ชนิดข้อมูล ตัวแปร และตัวดำเนินการ

ชนิดข้อมูล (Data Type)

ในการประมวลผลด้วยคอมพิวเตอร์นั้น จะมีการนำข้อมูลมาคำนวณหรือประมวลผลเพื่อให้ได้ผลลัพธ์ตามที่ต้องการ ข้อมูลที่จะนำมาประมวลผลนั้นจะต้องจัดเก็บไว้ในหน่วยความจำก่อนที่จะนำไปประมวลผลในหน่วยประมวลผลกลาง (CPU) ข้อมูลที่จัดเก็บในหน่วยความจำมีหลากหลายรูปแบบ ภาษาโปรแกรมแต่ละภาษาจะมีการกำหนดชนิดของข้อมูลที่สามารถใช้ได้ในภาษานั้นๆ ขึ้นมาแตกต่างกันไปตามความเหมาะสมในการใช้งานของแต่ละภาษา สำหรับภาษาจาวาซึ่งหยิบยืมไวยากรณ์ส่วนหนึ่งมาจากภาษาซี ได้รับเอาชนิดของข้อมูลที่มีใช้ในภาษาซีมาใช้เพียงบางส่วน ชนิดของข้อมูลที่ใช้ไม่บ่อยนักถูกตัดออกไปเพื่อให้ง่ายต่อการเรียนรู้และใช้ภาษา ข้อมูลบางชนิดที่มีใช้ในภาษาซีที่เข้าใจยากและอาจเกิดปัญหาในการใช้ได้ง่ายได้ถูกกำหนดให้เป็นชนิดข้อมูลแบบใหม่ที่ทำให้เข้าใจง่ายกว่าและป้องกันปัญหาที่อาจเกิดขึ้นจากการใช้ข้อมูลผิดพลาดหรือใช้ผิดประเภท

จาวาแบ่งการจัดเก็บและประมวลผลข้อมูลเป็น 2 กลุ่ม คือข้อมูลพื้นฐาน (primitive type) และข้อมูลอ้างอิง (reference type) ในบทนี้จะกล่าวถึงเฉพาะข้อมูลพื้นฐาน ซึ่งประกอบด้วยข้อมูลทางตรรกะ (Boolean type) อักขระ (Character type) และข้อมูลตัวเลข(Numerical types) ซึ่งแบ่งย่อยออกเป็นอีกหลายชนิด

ชนิดของข้อมูลจะมีผลต่อรูปแบบในการจัดเก็บข้อมูลในหน่วยความจำและการกระทำหรือการดำเนินการต่างๆ ที่สามารถกระทำได้กับข้อมูลชนิดนั้นๆ เช่นข้อมูลชนิดตัวเลขสามารถนำไปคำนวณได้ ในขณะที่ข้อมูลทางตรรกะนำไปคำนวณไม่ได้แต่สามารถนำไปดำเนินการทางตรรกะได้ รูปแบบในการจัดเก็บข้อมูลจะมีผลต่อค่าของข้อมูลที่สามารถจัดเก็บได้

ภาษาจาวาได้กำหนดชื่อชนิด ขนาด และค่าที่สามารถใช้ได้ของข้อมูลพื้นฐานชนิดต่างๆ ไว้ดังนี้

ชนิด	ข้อมูลที่เก็บ	ขนาด (ไบต์)	ค่าที่สามารถใช้ได้
`boolean`	ข้อมูลตรรกะ		`true` และ `false`
`char`	ตัวอักขระ	2	ค่าอักขระตามมาตรฐาน Unicode
`byte`	เลขจำนวนเต็ม	1	-128 ถึง 127
`short`	เลขจำนวนเต็ม	2	-32768 ถึง 32767
`int`	เลขจำนวนเต็ม	4	-2147483648 ถึง 2147483647
`long`	เลขจำนวนเต็ม	8	-9223372036854775808 ถึง 9223372036854775807
`float`	เลขทศนิยมลอยตัว	4	-3.40282347E+38 [1]ถึง 3.40282347E+38
`double`	เลขทศนิยมลอยตัว	8	-1.79769313486231570E+308 ถึง 1.79769313486231570E+308

[1] เป็นการเขียนค่าข้อมูลแบบทศนิยมลอยตัวตามแบบวิทยาศาสตร์ เช่น 1.234567E+32 หมายถึง 1.234567×10^{32}

ข้อมูลตรรกะ

ข้อมูลชนิด boolean เป็นข้อมูลตรรกะซึ่งมีค่าที่เป็นไปได้ ๒ ค่าคือจริง(`true`) และ*เท็จ*(`false`) ข้อมูลชนิดนี้สามารถนำไปใช้ในการดำเนินการทางตรรกะเช่น แอนด์(AND) และ ออร์(OR) เป็นต้น เรามักนำข้อมูลชนิดนี้ไปใช้เป็นส่วนหนึ่งในรูปประโยคของข้อความสั่งสำหรับตรวจสอบเงื่อนไข หรือทำซ้ำ นอกจากนี้เราอาจใช้ข้อมูลชนิดนี้แทนค่าข้อมูลอื่นที่มีค่า ๒ ค่าได้เช่นกัน เช่นแทนข้อมูลที่มีความหมายว่า มีหรือไม่มี ใช่หรือไม่ใช่ และชายหรือหญิง เป็นต้น

ตัวอักขระ

ข้อมูลชนิด char เป็นข้อมูลอักขระ ซึ่งโดยทั่วไปแล้วใช้แทนตัวอักษร สัญลักษณ์ ตัวเลขและรหัสควบคุม จาวาจัดเก็บข้อมูลอักขระต่างจากภาษาอื่นซึ่งโดยทั่วไปแล้วจะใช้ข้อมูลขนาด ๑ ไบต์ในการเก็บตัวอักขระ ๑ ตัว ภาษาจาวาถูกออกแบบมาให้สนับสนุนการใช้อักขระภาษาต่างๆ ได้หลายภาษา โดยใช้มาตรฐาน ยูนิโค๊ด (Unicode) ในการจัดเก็บข้อมูลอักขระ แทนที่จะใช้มาตรฐานแอสกี้ (ASCII) ซึ่งรองรับตัวอักขระได้ไม่เกิน ๒๕๖ ตัว ยูนิโค๊ดใช้ ๑๖ บิตหรือ ๒ ไบต์ในการจัดเก็บข้อมูล ซึ่งสามารถรองรับจำนวนอักขระได้นับหมื่นตัว จึงสามารถรองรับตัวอักขระภาษาต่างๆ ได้เป็นจำนวนมาก รวมทั้งอักขระตามมาตรฐาน ASCII ก็บรรจุอยู่ในมาตรฐานยูนิโค๊ดด้วยโดยเติมบิตศูนย์ข้างหน้าของรหัสแต่ละตัวอักขระของแอสกี้จนครบ 16 บิต

เลขจำนวนเต็ม

เลขจำนวนเต็มเป็นข้อมูลตัวเลขแบบไม่มีจุดทศนิยมสามารถนำไปใช้ในการคำนวณทางคณิตศาสตร์ต่างๆ ได้ คอมพิวเตอร์สามารถเก็บข้อมูลชนิดนี้ได้อย่างมีประสิทธิภาพและสามารถคำนวณได้เร็ว

เลขทศนิยมลอยตัว

เป็นข้อมูลตัวเลขซึ่งสามารถเก็บเศษทศนิยมได้ด้วย โดยไม่ต้องกำหนดว่าจะเก็บทศนิยมกี่ตำแหน่ง ระบบจะปรับตำแหน่งทศนิยมให้เหมาะสมเองโดยอัตโนมัติ ข้อมูลประเภทนี้สามารถรองรับการเก็บค่าของข้อมูลเป็นช่วงที่กว้างมาก เนื่องจากแบ่งบิตจำนวนหนึ่งไปเก็บเป็นเลขยกกำลัง จึงทำให้เก็บค่าตัวเลขสูงหรือต่ำมากๆได้ แต่จำเป็นต้องลดความละเอียดแม่นยำในการจัดเก็บลง ซึ่งจะทำให้ข้อมูลที่จัดเก็บคลาดเคลื่อนไปบ้าง จึงไม่นิยมใช้ในการเก็บข้อมูลทางธุรกิจที่ต้องการความแม่นยำสูง แต่นิยมใช้ในการเก็บข้อมูลทางวิทยาศาสตร์และวิศวกรรม โดยปรกติแล้วคอมพิวเตอร์มักจะคำนวณเลขแบบทศนิยมลอยตัวได้ช้ากว่าเลขจำนวนเต็ม

การประกาศตัวแปร

การประมวลผลด้วยคอมพิวเตอร์จำเป็นต้องมีการจัดเก็บข้อมูลในหน่วยความจำก่อนประมวลผลเสมอ ข้อมูลที่จัดเก็บในหน่วยความจำจะมีเลขที่อยู่ (address) ของมัน ซึ่งเป็นตัวบอกตำแหน่งที่เก็บข้อมูลในหน่วยความจำ หน่วยประมวลผลกลาง (CPU) จะใช้เลขที่อยู่ของข้อมูลสำหรับการอ้างอิงในการเข้าถึงข้อมูลนั้นๆ การใช้ภาษาโปรแกรมขั้นสูงผู้เขียนโปรแกรมไม่จำเป็นต้องทราบเลขที่อยู่ของข้อมูลในหน่วยความจำเนื่องจากผู้ออกแบบภาษาได้ออกแบบภาษาให้ผู้เขียนโปรแกรมตั้งชื่อเรียกข้อมูลแทนการใช้เลขที่อยู่ของข้อมูลเพื่อความง่ายและสะดวกในการเขียนโปรแกรม ภาษาขั้นสูงต่างๆ อนุญาตให้มีการตั้งชื่อเรียกตำแหน่งของข้อมูล โดยเรียกเป็นชื่อ*ตัวแปร (variable)* หรือ

