การควบคุมขั้นตอนการทำงาน (Flow Control)

วัตถุประสงค์

- เข้าใจว่าทำไมต้องมีการใช้เงื่อนไข เพื่อควบคุมขั้นตอนการทำงาน
- เข้าใจตัวแปรพื้นฐานชนิด Boolean
- สามารถประยุกต์ใช้ตัวดำเนินการแบบเปรียบเทียบและแบบตรรกะ
- ใช้คำสั่ง if และ nested if ได้
- สามารถเปรียบเทียบจำนวนเต็ม และ จำนวนทศนิยม
- ใช้คำสั่ง switch ได้

ทำไมต้องมีเงื่อนไข

เพราะ ในบางกรณีเราต้องการให้โปรแกรมของเราทำงานอย่างหนึ่ง แต่ในบางกรณีเราต้องการให้โปรแกรมทำงานอีกอย่างหนึ่ง ตัวอย่างเช่น

- ในการ log in เข้า Facebook ระบบต้องเช็คว่า ผู้ใช้ใส่ user name และ password เข้ามาผิดหรือไม่ **ถ้าใส่ผิด**เข้ามา ระบบจะแจ้งผู้ใช้ให้ใส่ใหม่ **แต่ถ้าใส่ถูก** ระบบจะเริ่มดำเนินการ
- ในบางโปรแกรม เราต้องการให้โปรแกรมตัดสินใจเพื่อเลือกวิธีการคำนวณ เช่น **ถ้าอายุของผู้ใช้น้อยกว่า 20** เราจะคำนวณแบบหนึ่ง **แต่ถ้าอายุของผู้ใช้มากกว่าหรือเท่ากับ 20** เราจะคำนวณอีกแบบหนึ่ง

ในภาษา Java เราสามารถควบคุมสายการทำงานของโปรแกรมได้โดยใช้คำสั่ง if, else หรือ switch

Boolean

boolean เป็นชนิดข้อมูลพื้นฐานชนิดหนึ่ง โดยค่าของมันจะเป็น true หรือ false เท่านั้น boolean เกี่ยวข้องกับ if เพราะ if ต้องเช็คว่า เงื่อนไขเป็นจริง (true) หรือ เท็จ (false) ก่อนตัดสินใจเลือกสายการทำงาน

Operator ที่เกี่ยวข้อง

1. ตัวดำเนินการเกี่ยวกับการเปรียบเทียบ (Comparison Operators) เป็น binary operator (ใช้ operand 2 ตัวในการดำเนินการ) โดยชนิดของ operand ทั้งสองเป็นชนิด short, byte, int, float, หรือ double ค่าจากการดำเนินการของ comparison operator จะได้เป็นชนิด boolean เสมอ

ตัวดำเนินการ	ชื่อ
<	น้อยกว่า
<=	น้อยกว่าหรือเท่ากับ
>	มากกว่า
>=	มากกว่าหรือเท่ากับ
==	เท่ากับ
!=	ไม่เท่ากับ

เช่น expression 10 < 14 จะให้ค่าเป็น true

expression 10 > 14 จะให้ค่าเป็น false

expression 10 != 14 จะให้ค่าเป็น true

2. ตัวดำเนินการเกี่ยวกับตรรกะแบบบูล (Boolean Operators) เป็น binary operator เช่นกัน โดยชนิดของ operand ทั้งสองต้องเป็นชนิด boolean ทั้งคู่ ค่าจากการดำเนินการของ boolean operator จะได้เป็นชนิด boolean เสมอ

