ข้อความสั่ง switch

เราสามารถใช้ข้อความสั่ง if ในการเลือกทางเลือกจากหลายๆทางเลือกได้ อย่างไรก็ตามถ้ามีทางเลือกมาก การใช้ if ซ้อนกันมากๆจะทำให้โปรแกรมดูยุ่งและซับซ้อนขึ้น ทำให้ยากแก่การอ่านและทำความเข้าใจ ภาษาสมัยใหม่จึงมักมีข้อความสั่งสำหรับใช้เลือกทางเลือกใดทางเลือกหนึ่งจากหลายทางเลือกให้ทำงาน สำหรับภาษาจาวามีข้อความสั่ง switch สำหรับทำหน้าที่ดังกล่าว

ข้อความสั่ง switch จะทำการทดสอบค่าของตัวแปรหรือนิพจน์ที่กำหนดเทียบกับค่าคงที่ในรายการต่างๆของ *กรณี* (case) ถ้าพบค่าที่เท่ากัน กลุ่มของข้อความสั่งที่เกี่ยวข้องกับกรณีนั้นก็จะทำงาน รูปแบบทั่วไปของข้อความสั่ง switch เป็นดังนี้

```
switch (นิพจน์)
{
    case นิพจน์ค่าคงที่-1: กลุ่มข้อความสั่ง
    case นิพจน์ค่าคงที่-2: กลุ่มข้อความสั่ง
    ...
    ...
    case นิพจน์ค่าคงที่-n: กลุ่มข้อความสั่ง
    default : กลุ่มข้อความสั่ง
}
```

คำว่า switch case และ default เป็นคำสำคัญ(keyword) *นิพจน์*ของข้อความสั่ง switch เป็นนิพจน์ที่ให้ผลลัพธ์เป็นเลขจำนวนเต็มหรือเป็นอักขระ ส่วนนิพจน์ค่าคงที่-1 นิพจน์ค่าคงที่-2 ... นิพจน์ค่าคงที่-n เป็นค่าคงที่ หรือนิพจน์ที่ให้ค่าคงที่เป็นเลขจำนวนเต็ม (ตัวอักขระในภาษาจาวาถือว่ามีค่าเป็นเลขจำนวนเต็มเช่นกัน) ค่าคงที่เหล่านี้เรียกว่า 'ป้ายกรณี' (case label) ซึ่งเปรียบเสมือนป้ายบอกตำแหน่งของข้อความสั่งที่ต้องจะกระทำการถ้าค่าของ *นิพจน์* มีค่าเท่ากับค่าคงที่หลังคำว่า case ของกรณีนั้น โปรดสังเกตว่าหลัง*นิพจน์ค่าคงที่* แต่ละตัวจะมีเครื่องหมาย colon (:) กำกับอยู่ ซึ่งในภาษาจาวาเราใช้ตัว colon ปิดหลังชื่อเพื่อบอกว่านี่คือป้าย (label) *นิพจน์ค่าคงที่*แต่ละตัวที่อยู่ในข้อความสั่ง switch จะต้องไม่ซ้ำกัน *กลุ่มข้อความสั่ง* หมายถึงข้อความสั่งในภาษาจาวาซึ่งสามารถมีได้หลายข้อความสั่ง หรือจะมีเพียงข้อความสั่งเดียว หรือจะไม่มีข้อความสั่งเลยก็ได้ โครงสร้างของข้อความสั่ง switch แตกต่างจากโครงสร้างของข้อความสั่งอื่นที่มีการดำเนินการหลายข้อความสั่งจำเป็นต้องใส่กลุ่มข้อความสั่งไว้ภายในวงเล็บปีกกา แต่สำหรับกลุ่มข้อความสั่งในข้อความสั่ง switch ไม่จำเป็นต้องอยู่ภายในวงเล็บปีกกา