ชื่อข้อมูล (data name) แทนการบอกเลขที่อยู่ของมัน คอมไพเลอร์จะทำหน้าที่แปลงชื่อตัวแปรหรือชื่อข้อมูลไปเป็นเลขที่อยู่ข้อมูลในขั้นตอนของการแปลโปรแกรม (compile) ตำแหน่งที่จัดเก็บข้อมูลในหน่วยความจำในภาษาจาวาเรียกว่า *ตัวแปร* (variable)

ภาษาจาวากำหนดให้มีการประกาศตัวแปรก่อนที่จะมีการนำตัวแปรนั้นไปใช้ การประกาศตัวแปรเป็นการบอกว่าผู้เขียนโปรแกรมต้องการให้จัดเตรียมเนื้อที่ในหน่วยความจำสำหรับเก็บข้อมูลชนิดใด และตั้งชื่อเรียกข้อมูลนั้นว่าอะไร การประกาศตัวแปรในภาษาจาวาจะเริ่มด้วยชนิดของตัวแปร แล้วตามด้วยชื่อตัวแปร ตามรูปแบบดังนี้

ชนิดของตัวแปร ***ชื่อตัวแปร*** [= ***ค่าเริ่มต้นของข้อมูล***]

- **ชนิดของตัวแปร** เป็นชื่อประเภทของตัวแปร เป็นคำสำคัญ (keyword) ที่กำหนดไว้ในภาษาจาวา
- ***ชื่อตัวแปร*** เป็นชื่อที่ผู้เขียนโปรแกรมสามารถตั้งได้เอง แต่ต้องเป็นไปตามกฎการตั้งชื่อตัวระบุ
- **[= *ค่าเริ่มต้นของข้อมูล*]** เป็นตัวเลือกที่จะใส่หรือไม่ก็ได้ ใช้ในกรณีที่ต้องการกำหนดค่าเริ่มต้นให้กับตัวแปร

ตัวอย่างการประกาศตัวแปร

```
byte    nFaculties;        // ประกาศให้ตัวแปร nFaculties เป็นชนิด byte
int     salary;            // ประกาศตัวแปรชื่อว่า salary มีชนิดเป็น int
double distance;           // ประกาศตัวแปรชนิด double ให้ชื่อว่า distance
boolean isNum = false;     // ประกาศตัวแปรชนิดบูล ให้ชื่อว่า isNum มีค่าเริ่มต้นเป็น false
byte   nMonths = 12;       //  ประกาศตัวแปรชื่อ nMonths เป็นชนิด byte ให้มีค่าเริ่มต้นเป็น 12
char   ch = 'a';           // ประกาศตัวแปรชนิด char ชื่อ ch มีค่าเริ่มต้นเป็นอักขระ 'a'
short  qty = 12345;        // ประกาศตัวแปรชื่อ qty เป็นชนิด short มีค่าเริ่มต้นเป็น  12345
double y = 12345.678;      // ประกาศตัวแปรชนิด double ให้มีชื่อเป็น y มีค่าเริ่มต้นเป็น 12345.678
float  iRate = 0.0025f;// ประกาศตัวแปรชนิด float มีชื่อว่า iRate มีค่าเริ่มต้นเป็น 0.0025
```

ค่าที่กำหนดให้กับตัวแปรต้องเป็นชนิดเดียวกับชนิดของตัวแปรหรือถ้าเป็นต่างชนิดกันต้องเป็นชนิดที่สามารถเข้ากันได้โดยไม่ทำให้เมื่อนำข้อมูลไปใส่ในตัวแปรแล้วทำให้เกิดการสูญเสียหรือค่าผิดเพี้ยนไป

ต่อไปนี้เป็นตัวอย่างการกำหนดค่าให้กับตัวแปรที่ไม่ถูกต้อง

```
byte    b = 1000;       // b เป็นตัวแปรชนิด byte สามารถรองรับข้อมูลค่าสูงสุดเพียง 127
long    big = 1234567.89; // big เป็นตัวแปรชนิด long เก็บเลขจำนวนเต็มรับค่าแบบมีทศนิยมไม่ได้
float   f = 123.45;     //  f เป็นชนิด float แต่ 123.45 เป็นชนิด double ซึ่งใหญ่กว่า float
                        //  คอมไพเลอร์ไม่ยอมให้นำชนิดของข้อมูลที่ใหญ่ไปใส่ในตัวแปรที่เล็กกว่า
                        // กรณีนี้อาจแก้ไขโดยกำหนดให้ใส่ suffix f ให้กับค่าข้อมูลเป็น 123.45f
                        //  จะทำให้ชนิดของค่าของข้อมูล กับชนิดของตัวแปรเป็นชนิดเดียวกัน
```

เราสามารถประกาศตัวแปรชนิดเดียวกันหลายตัวพร้อมกันได้โดยบอกชนิดเพียงครั้งเดียว และใช้จุลภาคคั่นตัวแปรแต่ละตัวได้ดังตัวอย่างต่อไปนี้

```
int x,y; // ประกาศตัวแปร x และ y เป็นชนิด double
double a=1.2, b=2.4, c, d=5.6; // ประกาศตัวแปร a b c และ d เป็นชนิด double พร้อมทั้งกำหนดค่าเริ่มต้นให้ ยกเว้น c ที่ไม่ได้กำหนดค่าให้
```

นิพจน์

นิพจน์(expression)คือการก่อรูปของ**ค่าข้อมูล**(literal) และ/หรือ **ตัวแปร** และ/หรือ **การเรียกเมธอด**(method call หรือ method invocation) เพียงตัวใดตัวหนึ่งเพียงตัวเดียวหรือหรือหลายตัวผสมกันก็ได้ หากมีหลายตัวผสมกันจะใช้ตัวดำเนินการ (operator) เป็นตัวเชื่อม การก่อรูปนี้จะมีการประเมินแล้วให้ผลลัพธ์หนึ่งค่า ซึ่งสามารถนำไปจัดเก็บไว้ในตัวแปร หรือนำค่าที่ได้นี้ไปใช้ในข้อความสั่งได้โดยตรงในทันที หรือจะไม่ใช้ค่าที่ได้นั้นเลยก็ได้

ตัวอย่างเช่น

```
x
9
Math.random()
```

ตัวอย่างทั้ง ๓ บรรทัดข้างต้นนี้ถือว่าเป็นนิพจน์ซึ่งไม่มีตัวดำเนินการเข้ามาเกี่ยวข้อง ตัวอย่างในบรรทัดที่ ๑ `x` เป็นตัวแปร ตัวอย่างบรรทัดที่ ๒ 9 เป็นค่าคงที่ ตัวอย่างบรรทัดที่ ๓ `Math.random()` เป็นการเรียก*เมธอด* การเรียกเมธอดจะประกอบด้วยชื่อเมธอดตามด้วยเครื่องหมายวงเล็บ ภายในวงเล็บอาจประกอบด้วยนิพจน์หรือไม่ก็ได้ขึ้นอยู่กับว่าเมธอดที่เรียกใช้นั้นต้องการให้ส่งข้อมูลไปให้ในขณะเรียกเมธอดหรือไม่

นิพจน์อาจมีความซับซ้อนมากขึ้นเช่นมีตัวแปรหลายตัว มีค่าคงที่ หรือมีการเรียกเมธอดเข้ามาเกี่ยวข้องกันหลายตัว ในกรณีนี้จำเป็นต้องใช้ตัวดำเนินการเป็นตัวเชื่อม ตัวอย่างเช่น

```
x + y
x * 5 + y
Math.random() / 99
```

นิพจน์ในภาษาจาวามีหลายชนิด ในเบื้องต้นนี้จะกล่าวถึงเฉพาะนิพจน์ทางคณิตศาสตร์ ซึ่งมีวิธีการเขียนคล้ายคลึงกับนิพจน์ที่ใช้ในวิชาคณิตศาสตร์ จะมีต่างกันอยู่บ้างก็ตรงที่การคูณ จะต้องใช้เครื่องหมายดอกจันทร์(*) เป็นสัญญลักษณ์แทนการคูณเสมอ ดังนั้น ถ้าต้องการให้ a คูณกับ b จะต้องเขียนเป็น a * b จะเขียนเพียงแค่ ab ไม่ได้

ตัวดำเนินการ

ตัวดำเนินการ(operator)คือสัญญลักษณ์ที่ใช้บ่งบอกการกระทำที่จะกระทำต่อข้อมูล การกระทำต่างๆ จะมีข้อมูลบางตัวที่ถูกกระทำเรียกว่าตัวถูกดำเนินการ ตัวถูกดำเนินการ(operand) คือค่าหรือตัวแปรที่ถูกกระทำ หรือถูกใช้ในการคำนวณ

