0; i < 20; i++ ) {
3:      for( j = 0; j < 12; j++) {
4:              System.out.print(j + " ");
5:      }
6:      System.out.println(" ");
7: }
```

เราประกาศตัวแปร i และ j ในบรรทัดที่ 1 (ความจริงเราประกาศตัวแปร i ใน () หลังคำสั่ง for ที่บรรทัดที่ 2 และ ประกาศตัวแปร j ใน ( ) หลัง for ที่บรรทัดที่ 3 ก็ได้)

เราเรียก for loop ในบรรทัดที่ 2 ว่า loop ด้านนอก (outer loop) เราเรียก for loop ในบรรทัดที่ 3 ว่า loop ด้านใน (inner loop) ในที่นี้ loop ด้านนอกจะถูกวนจำนวน 20 ครั้ง แต่ละครั้งของ loop ด้านด้านนอก loop ด้านในจะถูกวน 12 ครั้ง นั่นคือบรรทัดที่ 3, 5, 6 จะถูกดำเนินการ 20 ครั้ง แต่บรรทัดที่ 4 จะถูกดำเนินการ 20 * 12 = 240 ครั้ง

ข้อสังเกตุ 1 ถ้าเราต้องการทำ loop ซ้อน loop และต้องการนับจำนวน loop ทั้ง loop ด้านนอกและด้านใน เราควรใช้ตัวแปรที่ใช้นับ loop ต่างกัน (ในที่นี้ เราใช้ i นับจำนวน loop ด้านนอก และ ใช้ j นับจำนวน loop ด้านใน)

ข้อสังเกตุ 2 ในบรรทัดที่ 3 j จะถูกตั้งค่าเริ่มต้นให้เท่ากับ 0 ทุกครั้ง ก่อนที่เข้าไปวน loop ด้านใน ทำให้เรามั่นใจได้ว่า จำนวนการวน loop ด้านในจะเริ่มจาก j เท่ากับ 0 ถึง j เท่ากับ 11 ทุกครั้ง เราจึงไม่ควรลืม reset ค่า j ก่อนเริ่มวน loop ด้านในทุกครั้ง

ข้อสังเกตุ 3 การประกาศตัวแปร i และ j ไม่ต้องทำที่บรรทัดที่ 1 โดยเราสามารถประกาศตัวแปร i ใน ( ) หลัง for ในบรรทัดที่ 2 และ สามารถประกาศตัวแปร j ใน ( ) หลัง for ในบรรทัดที่ 3

ข้อสังเกตุ 4 สำหรับที่นิสิตที่เริ่มเขียนโปรแกรมใหม่ ๆ ให้นิสิตเริ่มเขียนจาก loop ด้านในให้เสร็จก่อนแล้วค่อยมาเขียนโค้ดสำหรับ loop ด้านนอกครอบโค้ดของ loop ด้านในอีกที นอกจากนี้ นิสิตควร println ค่าของตัวแปรสองตัวที่ควบคุมจำนวน loop ทั้งสองด้วย (นั่นคือ ตัวแปร i และ j) เพื่อตรวจสอบศึกษาการทำงานของลูป

ให้นิสิตสังเกตุตัวอย่างต่อไปนี้

```
1. for (i = 1; i <= 4; i++) {
2.     for (j = 1; j <= i; j++) {
3.             System.out.print("*");
4.     }
5.      System.out.println( );
6. }
```

ซึ่งให้ output ดังต่อไปนี้