เมื่อข้อความสั่ง switch เริ่มทำงาน จะมีการเปรียบเทียบค่าของ*นิพจน์*ซึ่งเขียนไว้ในวงเล็บหลังคำสำคัญ switch กับค่าของ *นิพจน์ค่าคงที่-1 นิพจน์ค่าคงที่-2 .... นิพจน์ค่าคงที่-n* ตามลำดับ ถ้าค่าของกรณีใดเทียบได้เท่ากับค่าของ*นิพจน์*(ในวงเล็บหลังคำสำคัญ switch) ข้อความสั่งที่ตามมาของกรณีนั้นก็จะถูกดำเนินการ

จากนั้นมันจะเลื่อนลงไปทำข้อความสั่งของ `case` ถัดไปเรื่อยๆจนหมดทุก `case` โดยที่ไม่ได้เปรียบเทียบค่ากับค่าคงที่ของ `case` ใดๆอีกเลย

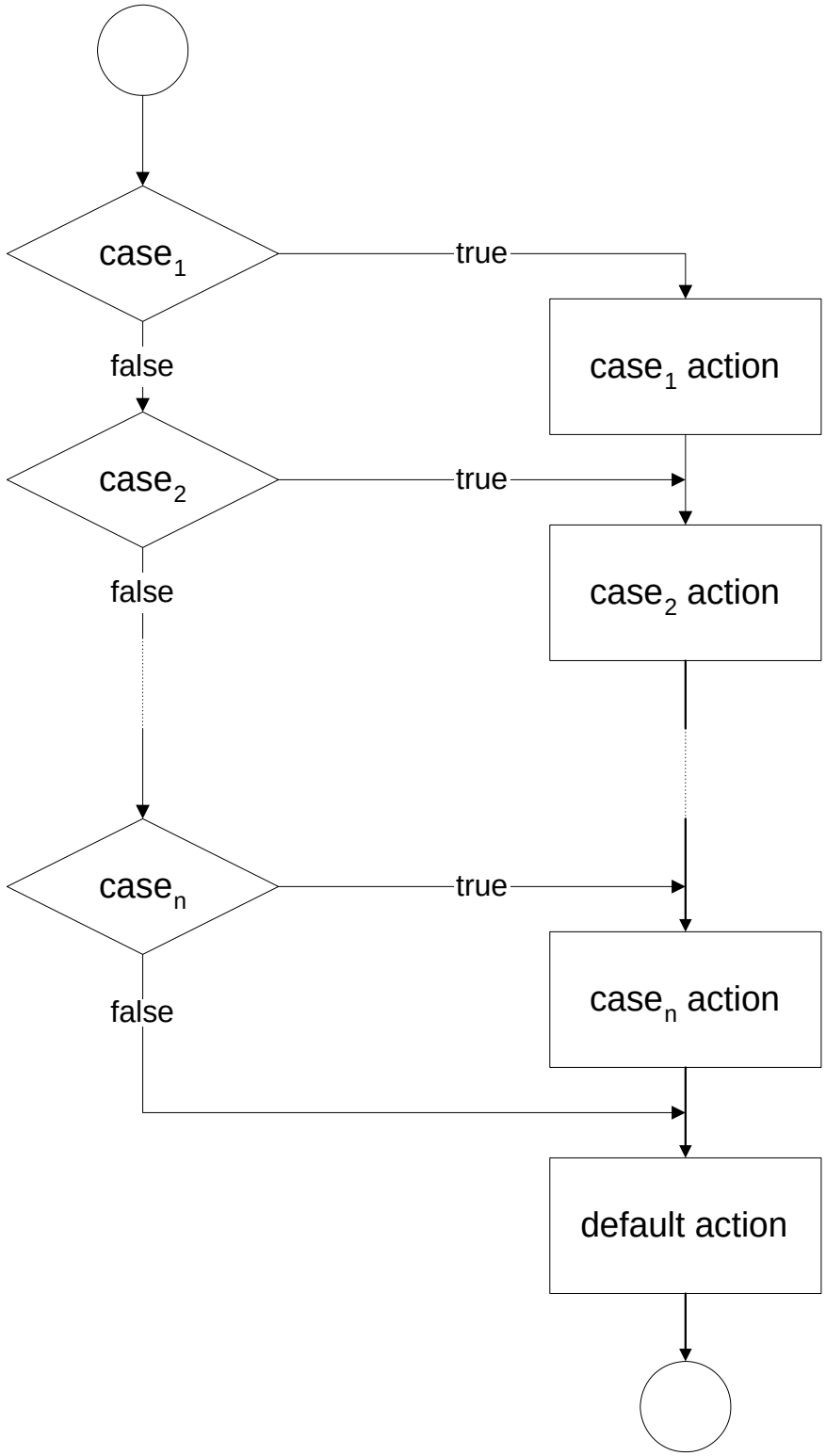
กรณี `default` จะมีหรือไม่มีก็ได้ ถ้ามี ข้อความสั่งหลัง `default` จะทำงานถ้าค่าของ*นิพจน์*ไม่เท่ากับ*นิพจน์ค่าคงที่*ของ `case` ใดเลย แต่ถ้ามีค่าคงที่ของ `case` ใด `case` หนึ่งเท่าขึ้นมา ข้อความสั่งหลัง `default` จะไม่ทำงาน ถ้าไม่มีการใส่คำ `default` ไว้ในข้อความสั่ง `switch` เมื่อมีการเปรียบเทียบค่ามาจนถึง `case` สุดท้ายแล้วยังไม่พบค่าที่เท่ากันเลย ก็จะออกจากข้อความสั่ง `switch` ไปโดยไม่ทำข้อความสั่งใดๆใน `switch` เลย