ตัวดำเนินการ	ชื่อ
!	not
&&	logical and
||	logical or
^	exclusive or

เช่น expression (2 < 3) && (5 < 1) จะให้ค่าเป็น false

expression true && false จะให้ค่าเป็น false

ตาราง Truth Table ของตัวดำเนินการเกี่ยวกับตรรกะบูล มีดังนี้

A	! A
true	false
false	true

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

A	B	A ^ B
true	true	false
true	false	true
false	true	true
false	false	false

การใช้ statement if

คำสั่ง if ถูกใช้เมื่อเราต้องการให้คอมพิวเตอร์ทำงานบางอย่างก็ต่อเมื่อเงื่อนไขเป็นจริงหรือเท็จ (if เป็นคำสั่ง 2 ทางเลือก ทางเลือกในกรณีเงื่อนไขเป็นจริง และ ในกรณีเงื่อนไขเป็นเท็จตามลำดับ) โดยการใช้ if มีไวยากรณ์ (syntax) ดังนี้

```
if (เงื่อนไข) {
        คำสั่ง หรือ กลุ่มคำสั่ง ที่จะถูกเรียกเมื่อเงื่อนไขเป็นจริง
} else {
        คำสั่ง หรือ กลุ่มคำสั่ง ที่จะถูกเรียกเมื่อเงื่อนไขเป็นเท็จ
}
```

syntax ด้านบน ประกอบไปด้วยส่วนประกอบดังนี้

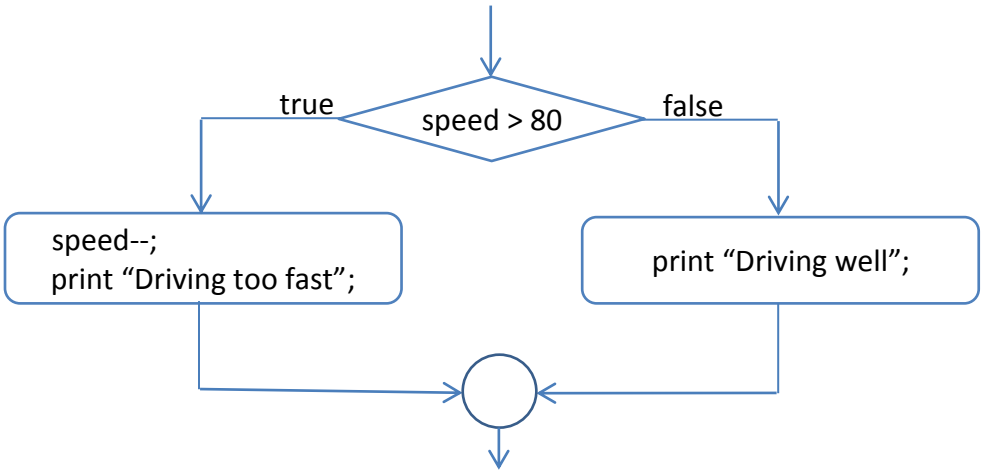
1. คำสั่ง if ตามด้วย เงื่อนไขที่อยู่ใน ( ) เงื่อนไขจะถูกเขียนด้วย expression ที่จะให้ผลเป็น true หรือ false (โดยปกติแล้ว จะมี operator ==, !=, <, <=, >, >=)

2. วงเล็บปีกกาคู่แรกหลัง if
3. คำสั่งหรือกลุ่มคำสั่งที่อยู่ภายในวงเล็บปีกกาหลัง if โปรแกรมจะดำเนินการคำสั่งหรือกลุ่มคำสั่งนี้ ก็ต่อเมื่อ เงื่อนไขใน ( ) หลัง if เป็นจริง
4. คำสั่ง else
5. วงเล็บปีกกาคู่ที่สองที่อยู่ถัดจาก else
6. คำสั่งหรือกลุ่มคำสั่งที่อยู่ภายในวงเล็บปีกกาหลัง else โปรแกรมจะดำเนินการคำสั่งหรือกลุ่มคำสั่งนี้ ก็ต่อเมื่อ เงื่อนไขใน ( ) หลัง if เป็นเท็จ

<u>ตัวอย่างการใช้ statement if</u>

```
// โปรแกรมสำหรับควบคุมความเร็วรถให้น้อยกว่า 80 กม.ต่อชม.
//    ถ้าความเร็วมากกว่า 80 กม.ต่อชม. ลดความเร็วลง และ พิมพ์ข้อความ Driving too fast
//    แต่ถ้าความเร็วน้อยกว่าหรือเท่ากับ 80 กม.ต่อชม. พิมพ์ข้อความ Driving well
if ( speed > 80 ) {
        speed--;
        System.out.println("Driving too fast");
} else {
        System.out.println("Driving well");
}
```