ตัวดำเนินการแบ่งตามจำนวนตัวถูกดำเนินการเป็น ๓ ชนิดคือ

ตัวดำเนินการเอกภาค (Unary Operators) : มีตัวถูกดำเนินการเพียงตัวเดียว

ตัวดำเนินการทวิภาค (Binary Operators) : มีตัวถูกดำเนินการ ๒ ตัว

ตัวดำเนินการไตรภาค (Ternary Operator) : มีตัวถูกดำเนินการ ๓ ตัว

ภาษาจาวาใช้อัขระพิเศษเป็นสัญญลักษณ์แทนตัวดำเนินการ เช่นตัวอักขระดอกจันทร์ (*****) เป็นสัญลักษณ์แทนการคูณ ตัวดำเนินการอาจแทนด้วยอักขระจำนวน ๑ หรือ ๒ ตัวแล้วแต่ชนิดของตัวดำเนินการ ตัวดำเนินการใดๆ ที่ใช้อักขระ ๒ ตัวประกอบกันจะต้องเขียนติดกันห้ามเว้นวรรคหรือมี *พื้นที่สีขาว* (white space) คั่นกลาง

ตัวดำเนินการคำนวณ

ตัวดำเนินการคำนวณ(Arithmetic Operators)เป็นตัวดำเนินการทวิภาค จะมีตัวถูกดำเนินการ ๒ ตัวคั่นด้วยตัวดำเนินการคำนวณซึ่งเป็นสัญญลักษณ์ที่ใช้แทนการคำนวณทางคณิตศาสตร์ ตารางต่อไปนี้แสดงเครื่องหมายตัวดำเนินการคำนวณที่ใช้ในภาษาจาวา โดยเรียงตามการมีสิทธิทำก่อน (precedence)จากสูงไปหาต่ำ

ตัวดำเนินการ	***ความหมาย***	*associativity*
-	ทำให้มีค่าเป็นลบ	ขวาไปซ้าย
*****	คูณ	ซ้ายไปขวา
/	หาร	
%	เศษเหลือ(Modulus)	
+	บวก	ซ้ายไปขวา
-	ลบ	

ตัวดำเนินการ ***** **/** และ **%** มีการมีสิทธิทำก่อนเท่าเทียมกัน

ตัวดำนเนินการ **+** และ **-** มีการมีสิทธิทำก่อนเท่าเทียมกัน

การมีสิทธิทำก่อน (precedence หรือ priority) คือลำดับก่อนหลังในการทำงานของตัวดำเนินการ ตัวดำเนินการบางตัวจะสิทธิที่จะดำเนินการก่อนตัวอื่น เช่นในนิพจน์

2 + 3 * 4

ตัวดำเนินการคูณ (*****) มีสิทธิในการทำก่อนสูงกว่าตัวดำเนินการบวก (**+**) ดังนั้นในนิพจน์นี้จึงทำ 3 คูณ 4 ก่อน แล้วจึงบวกด้วย 2 ผลลัพธ์ที่ได้เท่ากับ 14

associativity คือทิศทางที่ตัวดำเนินการทำงานเมื่อมีตัวดำเนินการที่มีสิทธิทำก่อนเท่าเทียมกันมากกว่า ๑ ตัว โดยทั่วไปแล้ว เมื่อมีตัวเนินการหลายตัวที่มีสิทธิในการทำก่อนเท่ากันอยู่ในนิพจน์เดียวกัน มักจะดำเนินการตามลำดับจากซ้ายไปขวา เรียกว่ามี associativity จากซ้ายไปขวา ตัวดำเนินการบางตัวซึ่งเป็นส่วนน้อยจะมี associativity จากขวาไปซ้าย

ตัวอย่าง

```
2 + 3 * 4 + 5
```

ตัวดำเนินการคูณ (*****) มีสิทธิทำก่อนสูงสุดจึงทำ 3 * 4 ก่อน ได้เป็น 12 ดังนั้นจึงเหลือเป็น 2 + 12 + 5 ตัวดำเนินการบวก (+) ที่เหลือทั้งสองตัวมีสิทธิทำก่อนเท่ากัน จึงใช้หลัก associativity จากซ้ายไปขวาในการทำงาน ดังนั้น 2 และ 12 จึงบวกกันก่อนได้เป็น 14 แล้วในที่สุดจึงบวกด้วย 5 ได้ผลลัพธ์เป็น 19

ถ้าต้องการให้ตัวดำเนินการที่มีสิทธิทำก่อนต่ำกว่าได้ทำงานก่อนให้ใส่วงเล็บ เช่น

```
( 2 + 3) * ( 4 + 5)
```

มีลำดับการทำงานดังนี้ ลำดับแรกจะหาค่าของ `(2+3)` ซึ่งได้เท่ากับ 5 ต่อจากนั้นจะหาค่าของ `(4+5)` ได้เป็น 9 แล้วจึงนำ 5 ที่ได้จากวงเล็บแรกมาคูณด้วย 9 ซึ่งได้จากวงเล็บที่ ๒ ได้ผลลัพธ์สุดท้ายเท่ากับ 45

การแปลงชนิดโดยอัตโนมัติ

ภาษาจาวาเป็นภาษาที่มีการคำนวณอย่างต่ำครั้งละ 32บิต (เท่ากับขนาดของข้อมูลชนิด `int`) ถ้าในนิพจน์มีตัวแปรหรือค่าคงที่ที่มีขนาดเล็กกว่า `int` คือมีขนาดเล็กกว่า 32 บิต จะมีการแปลงค่าที่เล็กกว่า 32 บิตเหล่านั้นให้เป็น 32 บิตก่อนที่จะนำไปคำนวณ สำหรับข้อมูลชนิด `char` ซึ่งเป็นข้อมูลตัวอักขระสามารถนำไปใช้ในการคำนวณได้ โดยจะแปลงค่าของตัวอักขระนั้นตามที่ได้กำหนดไว้ในมาตรฐาน Unicode ให้เป็นชนิด `int` ก่อนนำไปใช้ในนิพจน์ ตัวอย่างเช่น

```
int  a;
char c ='ก';
a =c + 1;
```

หลังจากทำข้อความสั่งเหล่านี้แล้ว ตัวแปร `a` จะมีค่าเป็น 3586 เนื่องจากตัวแปร `c` มีค่าเป็นตัวอักขระ ‘ก’ ซึ่งถูกกำหนดค่าตามมาตรฐาน Unicode ตรงกับ 3585 ในเลขฐานสิบ หรือ 0x0E01 ในเลขฐานสิบหก อักขระมีขนาด 16 บิต จะถูกแปลงให้เป็น 32 บิตตามขนาดของ `int` โดยที่ค่ายังคงเดิม เมื่อบวกด้วย 1 จะได้เป็น 3586 เป็นชนิด `int`

ในกรณีที่ตัวถูกดำเนินมีหลายตัวและเป็นค่าตัวเลขต่างชนิดกัน ผลลัพธ์ที่ได้จะเป็นชนิดเดียวกับตัวถูกดำเนินการที่สามารถรองรับค่าตัวเลข (magnitude) ที่มีขนาดใหญ่สุดที่อยู่ในนิพจน์ สำหรับความสามารถในการรองรับค่าตัวเลขเรียงตามลำดับจากเล็กไปหาใหญ่ดังนี้

`int` → `long` → `float` → `double`

โปรดสังเกตุว่า ข้อมูลชนิด `float` สามารถรองรับค่าของข้อมูลที่มีขนาดใหญ่ได้มากกว่าชนิด `long` ทั้งๆ ที่ขนาดของข้อมูลเล็กกว่า กล่าวคือ `long` มีขนาด 64 บิต ในขณะที่ float มีขนาด 32 บิต ทั้งนี้ชนิด `float` จะพยายามรักษาค่าของข้อมูลที่มีค่าต่ำหรือสูงมากๆ เอาไว้โดยลดความละเอียด (precision) ลง ดังนั้นค่าของข้อมูลชนิด `float` จึง “ใหญ่” แต่ “หยาบ” เก็บข้อมูลได้ไม่ละเอียด ซึ่งอาจมีการปัดขึ้นหรือปัดลงจนเกิดความคลาดเคลื่อน ซึ่งหากใช้เก็บข้อมูลทางวิทยาศาสตร์หรือวิศวกรรมแล้ว ถึงแม้จะคลาดเคลื่อนไปบ้างแต่ยังคงรักษา”ความใหญ่” ไว้ได้ จึงเป็นที่นิยมในการเก็บข้อมูลด้านวิทยาศาสตร์หรือวิศวกรรมเป็นชนิด `float` หรือ `double` แต่ความคลาดเคลื่อน

ถึงจะมีเพียงเล็กน้อยมักยอมรับไม่ได้ในการใช้งานทางธุรกิจ เช่นในการเก็บตัวเลขจำนวนเงินทางบัญชี ซึ่งมักไม่ยอมให้เกิดความคลาดเคลื่อนแม้แต่สตางค์เดียว

ตัวอย่างการใช้ข้อมูลหลายชนิดประสมกันในนิพจน์เช่น

`5 + 7.5F` ชนิด `int` บวกกับชนิด `float` จะได้ผลลัพธ์เป็นชนิด `float` มีค่าเป็น `12.5F`