ข้อความสั่ง `switch` เทียบได้กับการใช้ข้อความสั่ง `if` และ `goto`-*label* ดังนี้

```
if (นิพจน์ == นิพจน์ค่าคงที่-1)
        goto case-1;
if (นิพจน์ == นิพจน์ค่าคงที่-2)
        goto case-2;
...
if (นิพจน์ == นิพจน์ค่าคงที่-n)
        goto case-n;
goto default;
case-1: กลุ่มข้อความสั่ง
case-2: กลุ่มข้อความสั่ง
...
case-n: กลุ่มข้อความสั่ง
default: กลุ่มข้อความสั่ง
```

หมายเหตุ `goto` เป็นคำสำคัญสงวน (reserved keyword) ในภาษาจาวา แต่ไม่มีการนำคำนี้ไปใช้เป็นข้อความสั่ง ถึงแม้ไม่มีข้อความสั่ง `goto` ให้ใช้ แต่การนำ `goto` มาช่วยในการอธิบายการทำงานของข้อความสั่ง `switch` ทำให้ทำความเข้าใจข้อความสั่ง switch ได้ง่ายขึ้น ข้อความสั่ง `goto` ที่ใช้อธิบายนี้ ใช้แทนความหมายของการเปลี่ยนทิศทางการไหลของโปรแกรม เมื่อโปรแกรมได้รับคำสั่ง `goto` โปรแกรมจะข้ามไปทำคำสั่งหลังตำแหน่งในโปรแกรมที่ตั้งชื่อเรียกที่ระบุไว้หลังข้อความสั่ง `goto` เช่นข้อความสั่ง `goto case-1` หมายถึงให้ข้ามไปทำคำสั่งที่อยู่หลังป้ายบอกตำแหน่ง `case-1`:

กระบวนการเลือกของข้อความสั่ง `switch` แสดงด้วยผังงานได้ดังรูปที่ 1



รูปที่ 1 ผังงานแสดงการทำงานของข้อความสั่ง `switch`

ตัวอย่างที่ 1 แสดงโปรแกรมซึ่งใช้ข้อความสั่ง `switch` ในการเลือกทำงาน โดยรับค่าตัวเลขเข้ามาทางแผงแป้นอักขระ(keyboard) แล้วเข้าสู่ข้อความสั่ง `switch` เพื่อเลือกว่าจะไปเริ่มดำเนินการที่ข้อความสั่งใด

ตัวอย่างที่ 1 โปรแกรมที่ใช้ switch ในการเลือกทางเลือกที่มีหลายกรณี

```
1:   class Switch1 {
2:        public static void main(String args[]) {
3:             int choice;
4:             choice=1;
5:             switch ( choice )
6:             {
7:                  case 1 : System.out.println("do case-1");
8:                  case 2 : System.out.println("do case-2");
9:                  case 3 : System.out.println("do case-3");
10:                 default: System.out.println("do default case");
11:            }
12:       }
13:  }
```