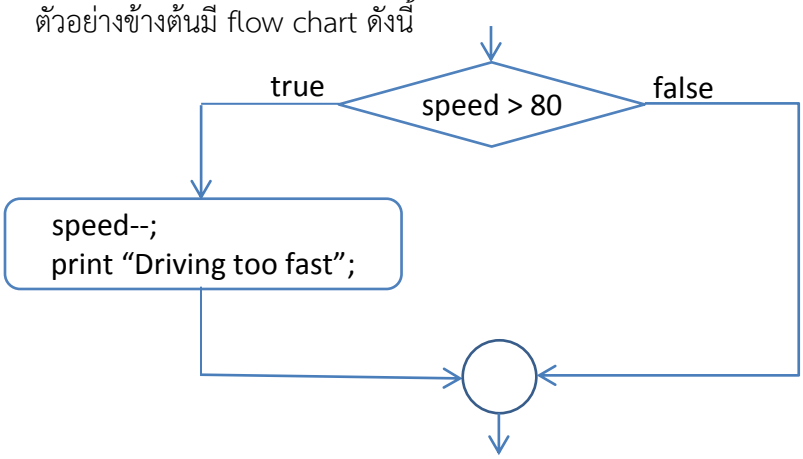
ตัวอย่างข้างต้นมี flow chart ดังนี้



- การใช้ statement if อาจไม่ต้องใช้คู่กับ else ก็ได้ ถ้าเราไม่ต้องการให้โปรแกรมทำอะไรในกรณีเงื่อนไขเป็นเท็จ เช่น

```
// โปรแกรมสำหรับควบคุมความเร็วรถให้น้อยกว่า 80 กม.ต่อชม.
//    ถ้าความเร็วมากกว่า 80 กม.ต่อชม. ลดความเร็วลง และ พิมพ์ข้อความ Driving too fast
//    แต่ถ้าความเร็วน้อยกว่าหรือเท่ากับ 80 กม.ต่อชม. ไม่ต้องทำอะไร
if ( speed > 80 ) {
        speed--;
        System.out.println("Driving too fast");
}
```

ตัวอย่างข้างต้นมี flow chart ดังนี้



- การใช้ statement if ไม่จำเป็นต้องมีวงเล็บปีกกาก็ได้ ถ้าภายในวงเล็บปีกกามีแค่คำสั่ง ๆ เดียว เช่น

```
//    ถ้าความเร็วมากกว่า 80 กม.ต่อชม. ลดความเร็วลง
//    แต่ถ้าความเร็วน้อยกว่าหรือเท่ากับ 80 กม.ต่อชม. พิมพ์ข้อความ Driving well
if ( speed > 80 )
        speed--;
else
        System.out.println("Driving well");
```

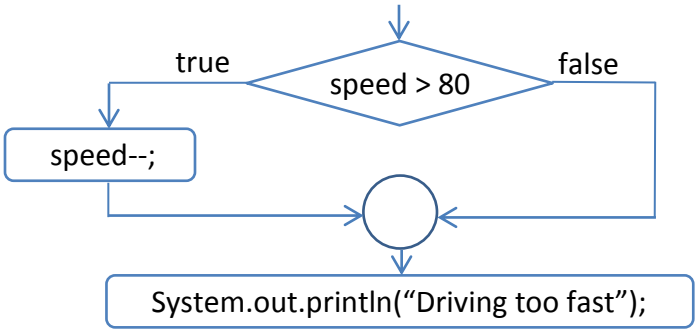
statement ด้านบน จะให้ถูกดำเนินการเหมือนกับกรณีของ statement ที่มีปีกกาด้านล่างนี้

```
if ( speed > 80 ) {
        speed--;
} else {
        System.out.println("Driving well");
}
```