```
*
**
***
****
```

สังเกต loop ด้านใน (บรรทัดที่ 2-4) มี j เป็นตัวแปรควบคุมจำนวน loop จะเห็นได้ว่า j จะเริ่มจาก 1 เสมอ และค่าของมัน จะเพิ่มทีละหนึ่งไปเรื่อย ๆ จนถึงค่า i (ในส่วนที่สองของ ( ) ของ for มีเงื่อนไข j <= i) นั่นคือ โปรแกรมจะสั่ง print * ในบรรทัดที่ 3 ซ้ำ i ครั้ง ดังนั้น พอออกจาก loop นี้แล้ว จะเห็น * ติดกัน i ตัว

พอออกจาก loop ด้านใน โปรแกรมจะทำต่อที่บรรทัดที่ 5 ซึ่งมีคำสั่ง System.out.println( ); เป็นการแสดงการขึ้นบรรทัดใหม่ที่ console ซึ่งบรรทัดที่ 5 อยู่ใน loop ด้านนอก นั่นคือ จะมีการขึ้นบรรทัดใหม่ใน console ตามจำนวนการถูกเรียกซ้ำของ loop ด้านนอก ซึ่ง ตัวแปร i เป็นตัวแปรควบคุม loop ด้านนอกนี้ (ดังนั้น i ใช้แทนเลขของบรรทัดใน console ได้ด้วย) ให้ดูรายละเอียดดังนี้

เมื่อ i = 1, ค่า j เริ่มที่ 1 หยุดที่ 1 บรรทัดที่ 3 ถูกเรียกแค่ครั้งเดียว บรรทัดที่ 5 ถูกเรียกหนึ่งครั้ง ทำให้ console แสดงผลดังนี้

```
*
```

เมื่อ i = 2, ค่า j เริ่มที่ 1 หยุดที่ 2 บรรทัดที่ 3 ถูกเรียกอีก 2 ครั้ง บรรทัดที่ 5 ถูกอีกเรียกหนึ่งครั้ง ทำให้ console แสดงผลดังนี้

```
*
**
```

เมื่อ i = 3, ค่า j เริ่มที่ 1 หยุดที่ 3 บรรทัดที่ 3 ถูกเรียกอีก 3 ครั้ง บรรทัดที่ 5 ถูกอีกเรียกหนึ่งครั้ง ทำให้ console แสดงผลดังนี้

```
*
**
***
```

เมื่อ i = 4 ค่า j เริ่มที่ 1 หยุดที่ 4 บรรทัดที่ 3 ถูกเรียกอีก 4 ครั้ง บรรทัดที่ 5 ถูกอีกเรียกหนึ่งครั้ง ทำให้ console แสดงผลดังนี้

```
*
**
***
****
```

เมื่อ i = 5, เงื่อนไขในส่วนที่ 2 ของ ( ) ของ for ในบรรทัดที่ 1 จะเป็นเท็จ ทำให้ไม่เข้า loop อีก

ข้อควรระวังสำหรับการใช้เลขทศนิยมในเงื่อนไขก่อนจะเข้า loop

เราไม่ควรใช้เลขทศนิยมหรือตัวแปรทศนิยมในการสร้างเงื่อนไข (ประเภทการเปรียบเทียบเท่ากับหรือไม่เท่ากับ) เพื่อควบคุม loop เพราะค่าทศนิยมบางค่าเป็นแค่การประมาณ ทำให้การควบคุมไม่ถูกต้อง ดังตัวอย่างต่อไปนี้

```
// data should be zero
double data = (Math.sqrt(2)* Math.sqrt(2)) - 2;
while (data != 0) {
        System.out.println("data is not zero");
}
```

จะเห็นว่า ในทางคณิตศาสตร์ square root ของ 2 คูณกับ square root ของ 2 ต้องได้ 2 เมื่อนำมา – 2 แล้วจะต้องได้ค่า 0 แต่เนื่องจาก Math.sqrt(2) ให้ผลลัพธ์เป็นชนิด double ซึ่งสามารถเก็บตำแหน่งของทศนิยมได้จำกัด ทำให้ทศนิยมบางส่วนโดนตัดทิ้ง square root ของ 2 คูณกับ square root ของ 2 จึงให้ค่าไม่เท่ากับ 2 ทำให้จริง ๆ แล้ว data มีค่าไม่เท่ากับ 0 เป็นผลให้ loop ข้างต้นจึงถูกวนไม่รู้จบ