จากตัวอย่างโปรแกรมที่ 1 ถ้ากำหนดให้ choice มีค่าเป็น 1 จะพิมพ์ผลลัพธ์ต่อไปนี้ออกมา

```
do case-1
do case-2
do case-3
do default case
```

ถ้า choice มีค่าเป็น 2 ได้ผลลัพธ์ดังนี้

```
do case-2
do case-3
do default case
```

ถ้า choice มีค่าเป็น 3 จะได้ผลลัพธ์ดังนี้

```
do case-3
do default case
```

ถ้า choice มีค่าอื่นใดที่ไม่ใช่ 1 ถึง 3 จะได้ผลลัพธ์ดังนี้

```
do default case
```

จะเห็นว่าเมื่อเข้าข่ายที่ทำข้อความสั่งของ case ใดแล้ว ข้อความสั่งของ case ถัดไปก็จะถูกดำเนินการด้วย เนื่องจากหัวข้อ case เป็นเพียงป้าย(label)บอกตำแหน่งเท่านั้น มิได้มีการตรวจสอบค่าหรือตัดสินใจแต่อย่างใด ลักษณะการเลื่อนไหลต่ำลงมาดำเนินงานข้อความสั่งในลำดับถัดลงมาเรียกว่า falls through

นิพจน์ค่าคงที่ของกรณีต่างๆใน switch ไม่จำเป็นต้องเขียนเรียงตามลำดับจากน้อยไปหามาก จะให้ค่าใดขึ้นก่อนก็ได้ รวมทั้งกรณี default ด้วยไม่จำเป็นต้องอยู่ล่างสุดของข้อความสั่ง switch เสมอไป อาจอยู่ตำแหน่งเหนือขึ้นไปก่อน case บาง case ก็ได้

ในการแก้ปัญหาโจทย์โดยทั่วไปเรามักไม่ต้องการให้ไปดำเนินงานข้อความสั่งของ case อื่นๆนอกเหนือจาก case ที่จับคู่กันได้เท่านั้น ดังนั้นหลังจากเสร็จสิ้นการดำเนินงานข้อความสั่งของ case ที่จับคู่ได้แล้วจึงมักต้องการที่จะออกจากข้อความสั่ง switch ไปเลย ซึ่งสามารถทำได้โดยใส่ข้อความสั่ง break เข้าไปด้วย ดัง

ตัวอย่างที่ 2 โดยปกติแล้วเรามักใช้ข้อความสั่ง `break` ควบคู่กับข้อความสั่ง `switch` ดังนั้นรูปแบบของการใช้ `switch` ที่พบเห็นโดยทั่วไปจึงเขียนเป็นผังงานได้รูปที่ 2

ตัวอย่างที่ 2 ใช้ข้อความสั่ง `switch` ควบคู่กับ `break`

```
1:   import java.util.Scanner;
2:   class Switch2 {
3:        public static void main(String args[]){
4:             int marks;
5:             char grade;
6:             Scanner sn = new Scanner(System.in);
7:             System.out.print("Enter marks between 0 - 100\n");
8:             marks = sn.nextInt(); // input marks from keyboard
9:             switch ( marks/10 )
10:            {
11:                 case 10:
12:                 case  9:
13:                           grade = 'A';
14:                            break;
15:                 case  8:
16:                           grade = 'B';
17:                            break;
18:                case  7:
19:                          grade = 'C';
20:                           break;
21:                 case  6:
22:                           grade = 'D';
23:                            break;
24:                 default:
25:                           grade = 'F';
26:                            break;
27:            }
28:            System.out.println("Grade is " + grade );
29:       }
30:  }
```