`1000000000 + 12.5F` ได้ผลลัพธ์เป็นชนิด `float` จากการทดสอบด้วยโปรแกรมให้พิมพ์ผลลัพธ์ของนิพจน์นี้ พบว่าได้คำตอบเป็น `1.0E9` ซึ่งเท่ากับ 1,000,000,000 แทนที่จะเป็น 1,000,000,012.5 จะเห็นว่ามีความคลาดเคลื่อนเกิดขึ้น

ตัวดำเนินการหารทำหน้าที่ของมันตามปรกติทั่วไปที่เราคุ้นเคย ยกเว้นกรณีที่ตัวถูกดำเนินการเป็นเลขจำนวนเต็มทั้งคู่ เศษเหลือจากการหารจะถูกตัดทิ้งไป ผลลัพธ์ที่ได้จากการหารจะเป็นเลขจำนวนเต็ม ไม่สามารถเก็บทศนิยมได้ ดังนั้น

`10 / 2`	ได้ผลลัพธ์เป็น 2 ชนิด `int`
`13.5 / 5`	ได้ผลลัพธ์เป็น 2.6 ชนิด `double`
`13 / 5.0`	ได้ผลลัพธ์เป็น 2.6 ชนิด `double`
`13 / 5`	ได้ผลลัพธ์เป็น 2 ชนิด `int` ไม่ใช่ `2.6`

ตัวอย่างข้างต้นบรรทัดสุดท้ายเป็นการกระทำระหว่างข้อมูลชนิด `int` ด้วยกันทั้งคู่ ค่าผลลัพธ์ของนิพจน์ที่ได้จึงเป็น `int` เช่นกัน ดังนั้นผลลัพธ์ที่ได้จะเป็นจำนวนเต็มเศษจากการหารจะถูกตัดทิ้งไป ค่าผลลัพธ์ของนิพจน์ที่ได้จึงเป็นชนิด `int` มีค่าเป็น 2

ถ้าตัวถูกดำเนินการตัวใดตัวหนึ่งเป็นลบ ผลลัพธ์จากการหารจะได้เป็นลบ ถ้าตัวถูกดำเนินการมีค่าเป็นลบทั้งคู่ ผลลัพธ์จากการหารจะมีค่าเป็นบวก เช่น

-13 / 5	ได้ผลลัพธ์เป็น -2
13 / -5	ได้ผลลัพธ์เป็น -2
-13 / -5	ได้ผลลัพธ์เป็น 2

ในกรณีที่มีการหารด้วยศูนย์ (0) ถ้าผลลัพธ์ของนิพจน์ได้เป็นชนิด `float` หรือ `double` จะได้ค่าของผลลัพธ์เป็นอนันต์ (infinity) แต่ถ้าผลลัพธ์ของนิพจน์เป็นเลขจำนวนเต็มจะเกิดข้อผิดพลาดขึ้นเมื่อมีการหารด้วยศูนย์ โปรแกรมจะยุติการทำงาน

ถ้าทั้งตัวตั้งและตัวหารต่างเป็นชนิดเลขทศนิยมลอยตัวมีค่าศูนย์ (เช่น 0.0f หรือ 0.0d)ทั้งคู่ ผลลัพธ์จะได้เป็น `NaN` (Not a Number)

ตัวดำเนินการมอดูลัส (%) ให้ผลลัพธ์เป็นเลขจำนวนเต็มซึ่งได้มาจากเศษเหลือจากการหารของตัวถูกดำเนินการสองตัว เช่น

13 % 5 ให้ผลลัพธ์เป็น 3 เนื่องจาก 13 หารด้วย 5 ได้จำนวนเต็มที่มากที่สุดคือ 2 ผลคูณของ 2 กับ 5 คือ 10 ดังนั้นเศษเหลือของการหารคือ 13 - 10 ได้เป็น 3

15 % 5 ให้ผลลัพธ์เป็น 0 เนื่องจาก 15 หารด้วย 5 ลงตัว

ถ้าตัวถูกดำเนินการตัวใดตัวหนึ่งเป็นลบ เศษเหลือที่ได้จะมีเครื่องหลายเช่นเดียวกับตัวตั้ง เช่น

-13 % 5 ให้ผลลัพธ์เป็น -3

13 % -5 ให้ผลลัพธ์เป็น 3

ถ้าตัวหารเป็นเลขจำนวนเต็มมีค่าเป็นศูนย์ จะเกิดข้อผิดพลาดโปรแกรมจะทำงานต่อไม่ได้ แต่ถ้าตัวหารเป็นเลขแบบมีทศนิยมมีค่าเป็นศูนย์จะได้ผลลัพธ์เป็น **NaN** (Not a Number)

ข้อควรระวัง

การหารด้วยศูนย์ (0) ถือว่าเป็นเรื่องไม่ปรกติ คอมไพเลอร์จาวาบางรุ่นมีการตรวจสอบด้วยว่าในโปรแกรมต้นฉบับมีการหารด้วยค่าคงที่ศูนย์หรือไม่ ถ้าในโปรแกรมมีการหารด้วยค่าคงที่ศูนย์ เช่น X / 0 คอมไพเลอร์จะถือว่าเขียนโปรแกรมไม่ถูกต้อง คอมไพเลอร์จะแจ้งให้เราทราบว่ามีการหารด้วยศูนย์ แต่ถ้าตัวหารไม่ใช่ค่าคงที่แต่เป็นตัวแปร คอมไพเลอร์จะตรวจสอบไม่ได้ว่าตัวหารมีค่าเป็นศูนย์หรือไม่ ซึ่งหากนำโปรแกรมที่คอมไพล์แล้วนี้ไปใช้งาน ขณะที่โปรแกรมกำลังทำงานอยู่นั้นบังเอิญว่าตัวแปรที่เป็นตัวหารมีเกิดมีค่าเป็นศูนย์ขึ้นมา จะเกิด*สิ่งผิดปรกติขณะทำงาน* (runtime exception) ซึ่งจะทำให้โปรแกรมไม่สามารถทำงานต่อไปได้ เว้นเสียแต่ว่าเราได้เตรียมการรับมือกับความผิดปรกตินี้เอาไว้แล้ว รายละเอียดเกี่ยวกับการเตรียมการรับมือความผิดปรกติจะกล่าวถึงในบทที่ ๙

ตัวดำเนินการกำหนดค่า

ตัวดำเนินการกำหนดค่า(Assignment operator)เป็นตัวดำเนินการทวิภาคใช้กำหนดค่าใหม่ให้กับตัวแปร มีรูปแบบอย่างง่ายดังนี้

ตัวแปร = นิพจน์

จะทำงานโดยการหาค่าของนิพจน์แล้วนำค่าที่ได้นั้นไปใส่ไว้ที่ตัวแปรซึ่งเขียนไว้ด้านซ้ายของตัวดำเนินการกำหนดค่า (=)

ตัวอย่าง

```
speed = distance / time; /* หาร distance ด้วย time แล้วเก็บผลลัพธ์ไว้ที่ speed */
count = count + 1;       /* บวก count ด้วย 1 แล้วเก็บผลลัพธ์ไว้ที่ count */
```

เราสามารถกำหนดค่าให้ตัวแปรหลายตัวได้พร้อมกันในประโยคเดียวได้ดังตัวอย่าง

```
x = y = z = p + q
```

มีความหมายเช่นเดียวกับ

```
x = (y = (z = p + q))
```

ด้านซ้ายของตัวดำเนินการกำหนดค่าไม่สามารถเป็นค่าคงที่ได้อย่างสมการทางคณิตศาสตร์ทั่วไป เช่น

```
5 = x;
```

เนื่องจากตัวดำเนินการกำหนดค่าใช้ในการนำค่าทางขวาไปใส่ไว้ทางซ้าย ดังนั้น 5 ซึ่งเป็นค่าคงที่ไม่ใช่ตัวแปรจึงไม่สามารถรับค่าใดๆเข้าไปเก็บได้ ตัวอย่างนี้จึงใช้ไม่ได้ในภาษาจาวา

ตัวดำเนินการกำหนดค่าประกอบ

ในกรณีที่ต้องการเอาค่าในตัวแปรมาดำเนินการทางคณิตศาสตร์แล้วนำผลลัพธ์ที่ได้ไปเก็บไว้ในตัวแปรเดิมเราสามารถเขียนข้อความสั่งแบบย่อโดยใช้ตัวดำเนินการกำหนดค่าประกอบ (Compound assignment operators)ต่อไปนี้

ตัวดำเนินการ	ความหมาย
+=	บวกเพิ่มตัวแปรทางซ้ายด้วยค่าทางขวาของตัวดำเนินการ
-=	ลบค่าของตัวแปรทางซ้ายออกด้วยค่าทางขวาของตัวดำเนินการ
*=	คูณตัวแปรทางซ้ายด้วยค่าทางขวาของตัวดำเนินการ
/=	หารตัวแปรทางซ้ายด้วยค่าทางขวาของตัวดำเนินการ
%=	หารตัวแปรจำนวนเต็มทางซ้ายด้วยค่าทางขวาของตัวดำเนินการแล้วนำเศษเหลือจากการหารไปไว้ในตัวแปรทางซ้าย

เช่น `count += 2` หมายความว่า บวก `count` ด้วย 2 เทียบเท่าประโยคคำสั่ง `count = count + 2`