ข้อควรระวัง 1 การย่อหน้าให้ statement เรียงอยู่ในระดับเดียวกัน ไม่ได้เป็นการจัดกลุ่มของ statement ให้อยู่ใน block เดียวกัน เช่น ถ้าเราต้องการให้ โปรแกรมดำเนินการกับ **statement ทั้งหมดในกลุ่มของ statement** เมื่อเงื่อนไขเป็นจริง เราต้องเขียน statement ทั้งกลุ่มให้อยู่ใน block เดียวกัน โดย (นั่นคือ ใส่ stamen ทั้งหมดในวงเล็บปีกกา)

```
if ( speed > 80 )
        speed--;
        System.out.println("Driving too fast");
```

statement if ข้างต้น มี flow chart ดังแสดงต่อไปนี้ (สังเกต ความแตกต่างจาก flow chart ด้านบน)



ข้อควรระวัง 2 การเขียน statement if แล้วตามด้วย ; ดังแสดงข้างล่าง อาจทำให้เกิดความผิดพลาดได้

```
if ( speed > 80 ) ; {
        speed--;
}
```

ความผิดพลาดเกิดขึ้นเพราะ Java ตีความ ; เป็น statement ที่ติดกับ if แม้ว่า มันจะเป็น statement ที่ไม่ได้ทำอะไรเลย

นั่นคือ statement if ด้านบนจะถูกตีความเหมือน statement ด้านล่างนี้

```
if ( speed > 80 ) {
        ;                   // <---- Null statement
}
{
        speed--;
}
```

ในโค้ดข้างต้น ถ้า speed มากกว่า 80 โปรแกรมจะดำเนินการ Null statement (นั่นคือไม่ทำอะไร) จากนั้นจะทำการลดค่าของ speed ลง 1 ค่า แต่ถ้า speed น้อยกว่าหรือเท่ากับ 80 โปรแกรมจะไม่ดำเนินการ

Null statement แต่ก็จะทำการลดค่าของ speed ลง 1 ค่าเหมือนกัน ดังนั้น โค้ดข้างต้นให้ผลลัพธ์เหมือนผลของ speed-- ; บรรทัดเดียว

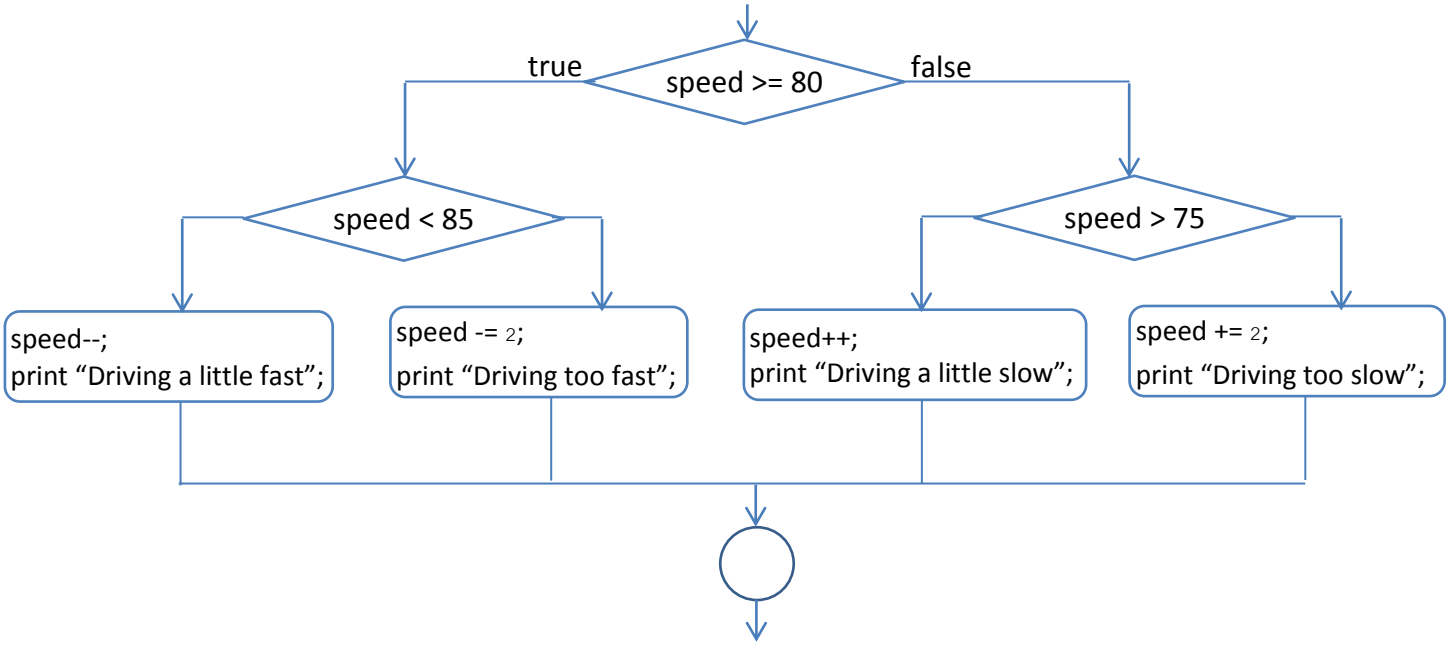
การใช้ statement if ซ้อน if (Nested if)