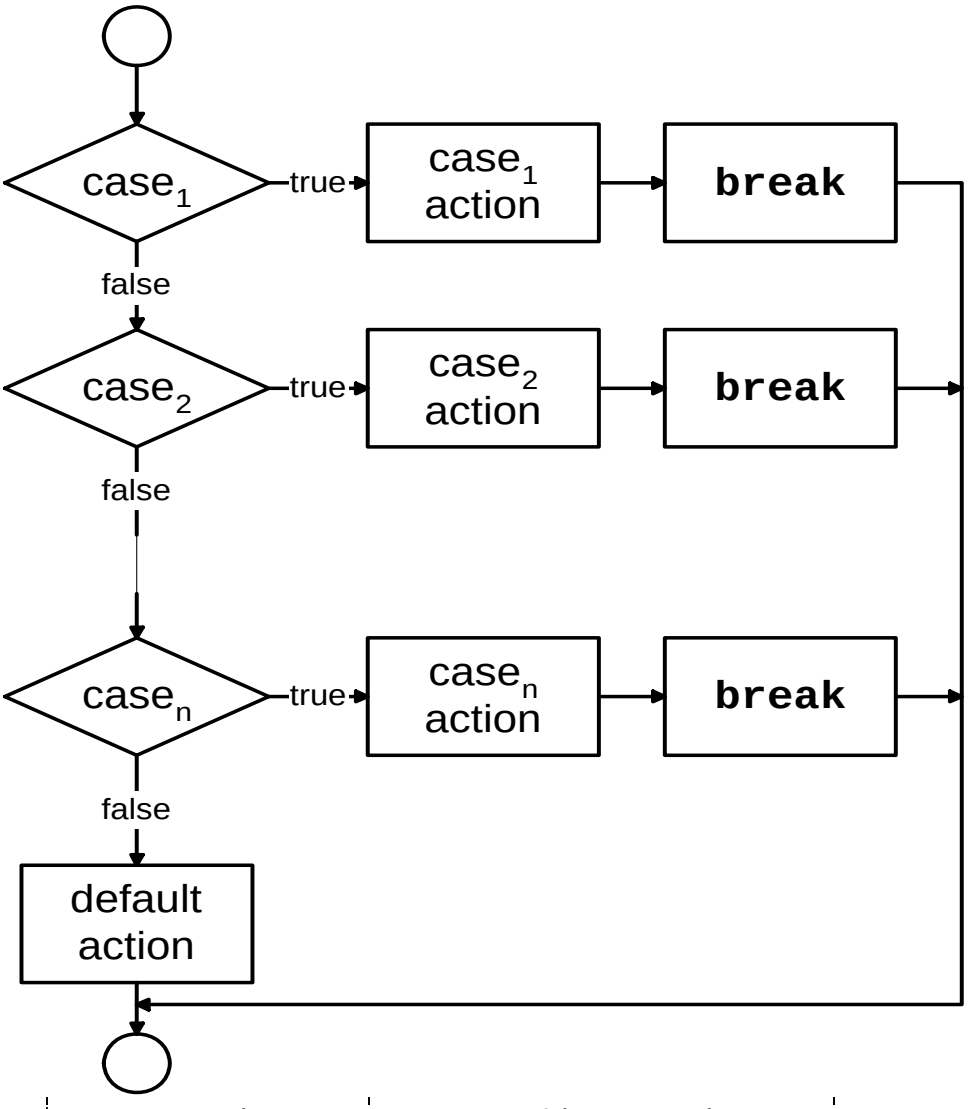
ตัวอย่างนี้มีการอ่านข้อมูลทางแผงแป้นพิมพ์เป็นค่าตัวเลขคะแนนระหว่าง 0 ถึง 100 แล้วแปลงคะแนนเป็นเกรด A - F เนื่องจากโปรแกรมนี้มีการอ่านค่าข้อมูลทางแผงแป้นอักขระ จึงต้องมีข้อความสั่ง `import java.Scanner` ในบรรทัดที่ 1 เพื่อบอกคอมไพเลอร์ว่าเราต้องการนำคลาส Scanner มาใช้ในการอ่านข้อมูล บรรทัดที่ 8 ข้อความ `marks = sn.nextInt()` ใช้สำหรับอ่านค่าตัวเลขทางแผงแป้นพิมพ์แล้วนำไปเก็บไว้ที่ตัวแปร `marks` ซึ่ง `nextInt()` เป็นเมธอดที่ใช้ในการรับข้อมูลทางแผงแป้นอักขระซึ่งรับเข้ามาเป็นอักขระ(ถึงแม้จะป้อนตัวเลขเข้ามาทางแผงแป้นอักขระก็ตาม ข้อมูลที่รับเข้ามาเป็นรหัสของอักขระอยู่ดีซึ่งไม่สามารถนำไปคำนวณได้ในทันที ต้องแปลงให้เป็นค่าเลขจำนวนเต็มเสียก่อนจึงจะนำไปคำนวณได้) แล้วแปลงให้เป็นค่าตัวเลขจำนวนเต็ม ดังนั้นถ้าผู้ใช้โปรแกรมป้อนข้อมูลเป็นตัวเลขเข้ามาทางแผงแป้นพิมพ์ จะได้ค่าข้อมูลในเก็บไว้ในตัวแปร `marks` แต่ถ้าป้อนข้อมูลที่ไม่ใช้ตัวเลขจะเกิดสิ่งผิดปรกติขึ้น