ตัวดำเนินการกำหนดค่าประกอบเหล่านี้สามารถใช้ทดแทนตัวดำเนินการคำนวณทั้ง ๕ ตัวได้ในกรณีที่ตัวถูกดำเนินการเป็นตัวแปรตัวเดียวกับตัวแปรที่อยู่ทางซ้ายของตัวดำเนินการกำหนดค่า (=) ตัวอย่างเช่น

```
count += 5          เทียบเท่า    count = count + 5
stock += 3          เทียบเท่า    stock = stock - 3
power *= 2.71828    เทียบเท่า    power = power * 2.71828
share /= 10.0       เทียบเท่า    share = share / 10.0
remainder %= 10     เทียบเท่า    remainder = remainder % 10
```

เราสามารถใช้ตัวดำเนินการกำหนดค่าประกอบเหล่านี้ได้กับทุกชนิดของข้อมูลทางคณิตศาสตร์ ยกเว้น ตัวดำเนินการประกอบมอดูลัส (%=) ที่ใช้ได้เฉพาะกับข้อมูลชนิดจำนวนเต็มเท่านั้น

ตัวดำเนินการเพิ่มค่าและลดค่า

++ เป็นตัวดำเนินการเพิ่มค่า (Increment operator) ใช้สำหรับเพิ่มค่าให้กับตัวแปร จะทำให้ตัวแปรที่ได้รับการกระทำมีค่าสูงขึ้นจากเดิมอีกหนึ่ง เช่น n++ ถ้าเดิม n มีค่าเป็น 5 หลังจากทำนิพจน์นี้แล้ว n จะมีค่าเป็น 6

`--` เป็นตัวดำเนินการลดค่า (Decrement operator) ใช้สำหรับลดค่าของตัวแปรให้มีค่าน้อยลงกว่าเดิมอีกหนึ่ง

ตัวดำเนินการเพิ่มค่า และตัวดำเนินการลดค่านี้ใช้ได้ ๒ รูปแบบคือจะใส่ตัวดำเนินการไว้หน้าหรือหลังตัวแปรก็ได้แต่มีความหมายต่างกัน ถ้าใส่ตัวดำเนินการเพิ่มค่าไว้หน้าตัวแปรเรียกว่า ตัวดำเนินการเพิ่มค่าก่อน(preincrement) จะทำการเพิ่มค่าของตัวแปรด้วย 1 ก่อนแล้วจึงนำค่าใหม่ที่ได้ไปใช้ แต่ถ้าใส่ตัวดำเนินการเพิ่มค่าไว้หลังตัวแปร เรียกว่าตัวดำเนินการเพิ่มค่าหลัง(postincrement)จะนำค่าของตัวแปรไปใช้ก่อนแล้วจึงจะเพิ่มค่าให้กับตัวแปรนั้น ในทำนองเดียวกันตัวดำเนินการลดค่าก็จะมีทั้งตัวดำเนินการลดค่าก่อน (predecrement) และตัวดำเนินการลดค่าหลัง (postdecrement)

ตัวอย่าง

```
๑. count=0;
๒. sum=0;
๓. count++;            /* count เพิ่มค่าจาก 0 เป็น 1 */
๔. ++count;            /* count เพิ่มค่าจาก 1 เป็น 2 */
๕. sum += count++;     /* sum มีค่าเป็น 2 count มีค่าเป็น 3 */
๖. sum += ++count;     /* sum มีค่าเป็น 6 count มีค่าเป็น 4 */
```

ในกรณีที่ต้องการเพิ่มหรือลดค่าตัวแปรเพียงอย่างเดียวไม่ต้องการนำผลลัพธ์นั้นไปใช้ในการส่งค่าไปให้เมธอดในทันที หรือใช้ในนิพจน์ที่ซับซ้อนเช่นตัวอย่างในบรรทัดที่ ๓ และ ๔ จะใส่ตัวดำเนินการไว้หน้าหรือหลังตัวแปรผลลัพธ์ที่ได้จะไม่แตกต่างกันถึงแม้ว่าเมธอดทำงานจะแตกต่างกันก็ตาม แต่ถ้าใช้ในรูปแบบที่ซับซ้อนขึ้นเช่นตัวอย่างในบรรทัดที่ ๔ และ ๕ จะมีการนำค่าของตัวแปรที่ถูกดำเนินการโดยตัวดำเนินการเพิ่มหรือลดค่าไปใช้อย่างอื่นด้วย ซึ่งการเพิ่มหรือลดค่าของตัวแปรก่อนนำค่าของตัวแปรไปใช้ กับการที่นำค่าของตัวแปรไปใช้ก่อนที่จะทำการเพิ่มหรือลดค่านั้นจะให้ผลที่แตกต่างกัน ตัวอย่างในบรรทัดที่ ๕ ขณะที่เริ่มเข้าสู่บรรทัดนี้ `count` มีค่าเป็น 2 ตัวดำเนินการเพิ่มค่าอยู่หลังตัวแปร `count` จึงนำค่าของ count ไปใช้ก่อนที่จะเพิ่มค่าให้กับ `count` จึงบวก `sum` ด้วย 2 ส่วน `count` จะเพิ่มค่าด้วยหนึ่งหลังจากนำค่าเดิมไปบวกเข้ากับ `sum` แล้ว ดังนั้น หลังจากทำบรรทัดนี้เสร็จสิ้นแล้ว `sum` จึงมีค่าเป็น 2 ส่วน count มีค่าเป็น 3

ส่วนตัวอย่างในบรรทัดที่ ๖ ขณะเริ่มเข้าสู่บรรทัดนี้ `sum` มีค่าเป็น 2 ส่วน `count` มีค่าเป็น 3 ตัวดำเนินการเพิ่มค่าอยู่หน้าตัวแปรดังนั้นจึงทำการเพิ่มค่าของตัวแปรก่อนที่จะนำค่าของตัวแปรไปใช้ จึงเพิ่มค่าของ `count` จาก 3 เป็น 4 แล้วจึงนำค่าของ `count` คือ 4 ไปบวกเพิ่มในตัวแปร `sum` ทำให้ `sum` มีค่า เป็น 6

ตัวอย่างการใช้ตัวดำเนินการเพิ่มหรือลดค่ากับการส่งค่าให้เมธอด เช่น

```
abc(x++)
abc(--y)
```

ตัวอย่างแรกจะส่งค่าของ x ก่อนทำการเพิ่มค่าไปให้เมธอด abc แล้วจึงจะทำการเพิ่มค่าของ x
ตัวอย่างหลังจะลดค่าของ y ลงก่อนแล้วจึงส่งค่าที่ได้นั้นไปให้เมธอด `abc`

ตัวดำเนินการสัมพันธ์

ตัวดำเนินการสัมพันธ์(Relational Operators)ใช้ในการเปรียบเทียบนิพจน์หนึ่งกับอีกนิพจน์หนึ่ง จะได้ผลลัพธ์เป็นนิพจน์แบบ*บูล* (boolean expression) ซึ่งมีค่าเป็นจริงหรือไม่ก็*เท็จ* ถ้าเปรียบเทียบแล้วได้ผลเป็นเท็จ จะได้ผลลัพธ์มีค่าแบบ*บูล*เป็น `false` ถ้าเป็นจริงจะได้ค่าแบบ*บูล*เป็น `true`

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

เราใช้ตัวดำเนินการสัมพันธ์ร่วมกับข้อความสั่งที่ต้องการตรวจสอบเงื่อนไข เช่นใช้ในข้อความสั่งที่ควบคุมการทำงานซ้ำเป็นวงวน(กล่าวถึงในบทอื่น) หรือข้อความสั่ง `if` ตัวอย่างเช่น

```
if (gender == 'F')
      Females++;
```

จะมีการหาค่าของเงื่อนไข `gender == 'F'` ว่าเป็นจริงหรือเป็นเท็จโดยการเปรียบเทียบว่า `gender` มีค่าเท่ากับอักขระ `'F'` หรือไม่ ถ้าเท่ากันจะได้ค่าเป็นจริง ถ้าไม่เท่าจะได้ค่าเป็นเท็จ

ตัวดำเนินการตรรกะ

ตัวดำเนินการตรรกะ(Logical Operators) ใช้ในการประสมนิพจน์เชิงสัมพันธ์ (relational expressions)

ตัวดำเนินการ	ความหมาย	หมายเหตุ
!	ตรรกะนอต (logical NOT)	ให้ผลลัพธ์เป็นจริง ถ้าตัวถูกดำเนินการมีค่าเป็นเท็จ ให้ผลลัพธ์เป็นเท็จ ถ้าตัวถูกดำเนินการมีค่าเป็นจริง
&&	ตรรกะแอนด์ (logical AND)	ให้ผลลัพธ์เป็นจริง ถ้าตัวถูกดำเนินการทั้งคู่เป็นจริง ให้ผลลัพธ์เป็นเท็จ ถ้าตัวถูกดำเนินการตัวแรกเป็นเท็จ (จะไม่สนใจตัวถูกดำเนินการตัวที่สอง ถ้าตัวถูกดำเนินการตัวแรกเป็นเท็จ)

`||`	ตรรกะออร์ (logical OR)	ให้ผลลัพธ์เป็นเท็จ ถ้าตัวถูกดำเนินการทั้งคู่เป็นเท็จ ผลลัพธ์เป็นจริง ถ้าตัวถูกดำเนินการ ตัวแรกเป็นจริง (จะไม่สนใจตัวถูกดำเนินการตัวที่สอง ถ้าตัวถูกดำเนินการตัวแรกเป็นจริง)