เวลาเขียนโปรแกรม เราจะพบโจทย์ที่มีเงื่อนไขที่ซับซ้อนอย่างเลี่ยงไม่ได้ ทำให้เราต้องเขียน statement if ซ้อนกัน หลาย ๆ statement ให้พิจารณา code และ flow chart ด้านล่าง

```
/*****************************************************************************
    โปรแกรมสำหรับควบคุมความเร็วรถให้ใกล้เคียง 80 กม.ต่อชม.
      ■ ถ้าความเร็วมีค่าตั้งแต่ 80 กม. ต่อ ชม. แต่ยังไม่ถึง 85 กม. ต่อ ชม. ให้ลดความเร็วลงหนึ่งระดับและ
        พิมพ์ข้อความ Driving a little fast
      ■ ถ้าความเร็วมีค่าตั้งแต่ 85 กม. ต่อ ชม. ขึ้นไป ให้ลดความเร็วลงสองระดับและพิมพ์ข้อความ Driving
        too fast
      ■ ถ้าความเร็วน้อยกว่า 80 กม. ต่อ ชม. แต่ยังมากกว่า 75 กม. ต่อ ชม. ให้เพิ่มความเร็วขึ้นหนึ่งระดับ
        และพิมพ์ข้อความ Driving a little slow
      ■ ถ้าความเร็วน้อยกว่าตั้งแต่ 75 กม. ต่อ ชม. ลงไป ให้เพิ่มความเร็วขึ้นสองระดับและพิมพ์ข้อความ
        Driving too slow
*****************************************************************************/
if ( speed >= 80 ) {
        if ( speed  < 85 ) {
                speed--;
                System.out.println(“Driving a little fast”);
        } else {
                speed -= 2;
                System.out.println(“Driving too fast”);
        }
} else {
        if ( speed > 75) {
                speed++;
                System.out.println(“Driving a little slow”);
        } else {
                speed += 2;
                System.out.println(“Driving too slow”);

        }
}
```

ซึ่งสามารถเขียน flow chart ได้ดังนี้



ซึ่งในทางปฏิบัติ statement if อาจซับซ้อนกว่านี้มาก ดังนั้นเราควรจัด code ของเราให้เรียบร้อย เพื่อหลีกเลี่ยงการหลงคู่วงเล็บปีกกา นอกจากนี้ยังช่วยทำให้การไล่ code เพื่อทำความเข้าใจง่ายขึ้น เพราะเรารู้ว่า statement ใดอยู่รวมกลุ่มกับ statement ใน block ใด

(การเขียนโปรแกรมสำหรับปัญหาข้างต้นสามารถถูกเขียนได้อีกหลายรูปแบบ ให้ลองพิจารณาดูเองเป็นแบบฝึกหัด)