เมื่อได้ข้อมูลมาแล้วจะนำไปใช้ต่อไปในข้อความสั่ง `switch`

บรรทัดที่ 9 มีการใช้ข้อความสั่ง `switch` นิพจน์ของ `switch` ในตัวอย่างนี้คือ `marks/10` ซึ่งจะหารคะแนนที่รับเข้ามาด้วย 10 เพื่อลดจำนวน `case` ที่จะเปรียบเทียบลง ตารางต่อไปนี้เปรียบเทียบค่าของคะแนนที่รับเข้ามากับผลลัพธ์ที่ได้นิพจน์ของ `switch`

marks	ผลของนิพจน์
100	10
90 - 99	9
80 - 89	8
70 - 79	7
60 - 69	6
50 - 59	5
:	:
0	0

ตัวอย่างนี้แสดงให้เห็นข้อความสั่งที่มีหลายป้าย (multiple labels) ประโยค `grade = 'A'` ใช้ป้าย `case` 10 และ `case` 9 ร่วมกัน ถ้านิพจน์ `marks/10` มีค่าเป็น `10` หรือมีค่าเป็น 9 ต่างก็จะไปกระทำการที่ประโยคข้อความสั่งเดียวกันทั้งคู่ `case` แรกไม่มีข้อความสั่งกำหนดไว้ ถ้าเข้า `case` นี้เมื่อใดจะไปทำข้อความสั่งของ `case` ที่สอง เมื่อทำข้อความสั่งใน `case` ที่สองแล้วเจอข้อความสั่ง `break` ก็จะออกจากข้อความสั่ง `switch` ไป กรณี `default` จะดำเนินงานก็ต่อเมื่อคะแนนที่รับเข้ามามีค่าต่ำกว่า `60` ข้อความสั่ง `break` ในกรณี `default` จะไม่ใส่ไว้ก็ได้เนื่องจากไม่มีข้อความสั่งอื่นใดให้ดำเนินงานหลังจากนี้อีกแล้ว แต่การใส่ไว้จะช่วยให้โปรแกรมมีความชัดเจน และป้องกันการหลงลืมเมื่อมีการย้ายกลุ่มข้อความสั่งของกรณี `default` ไปไว้ยังตำแหน่งอื่นเหนือขึ้นไป หรือมีกรณีใหม่เข้ามาแทนที่ `default` เดิม