ตัวดำเนินการตรรกะนอต(`!`) เป็นตัวดำเนินการเอกภาคต้องการตัวถูกดำเนินการเพียงตัวเดียวซึ่งจะเขียนไว้หลังตัวดำเนินการ ตัวดำเนินการตรรกะนอตนี้จะดำเนินการเปลี่ยนค่าทางตรรกะของตัวถูกดำเนินการให้มีค่าเป็นตรงกันข้าม เช่นถ้าค่าของตัวถูกดำเนินการมีค่าเป็นจริงก็จะได้ผลลัพธ์เป็นเท็จ ถ้าตัวถูกดำเนินการมีค่าเป็นเท็จก็จะได้ผลลัพธ์เป็นจริง

ตารางความจริงต่อไปนี้สรุปผลลัพธ์ของการใช้ตัวดำเนินการตรรกะนอต

นิพจน์	**! นิพจน์**
`false`	`true`
`true`	`false`

ตัวดำเนินการตรรกะแอนด์(`&&`) และตัวดำเนินการตรรกะออร์(`||`) เป็นตัวดำเนินการทวิภาคต้องการตัวถูกดำเนินการสองตัว เราใช้ตัวดำเนินการเหล่านี้เชื่อมนิพจน์ที่ให้ผลลัพธ์เป็นค่าเชิงตรรกะ (จริงหรือเท็จ) เข้าด้วยกันเพื่อให้ได้ผลลัพธ์ทางตรรกะเพียงค่าเดียว

ตารางความจริง (truth table) ต่อไปนี้สรุปผลลัพธ์ของการใช้ตัวดำเนินการตรรกะแอนด์

นิพจน์ 1	**นิพจน์** 2	**ผลลัพธ์ของ นิพจน์** 1 **&& นิพจน์** 2
`false`	`false`	`false`
`false`	`true`	`false`
`true`	`false`	`false`
`true`	`true`	`true`

ตารางความจริง (truth table) ต่อไปนี้สรุปผลลัพธ์ของการใช้ตัวดำเนินการตรรกะออร์

นิพจน์ 1	**นิพจน์** 2	**ผลลัพธ์ของ นิพจน์** 1 **|| นิพจน์** 2
`false`	`false`	`false`
`false`	`true`	`true`
`true`	`false`	`true`
`true`	`true`	`true`

ตัวดำเนินการตรรกะนิยมใช้กับประโยคคำสั่ง `if` และใช้ในเงื่อนไขของการทำงานเป็นวงวนซึ่งจะกล่าวถึงในบทต่อไป

ตัวอย่างการใช้ตัวดำเนินการตรรกะแอนด์

```
if ( (x == 5) && (y == 6) )
      z = 0;
```

ตัวอย่างนี้มีการดำเนินการเชิงสัมพันธ์ 2 ส่วน ส่วนแรกคือ (x == 5) ส่วนที่สองคือ (y == 6) สองส่วนนี้เชื่อมด้วยตัวดำเนินการตรรกะแอนด์(&&) ดังนั้นผลลัพธ์จากการดำเนินการเชิงสัมพันธ์ทั้งสองส่วนนี้ต้องเป็นจริงทั้งคู่จึงจะมีการทำข้อความสั่ง z = 0 ของข้อความสั่ง if ดังนั้น ถ้า x มีค่าเท่ากับ 5 และ y มีค่าเท่ากับ 6 จึงจะทำให้ z มีค่าเป็นศูนย์

ตัวอย่างการใช้ตัวดำเนินการตรรกะออร์

```
if ( (x == 5) || (y == 6) )
      z = 0;
```

ตัวอย่างนี้ดัดแปลงมาจากตัวอย่างก่อนหน้านี้โดยเปลี่ยนตัวดำเนินการตรรกะแอนด์เป็นตัวดำเนินการตรรกะออร์ (||) ส่งผลให้ผลลัพธ์จากการดำเนินการเชิงสัมพันธ์เพียงส่วนใดส่วนหนึ่งเป็นจริงก็เพียงพอที่จะไปทำข้อความสั่ง z = 0 ของข้อความสั่ง if ได้แล้วดังนั้น ถ้า x มีค่าเท่ากับ 5 จะทำให้ z มีค่าเป็นศูนย์ไม่ว่า y จะมีค่าเป็นเท่าไร่ก็ตาม หรือถ้า y มีค่าเท่ากับ 6 ก็จะทำให้ z มีค่าเป็นศูนย์เช่นเดียวกันไม่ว่า x จะมีค่าเป็นเท่าใด หรือถ้านิพจน์เชิงสัมพันธ์สองนิพจน์นี้เป็นจริงท้าคู่ คือถ้า x มีค่าเท่ากับ 5 และ y มีค่าเท่ากับ 6 ก็จะยังคงทำให้ z มีค่าเป็นศูนย์ ในการทำงานจริงถ้าตัวถูกดำเนินการตัวแรกของตัวดำเนินการตรรกะออร์ ซึ่งในที่นี้คือนิพจน์เชิงสัมพันธ์ (x==5) เป็นจริง ก็จะไปทำข้อความสั่งของ if คือ z = 0 ได้เลยโดยไม่ต้องตรวจสอบตัวถูกดำเนินการตัวที่สองให้เสียเวลา ถ้าตัวถูกดำเนินการตัวแรกให้ผลลัพธ์เป็นเท็จจึงจะไปหาค่าของตัวถูกดำเนินการตัวที่สองต่อไป

ตัวอย่างการใช้ตัวดำเนินการตรรกะนอต

```
if ( !(x==5))
      z = 0;
```

ตัวอย่างนี้ถ้า x มีค่าเป็น 5 จะทำให้นิพจน์ (x==5) มีค่าตรรกะเป็นจริง แต่เมื่อถูกใช้กับตัวดำเนินการตรรกะนอตในรูป !(x==5) จะทำให้ผลลัพธ์ของนิพจน์ !(x==5) เป็นเท็จ ดังนั้นถ้า x มีค่าเป็น 5 จะไม่มีการทำข้อความสั่ง z = 0 แต่จะทำข้อความสั่งนี้ก็ต่อเมื่อ x มีค่าไม่เท่ากับ 5 ซึ่งประโยคคำสั่งนี้จะให้ผลเทียบเท่ากับประโยคคำสั่งต่อไปนี้

```
if ( x!=5)
      z = 0;
```

ข้อควรระวัง

นิพจน์ข้างขวาของตัวดำเนินการ && จะไม่มีโอกาสทำงานถ้านิพจน์ข้างซ้ายเป็นเท็จ หากตัวถูกดำเนินการข้างขวาเป็นนิพจน์ที่มีการเปลี่ยนค่าของตัวแปรหรือมีการเรียกเมธอด นิพจน์นี้จะไม่ได้รับการหาค่าซึ่งหมายถึงไม่มีการเปลี่ยนค่าตัวแปรหรือไม่มีการเรียกเมธอด ควรระวังว่าต้องการให้เป็นอย่างนั้นจริงหรือไม่ สำหรับตัวดำเนินการ || ก็ทำนองเดียวกัน ถ้านิพจน์ข้างซ้ายของมันเป็นจริง นิพจน์ข้างขวาจะไม่มีการทำงาน

ตัวดำเนินการตรรกะแบบบูล

การประสมนิพจน์เชิงสัมพันธ์ (relational expressions) โดยตัวดำเนินการตรรกะแบบธรรมดานั้น มีประสิทธิภาพสูงเนื่องจากอาจมีการทำงานลัดโดยไม่ไปหาค่านิพจน์ข้างขวาของตัวดำเนินการตรรกะ แต่ในบางกรณีเราอยากให้ทำนิพจน์ข้างขวาเสมอ จาวามีตัวดำเนินการตรรกะแบบบูล (Boolean Logical Operators) สำหรับทำงานแบบนี้

ตัวดำเนินการ	ความหมาย	หมายเหตุ
&	ตรรกะแอนด์แบบบูล (boolean logical AND)	ได้ผลลัพธ์เป็นเท็จ ถ้าตัวถูกดำเนินการตัวใดตัวหนึ่งเป็นเท็จ ได้ผลลัพธ์เป็นจริง ถ้าตัวถูกดำเนินการทั้งสอง เป็นจริง
|	ตรรกะออร์รวมแบบบูล (boolean logical inclusive OR)	ได้ผลลัพธ์เป็นจริง ถ้าตัวถูกดำเนินการตัวใดตัวหนึ่งเป็นจริง ได้ผลลัพธ์เป็นเท็จ ถ้าตัวถูกดำเนินการ ทั้งสองตัวเป็นเท็จ
^	ตรรกะออร์เฉพาะแบบบูล (boolean logical exclusive OR)	ได้ผลลัพธ์เป็นจริง ถ้าตัวถูกดำเนินการทั้งคู่มีค่าต่างกัน ได้ผลลัพธ์เป็นเท็จ ถ้าตัวถูกดำเนินการทั้งคู่มีค่าเหมือนกัน