ข้อควรระวัง 3 การใส่ปีกกาจะช่วยให้เราเห็นว่า else เป็นของ if ตัวใดได้ชัดเจนขึ้น พิจารณา code ข้างล่างต่อไปนี้

```
if ( speed >= 80 )
        if ( speed  < 85 )
                speed--;
else
        speed++;
```

code ข้างต้นไม่มีวงเล็บปีกกา ซึ่งเวลา compile จะไม่มี error เพราะ syntax ของมันถูกต้อง แต่ Java มีกฎการจับคู่ if-else ที่ว่า “Java จะจับให้ if ที่ใกล้ else ที่สุดที่เป็นไปได้ให้เป็นคู่กับ else นั้น” ดังนั้น else ใน code ข้างต้นจะคู่กับ if ที่สอง แทนที่จะเป็น if ตัวแรก นั่นคือ โค้ดข้างต้นจะเหมือนกับ

```
if ( speed >= 80 ) {
        if ( speed  < 85 ) {
                speed--;
        } else {
                speed++;
        }
}
```

ถ้าเราต้องการให้ else คู่กับ if ตัวแรก เราต้องใช้วงเล็บปีกกาเพื่อกำหนด block ของ if ทั้งสองให้ถูกต้อง ดังนี้:-

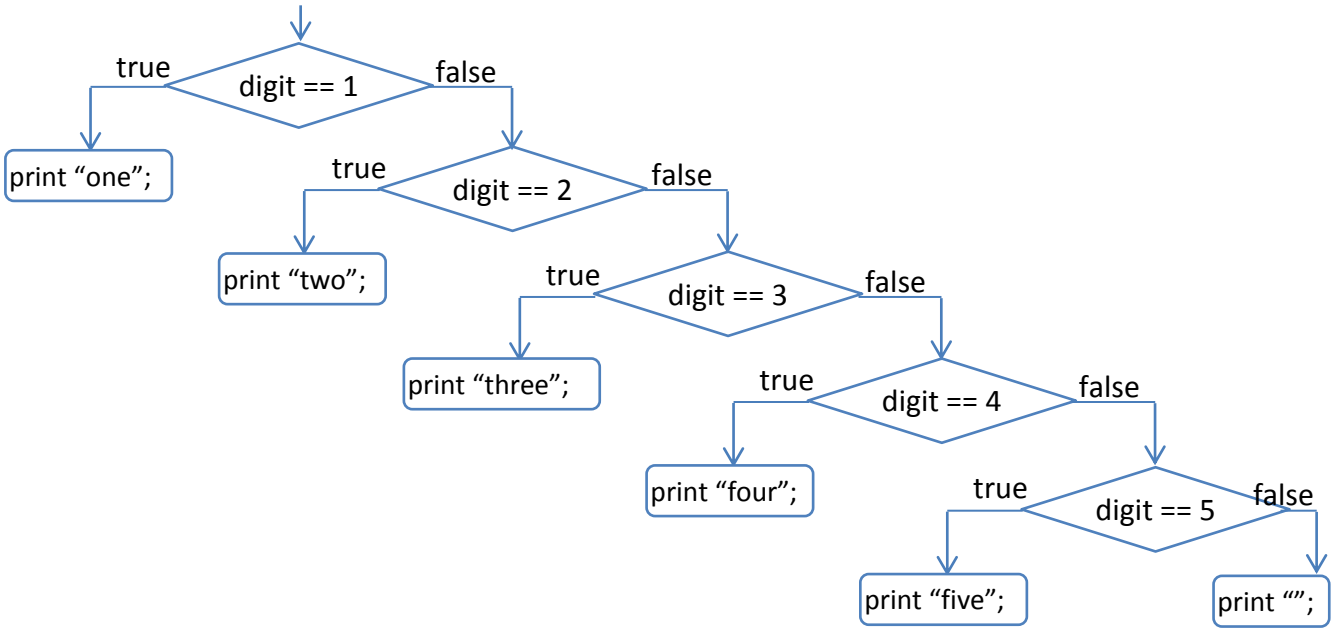
```
if ( speed >= 80 ) {
        if ( speed  < 85 ) {
                speed--;
        }
} else {
        speed++;
}
```