รูปที่ 2 ผังงานของข้อความสั่ง `switch` ใช้ควบคู่กับข้อความสั่ง `break`

นิพจน์ของ `switch` ไม่จำเป็นต้องมีผลลัพธ์เป็นค่าตัวเลขจำนวนเต็มเท่านั้นอาจเป็นอักขระก็ได้ดังตัวอย่างที่ 3 ซึ่งในกรณีเช่นนี้นิพจน์อักขระจะถูกแปลงให้เป็นนิพจน์จำนวนเต็มตามหลักการเลื่อนระดับ(promotion) ตัวอย่างนี้ตัวแปร `grade` มีประเภทเป็น `char` ใช้สำหรับเก็บข้อมูลที่รับเข้ามาทางแผงแป้นอักขระโดยใช้ข้อความสั่ง `System.in.read()` ซึ่งอาจทำให้เกิดสิ่งผิดปรกติชนิด `IOException` ได้ จึงเป็นต้องระบุ `throws IOException` ไว้ท้ายหัวเมธอด `main` ในบรรทัดที่ 3 (รายละเอียดกล่าวถึงในบทที่ ๘ เรื่องการจัดการสิ่งผิดปรกติ) ส่วนบรรทัดที่ 11 และ 37 ต้องมีการหล่อหลอม (casting) ค่าที่ได้จาก `System.in.read()` ซึ่งให้ค่าเป็นจำนวนเต็มไม่ใช่อักขระให้เป็นประเภทอักขระเสียก่อนที่จะนำไปเก็บในตัวแปร `grade` ทั้งๆ ที่ข้อมูลที่รับเข้าทางแผงแป้นอักขระเป็นประเภทอักขระอยู่แล้ว แต่เพื่อความสะดวกในการใช้งานจึงให้ค่าเป็นประเภทเลขจำนวนเต็มแทนเนื่องจากนำไปใช้ได้กว้างขวางกว่า เพราะว่าการคำนวณ หรือการหาค่าในนิพจน์ส่วนใหญ่ของจาวาจะหาค่าแบบเลขจำนวนเต็ม ซึ่งหากในตัวอย่างนี้เรากำหนดให้ `grade` มีประเภทเป็น `int` เสียเลยก็ไม่จำเป็นต้อง cast ส่วนการหาค่าในนิพจน์ของข้อความสั่ง `switch` นั้นทำงานแบบเลขจำนวนเต็มอยู่แล้ว จึงสะดวกด้วยประการทั้งปวง แต่วัตถุประสงค์ของตัวอย่างนี้ต้องการพิสูจน์ให้เห็นว่า ข้อความสั่ง `switch` สามารถรับนิพจน์ที่ให้ผลลัพธ์เป็นอักขระได้ อีก

อย่างคือเราต้องการเปรียบเทียบค่าตัวอักขระอยู่แล้ว การกำหนดให้ตัวแปร grade เป็นประเภท char ช่วยให้โปรแกรมมีความชัดเจนและทำความเข้าใจได้ง่ายขึ้น

ตัวอย่างที่ 3 การใช้ข้อความสั่ง switch กับป้ายกรณีที่เป็นนิพจน์อักขระ

```
1:   import java.io.*;
2:   class Switch3 {
3:        public static void main(String args[]) throws IOException{
4:             char grade;
5:             int aCount = 0, bCount = 0, cCount = 0,
6:              dCount = 0, fCount = 0;
7:
8:             System.out.println("Enter the letter grades");
9:             System.out.println("Enter X to end input");
10:
11:            grade = (char)System.in.read();
12:            while ( grade != 'X' && grade != 'x' )
13:            {
14:                switch (grade)
15:                {
16:                   case 'A' :
17:                   case 'a' :  ++aCount;
18:                               break;
19:                   case 'B' :
20:                   case 'b' :  ++bCount;
21:                               break;
22:                   case 'C' :
23:                   case 'c' :  ++cCount;
24:                               break;
25:                   case 'D' :
26:                   case 'd' :  ++dCount;
27:                               break;
28:                   case 'F' :
29:                   case 'f' :  ++fCount;
30:                               break;
31:                    case '\n':  break;  // ignore Enter key
32:                   default:    // catch all other characters
33:                       System.out.println("Incorrect letter grade.");
34:                       System.out.println(" Enter a new grade.");
35:                       break;
36:                }
37:                grade = (char)System.in.read();   // input grade again
38:            }
39:            System.out.println("\nTotals for each letter grade are:");
40:            System.out.println("A: " + aCount);
41:            System.out.println("B: " + bCount);
42:            System.out.println("C: " + cCount);
43:            System.out.println("D: " + dCount);
44:            System.out.println("F: " + fCount);
45:       }
46:  }
```