ตัวดำเนินการตรรกะแบบบูล แอนด์ (&) และ ตัวดำเนินการตรรกะออร์รวมแบบบูล (boolean logical inclusive OR) ซึ่งแทนด้วยอักขระ vertical bar (|) ให้ผลลัพธ์เหมือนกับตัวดำเนินการตรรกะแอนด์ และ ตัวดำเนินการตรรกะออร์ตามลำดับ แต่การทำงานต่างกันเล็กน้อย คือ ตัวดำเนินการตรรกะแบบบูลจะทำการหาค่าของตัวถูกดำเนินการทั้งสองข้างของมันเสมอ จะไม่ทำงานแบบลัดเหมือนอย่างดำเนินการตรรกะธรรมดา ดังนั้น นิพจน์ต่อไปนี้

```
gender == 'F' & age >= 65
```

จะหาค่าของนิพจน์ `age>= 65` เสมอไม่ว่านิพจน์ `gender == 'F'` จะให้ค่าเป็นจริงหรือเป็นเท็จก็ตาม การทำงานเช่นนี้จะไม่ทำให้เกิดผลข้างเคียงในบางสถานการณ์ เช่นถ้าเราเขียนนิพจน์ต่อไปนี้

```
gender == 'F' || ++Count  >=100
```

`Count` จะไม่มีการเพิ่มค่าถ้า `gender` ไม่เท่ากับ `'F'` เนื่องจากถ้านิพจน์แรกเป็นเท็จ จะไม่มีการหาค่านิพจน์ที่สองด้านขวาของ `||` แต่ถ้าเราต้องการให้ทำนิพจน์ด้านขวาเสมอไม่ว่านิพจน์ด้านซ้ายจะให้ผลอย่างไรเราควรเขียนเป็น

```
gender == 'F' | ++Count  >= 100
```

วิธีนี้รับประกันได้ว่านิพจน์ข้างขวาจะถูกหาค่าเสมอ

เงื่อนไขที่มีตัวดำเนินการตรรกะเฉพาะออร์แบบบูล (^) ให้ผลลัพธ์เป็นจริงก็ต่อเมื่อตัวถูกดำเนินการตัวหนึ่งของมันเป็นจริงและอีกตัวเป็นเท็จเท่านั้น ถ้าตัวถูกดำเนินการทั้งคู่เป็นจริง หรือเป็นเท็จทั้งคู่ผลลัพธ์ของทั้งเงื่อนไขจะเป็นเท็จ

ตารางความจริง (truth table) ต่อไปนี้สรุปผลลัพธ์ของการใช้ตัวดำเนินการตรรกะออร์เฉพาะแบบบูล

นิพจน์ 1	นิพจน์ 2	ผลลัพธ์ของ นิพจน์ 1 ^ นิพจน์ 2
false	False	false
false	True	true
true	False	true
true	True	false

ตัวดำเนินการเงื่อนไข

ตัวดำเนินการเงื่อนไข (Conditional Operator) เป็นตัวดำเนินการไตรภาค ใช้อักขระ ? และ : เป็นสัญญลักษณ์แทน มีรูปแบบดังนี้

นิพจน์ ๑ **?** *นิพจน์ ๒* : *นิพจน์ ๓*

ตัวดำเนินการ ? : และเป็นตัวดำเนินการไตรภาคเพียงตัวเดียวของภาษาจาวา มันต้องการตัวถูกดำเนินการ ๓ ตัว ตัวแรกเป็นนิพจน์เงื่อนไขที่ให้ผลลัพธ์เป็นชนิด boolean ตัวถูกดำเนินการตัวที่ ๒ เป็นนิพจน์ที่จะถูกหาค่าเป็นผลลัพธ์ของการดำเนินการถ้าเงื่อนไขของนิพจน์แรกเป็นจริง ส่วนตัวถูกดำเนินการตัวที่ ๓ เป็นนิพจน์ที่จะถูกหาค่าและใช้เป็นผลลัพธ์ของการดำเนินการถ้าผลลัพธ์ของนิพจน์ที่ ๑ ซึ่งใช้เป็นเงื่อนไขให้ค่าเป็นเท็จ ตัวอย่างเช่น ข้อความสั่งสำหรับพิมพ์ข้อความ

```
System.out.println( gender == 'M' ? "Male" : "Female");
```

นิพจน์แรกคือ `gender == 'M'` เป็นนิพจน์เงื่อนไขต้องเป็นนิพจน์แบบบูล ถ้ามีค่าเป็นจริง จะส่งค่าของนิพจน์ที่สองเป็นผลลัพธ์คือให้ผลลัพธ์เป็นสายอักขระ `"Male"` แต่ถ้านิพจน์เงื่อนไขเป็นเท็จ จะนำค่าของนิพจน์ที่สามคือสายอักขระ `"Female"` มาเป็นผลลัพธ์แทน

ตัวอย่างนี้ถ้าเขียนด้วยข้อความสั่ง `if` จะเป็นดังนี้

```
if (gender == 'M')
      System.out.println("Male");
else
      System.out.println("Female");
```

ซึ่งจะเยิ่นเย้อกว่าการใช้ตัวดำเนินการเงื่อนไข แต่การใช้ตัวดำเนินการเงื่อนไขจะทำให้โปรแกรมเข้าใจยากขึ้น

นิพจน์ที่สองและที่สามอาจเขียนเป็นนิพจน์ที่ให้ค่าต่างชนิดกันก็ได้ แต่ต้องเป็นชนิดที่แปลงให้เข้ากันได้ เช่นนิพจน์ที่สองเป็นชนิด `int` ส่วนนิพจน์ที่สามอาจเป็นชนิด `double` ค่าของผลลัพธ์ที่ได้จากการหาค่าของตัวดำเนินการเงื่อนไข ?: นี้จะมีเพียงชนิดเดียวเท่านั้น โดยยึดเอาชนิดที่สามารถเก็บค่าของข้อมูลที่มีขนาดใหญ่กว่าเป็นหลัก เช่น

```
(value > 80) ? 3.5 : 3
```

ผลลัพธ์ที่ได้จากการดำเนินการของตัวดำเนินการเงื่อนไขจะเป็นชนิด `double` ไม่ว่า นิพจน์เงื่อนไขจะเป็นจริงหรือเท็จ ในตัวอย่างนี้ ถ้าเงื่อนไขในนิพจน์ ๑ เป็นเท็จ ค่าของนิพจน์นี้ทั้งชุดจะได้จากนิพจน์ ๓ แต่จะมีการแปลงชนิดจาก `int` เป็น `double` เพื่อให้เข้ากันได้กับนิพจน์ ๒ ซึ่งเป็น double ซึ่งถือว่ามีขนาดในการรองรับค่าข้อมูลที่ใหญ่กว่า

ตารางต่อไปนี้แสดงการมีสิทธิทำก่อนและ associativity ของตัวดำเนินการเท่าที่กล่าวมาแล้ว

ตัวดำเนินการ	Associativity	**ประเภท**
`()`	ซ้ายไปขวา	วงเล็บ
`++ -- + - ! (type)`	ขวาไปซ้าย	เอกภาค
`* / %`	ซ้ายไปขวา	การคูณหาร
`+ -`	ซ้ายไปขวา	การเพิ่มลด
`< <= > >=`	ซ้ายไปขวา	การเปรียบเทียบ
`== !=`	ซ้ายไปขวา	ความเท่าเทียม
`&`	ซ้ายไปขวา	ตรรกะแอนด์แบบบูล
`^`	ซ้ายไปขวา	ตรรกะออร์เฉพาะแบบบูล
`|`	ซ้ายไปขวา	ตรรกะออร์รวมแบบบูล
`&&`	ซ้ายไปขวา	ตรรกะแอนด์
`||`	ซ้ายไปขวา	ตรรกะออร์
`?:`	ขวาไปซ้าย	เงื่อนไข
`= += -= *= /= %=`	ขวาไปซ้าย	การกำหนดค่า

ข้อความสั่ง

ข้อความสั่ง (statement) แบ่งตามโครงสร้างเป็นสองประเภทคือข้อความสั่งอย่างง่าย (simple statement) และข้อความสั่งประกอบ (compound statement หรือ block statement) ข้อความสั่งอย่างง่ายประกอบด้วยนิพจน์ใดก็ได้แต่มีเพียงนิพจน์เดียว ส่วนข้อความสั่งประกอบจะประกอบด้วยข้อความสั่งอย่างง่ายเพียงข้อความสั่งเดียวหรือหลายข้อความสั่งก็ได้แต่ต้องบรรจุอยู่ในวงเล็บปีกกา `{ }` ตัวแปรภาษาจาวาจะปฏิบัติต่อข้อความสั่งประกอบราวกับว่าเป็นข้อความสั่งอย่างง่ายหนึ่งข้อความ

ตัวอย่างข้อความสั่งอย่างง่าย

```
x = y + 5;    /* simple statement  */
```

ตัวอย่างข้อความสั่งประกอบ

```
{
      y = x * 7;
      z = y + 20;
}
```

ภายในบล็อกของข้อความสั่งประกอบอาจมีการประกาศตัวแปรอยู่ภายในด้วยก็ได้ ตัวแปรที่ประกาศขึ้นภายในบล็อกถือว่าเป็นตัวแปรเฉพาะที่ (*local variable*) ซึ่งสามารถใช้งานได้เฉพาะในบล็อกที่ประกาศขึ้นมาหรือในบล็อกชั้นในที่ลึกลงไปเท่านั้น เช่น

```
int x;
{
      int a=10;
      x =a *a;
      {
            int b=20;
            a =a +b;
      }
}
```