การใช้ switch

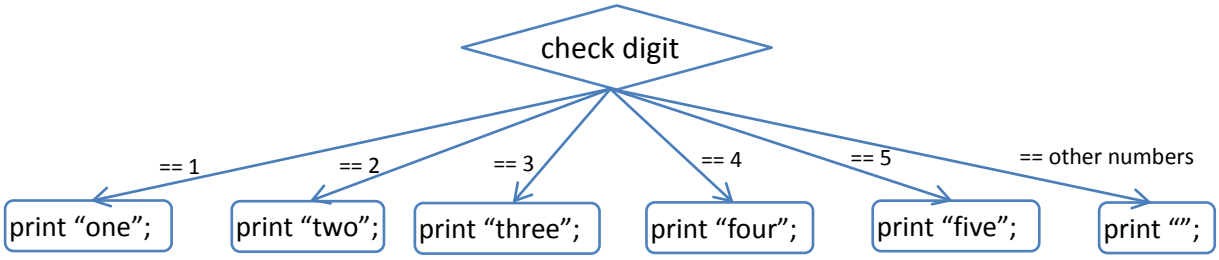
การเขียน code เพื่อควบคุมสายการทำงานของโปรแกรมด้วยการใช้ nested if นั้น มีหลายรูปแบบขึ้นกับลักษณะของปัญหา และมีอยู่รูปแบบหนึ่งที่พบมาก ดังแสดงด้านล่าง

```
if (digit == 1) { System.out.println("one"); }
else {   if (digit == 2) { System.out.println("two"); }
         else {   if (digit == 3) { System.out.println("three"); }
                  else {  if (digit == 4) { System.out.println("four"); }
                          else {   if (digit == 5) { System.out.println("five"); }
                                   else {   System.out.println(""); }
                          }
                  }
         }
}
```

statement if ข้างต้นเป็น statement ที่มีการเลือกตามเงื่อนไข โดยมีการกำหนดเงื่อนไขทุกครั้งที่มีคำสั่ง if (ในที่นี้มี 5 เงื่อนไข) ซึ่งสามารถเขียน flow chart ได้ดังแสดงด้านล่าง



รูปแบบข้างต้นเป็นรูปแบบที่ถูกพบบ่อย แต่ในกรณีที่มีหลายเงื่อนไขมาก ๆ จะทำให้การเขียน nested if มีความซับซ้อนมาก ทั้งนี้เนื่องจาก **statement if เป็นคำสั่งที่มีแค่สองทางเลือก** (ทางเลือกแรกใช้เวลาเงื่อนไขเป็นจริง ทางเลือกที่สองใช้เวลาเงื่อนไขเป็นเท็จ) ถ้าเราสามารถเปลี่ยน flow chart ข้างต้น ให้เป็นเหมือน flow chart ด้านล่างได้ การเขียนโปรแกรมจะชัดเจนขึ้น



Java จึงมี statement อีกชนิดเพื่อรองรับ flow chart ข้างต้น นั่นคือ **statement switch, case และ default ซึ่งเป็นคำสั่งที่มีกี่ทางเลือกก็ได้** โดย statement switch และ case เป็นโครงสร้างที่มีการเลือกสายการทำงานตามค่าของตัวแปร (ในตัวอย่างนี้ มี 6 ทางเลือก นั่นคือทางเลือกเมื่อ digit เท่ากับ 1, 2, 3, 4, 5, และค่าอื่น ๆ) ไม่ใช่ตามเงื่อนไขที่เป็นมากกว่า หรือ น้อยกว่า โดย code ที่ใช้ nested if ข้างต้นสามารถถูกแทนด้วย code ที่ใช้ switch และ case ดังนี้

```
switch (digit) {
        case 1: System.out.println(“one”);
                break;
        case 2: System.out.println(“two”);
                break;
        case 3: System.out.println(“three”);
                break;
        case 4: System.out.println(“four”);
                break;
        case 5: System.out.println(“five”);
                break;
        default: System.out.println(“”);
                break;
}
```

เรานำตัวแปรที่เราต้องการตรวจค่าใส่ไว้ในวงเล็บหลังคำสั่ง switch และวางค่าที่เป็นไปได้ของตัวแปรนั้นไว้หลังคำสั่ง case พร้อมด้วยการทำงานของแต่ละกรณีไว้ด้วย (ค่าที่เป็นไปได้ของตัวแปร และ คำสั่งการทำงานถูกคั่นด้วยเครื่องหมาย colon :)

ถ้าเราอยากให้โปรแกรมดำเนินการอย่างใดอย่างหนึ่ง ในกรณีที่ค่าของตัวแปรเป็นค่าอื่น ๆ นอกเหนือจากค่าที่เป็นไปได้ที่อยู่หลัง case เรานำคำสั่งหรือกลุ่มคำสั่งนั้น ๆ ไปอยู่หลัง : ของ default

statement switch, case และ default จะสามารถตรวจค่าของตัวแปรที่เป็นชนิดจำนวนเต็มได้เท่านั้น ได้แก่ char, short, int, และ long (แม้ char จะเป็นตัวแปรเก็บค่าอักขระ แต่จริง ๆ แล้ว Java แปลงอักขระเป็นจำนวนเต็มก่อนด้วยรหัส ASCII หรือ Unicode แล้วค่อยเก็บ ดังนั้น ในที่นี้ ตัวแปรชนิด char ถือเป็นตัวแปรเก็บค่าจำนวนเต็ม)

จากโปรแกรมย่อยข้างต้น digit เป็นตัวแปรจำนวนเต็มชนิด int สำหรับใช้จัดเก็บแต้มซึ่งมีค่าระหว่าง 1 - 10 โดยโปรแกรมจะตรวจสอบค่าของ digit เพื่อแสดงผล ถ้าค่าของ digit อยู่ระหว่าง 1-5 โปรแกรมจะแสดงผลค่านั้น แต่ถ้าเป็นค่าอื่น โปรแกรมจะขึ้นบรรทัดใหม่และไม่แสดงผลใด ๆ

statement แบบ switch, case และ default เป็น statement แบบทะลุผ่าน ซึ่งหมายความว่า หากค่าของ digit มีค่าเท่ากับค่าของตัวเลขหลัง case ตัวใดแล้ว คำสั่งต่อจาก : ของ case ตัวนั้นจะถูกดำเนินการเรื่อยลงมาตามโครงสร้างที่เหลือจนกว่าจะพบคำสั่ง break เพื่อให้ออกนอกโครงสร้างโดยทันที จาก code ตัวอย่างข้างต้น

มีคำสั่ง break หลังจากทุก ๆ System.out.println ทันที ดังนั้น ไม่ว่า digit จะเป็นค่าใด โปรแกรมจะดำเนินการ println แค่คำสั่งเดียวเท่านั้น แล้วออกจากโครงสร้าง switch

การที่ statement แบบ switch, case และ default เป็น statement แบบทะลุผ่านเป็นประโยชน์มากในบางเวลา เช่น บางครั้งเราต้องการให้ โปรแกรมทำงานเหมือนกันในบางกลุ่มกรณี และทำงานต่างกันในบางกลุ่มกรณี พิจารณาตัวอย่างต่อไปนี้

```
switch (day)  {
   case 1 : System.out.println("This is Saturday");
   case 7 : System.out.println("This is a weekend day");
            break;
   case 2 :
   case 3 :
   case 4 :
   case 5 :
   case 6 : System.out.println("This is a weekday");
             break;
   default : System.out.println("Not a legal day");
             break;
 }
```

ผลของการ run code ข้างต้น เมื่อ day มีค่าต่าง ๆ มีดังนี้

ค่าของ day	**output ของ code ข้างต้น**
1	This is Saturday This is a weekend day
2	This is a weekday
3	This is a weekday
4	This is a weekday
5	This is a weekday
6	This is a weekday
7	This is a weekend day
ค่าอื่น ๆ	Not a legal day

สังเกตุ ข้อมูลออกในกรณีที่ค่าของ day เป็น 1 จะต่างจากในกรณีที่ day เป็น 7 โดยในกรณีแรกข้อมูลออกจะมีบรรทัด “This is Saturday” ด้วย แต่ทั้งสองกรณีจะมีบรรทัด “This is a weekend day” ส่วนในกรณี

ที่ค่าของ day เป็นค่าอื่นที่ไม่ใช้ 1 ถึง 7 โปรแกรมจะเลือกดำเนินการคำสั่งหลัง : ของ default ซึ่งก็คือ พิมพ์ “Not a legal day” ออกทางหน้าจอ