ตัวอย่างข้างบนนี้แสดงการสร้างบล็อกซ้อนบล็อก บล็อกชั้นในสามารถใช้ตัวแปรที่ประกาศไว้ในบล็อกชั้นนอกที่ล้อมรอบมันได้ แต่บล็อกชั้นนอกไม่สามารถใช้ตัวแปรที่ประกาศไว้ในบล็อกชั้นใน

การแปลงผันชนิดข้อมูล (Type Conversion)

จาวาอนุญาตให้ใช้ค่าคงที่และตัวแปรหลายชนิดผสมกันได้ในนิพจน์ แต่ผลลัพธ์ที่ได้จากการหาค่าของนิพจน์มีเพียงค่าเดียว ซึ่งหมายถึงว่ามีชนิดของข้อมูลได้ชนิดเดียวด้วยเช่นกัน ค่าที่ได้นี้จะเป็นชนิดใดนั้น มีกฎเกณฑ์กำกับอยู่ จาวายึดกฎเกณฑ์ที่เข้มงวดในการแปลงชนิดข้อมูล

การแปลงผันโดยอัตโนมัติ

ในการหาค่าของนิพจน์นั้น จาวาจะดำเนินการตามการทำงานของตัวดำเนินการทีละตัว ในการดำเนินการแต่ละครั้งมักกระทำกับตัวถูกดำเนินการ ๒ ตัวซึ่งอาจเป็นคนละประเภทกันก็ได้ ผลลัพธ์ที่ได้จากการดำเนินการในแต่ละครั้งจะได้เป็นข้อมูลชนิดสูงสุดของตัวถูกดำเนินการที่ใช้ในนิพจน์นั้น ข้อมูลชนิดที่ต่ำกว่าจะถูกแปลงเป็นชนิดที่สูงสุดในนิพจน์นั้นๆ โดยอัตโนมัติ การแปลงผันชนิดจากชนิดที่ต่ำกว่าไปเป็นชนิดที่สูงกว่าเรียกว่า *การเลื่อนระดับสูงขึ้น* (promotion) ซึ่งคอมไพเลอร์จะทำให้เองโดยอัตโนมัติ

ตารางต่อไปนี้แสดงผลลัพธ์จากการแปลงชนิดซึ่งคอมไพเลอร์ทำให้โดยอัตโนมัติ จะเห็นว่าได้ผลลัพธ์เป็นชนิดสูงสุดของในแต่ละคู่ ยกเว้นชนิดใดที่ต่ำกว่า `int` จะได้เป็น `int` เสมอ ทั้งนี้เนื่องจากจาวาเป็นภาษาที่คำนวณแบบ 32 บิต การคำนวณเลขจำนวนเต็มแต่ละครั้งจะคำนวณทีละ 32 บิต ข้อมูลใดที่เล็กกว่า 32 บิต จะถูกแปลงให้เป็น 32 บิต คือขนาดของ `int` ก่อนแล้วจึงนำไปคำนวณ ผลลัพธ์ที่ได้ยังคงเป็น 32 บิตคือชนิด `int` นั่นเอง

operand	byte	char	short	int	long	float	double
byte	int	int	int	int	long	float	double
char	int	int	int	int	long	float	double
short	int	int	int	int	long	float	double
int	int	int	int	int	long	float	double
long	long	long	long	long	long	float	double
float	float	float	float	float	float	float	double
double	double	double	double	double	double	double	double

ตัวอย่าง เช่น ถ้ามีการประกาศตัวแปรดังต่อไปนี้

```
byte b;   short s;    int  i; float f;  double d;
```

ผลลัพธ์ของนิพจน์ตัวอย่างจะได้ดังนี้

นิพจน์	ชนิดของผลลัพธ์
`i * f`	`float`
`i – d`	`double`
`i + b`	`int`
`s / b`	`int`
`i * 0.07`	`double`
`f * 3`	`float`
`s + b`	`int`

พิจารณาตัวอย่าง ข้อความสั่งกำหนดค่าต่อไปนี้

```
s = s +5;      // short บวกกับ int ได้เป็น int
```

```
s = s * b;     // short บวกกับ byte ได้เป็น short
```

ทั้ง ๒ ตัวอย่างนี้จะคอมไพล์ไม่ผ่านเนื่องจากผลลัพธ์ของนิพจน์ทางขวาของเครื่องหมายเท่ากับได้เป็นชนิด `int` แต่ตัวแปร `s` มีประเภทเป็น `short` ซึ่งระดับต่ำกว่า `int` คอมไพเลอร์ไม่ยอมให้นำค่าของนิพจน์ซึ่งมีชนิดสูงกว่าไปใส่ในตัวแปรชนิดที่ต่ำกว่า แต่ตัวอย่างข้อความสั่งกำหนดค่าต่อไปนี้สามารถใช้ได้ คอมไพเลอร์ยอมให้นำค่าของนิพจน์ที่มีชนิดต่ำกว่าไปใส่ในตัวแปรชนิดที่สูงกว่าได้

```
f = s * b;
```

ผลลัพธ์ของนิพจน์ `s * b` เป็นชนิด `short` แต่ตัวแปร `f` เป็นประเภท `float` คอพไพเลอร์จะแปลงผลลัพธ์ของนิพจน์ให้เป็นประเภท `float` ก่อน แล้วจึงนำผลลัพธ์ที่แปลงแล้วไปเก็บที่ตัวแปร `f`

การแปลงผันโดยการบังคับ

เราทราบว่าแล้วว่าจาวามีการแปลงชนิดไปสู่ชนิดที่สูงกว่าให้โดยอัตโนมัติ แต่มีบางกรณีที่เราต้องบังคับให้แปลงชนิดตามที่เราต้องการ พิจารณาตัวอย่างต่อไปนี้

```
int a = 9, b = 2;  float ratio;
ratio = a / b;
```

ผลลัพธ์ที่ได้จากการทำงานของข้อความสั่งนี้คือ `ratio` มีค่า เป็น 4.0 ไม่ใช่ 4.5 เหตุที่เป็นเช่นนี้ก็เพราะว่า นิพจน์ `a / b` ได้ผลลัพธ์เป็น `int` ซึ่งเป็นเลขจำนวนเต็ม ไม่สามารถเก็บเศษของการหารได้ ถึงแม้จะประกาศให้ `ratio` เป็นชนิด `float` ซึ่งเก็บทศนิยมได้ก็ตาม แต่ผลจากการหาค่าของนิพจน์ได้เป็น int ทำให้เศษหายไปก่อนที่จะนำมาใส่ใน `ratio`

ปัญหานี้สามารถแก้ได้โดยการแปลงค่าของ a หรือ b ให้เป็นชนิด `float` ก่อนที่จะนำไปหาร เราสามารถบังคับแปลงชนิดข้อมูลให้เป็นชนิด `float` ได้โดยใช้ตัวดำเนินการ (`float`) ดังตัวอย่างต่อไปนี้

```
ratio = (float)a / b;
```

ดังนั้น นิพจน์นี้จึงเป็นการกระทำระหว่างข้อมูลชนิด `float` กับ `int` ผลลัพธ์ที่ได้เป็นชนิด float ซึ่งสามารถเก็บทศนิยมไว้ได้

การเปลี่ยนชนิดข้อมูลโดยใช้ตัวดำเนินการดังตัวอย่างข้างต้นเรียกว่า*การหล่อ*(cast) เสมือนหนึ่งนำข้อมูลมาหลอมแล้วเทใส่ในเบ้าใหม่เพื่อเปลี่ยนวิธีการจัดเก็บข้อมูล การหล่อนี้ไม่ได้ทำให้ชนิดของตัวแปรเปลี่ยนไป a ยังคงเป็นชนิด int อยู่เหมือนเดิม นำเฉพาะค่าของมันมาแปลงให้เป็น float จึงไม่ส่งผลให้เกิดการเปลี่ยนแปลงใดๆในตัวแปร a

รูปแบบทั่วไปของการหล่อเป็นดังนี้

(ชื่อชนิด) *นิพจน์*

เมื่อ **ชื่อชนิด**เป็นชนิดของข้อมูลที่ต้องการให้เป็น ใช้ชนิดใดชนิดหนึ่งจากชนิดของข้อมูลมาตรฐาน
นิพจน์เป็นนิพจน์ที่ให้ผลลัพธ์เป็นค่าตัวเลข

การหล่อเพื่อแปลงข้อมูลจากชนิดที่สูงกว่าไปเป็นชนิดที่ต่ำกว่าอาจทำให้สูญเสียข้อมูลบางส่วนไปดังนี้

1. หล่อจาก `float` ไปเป็น `int` ทำให้ปัดเศษทศนิยมทิ้งไป
2. การหล่อจาก double ไปเป็น float อาจมีการปัดเลขหลักท้ายทิ้งไป
3. การหล่อจาก long ไปเป็น `int` อาจมีการตัดบิตส่วนหน้า(high order bit)ออกไปเพื่อให้เหลือบิตส่วนท้ายจำนวน ๓๒ บิต ถ้าบิตส่วนหน้าที่ตัดทิ้งไปเป็น 0 ทั้งหมด จะไม่ทำให้ค่าผลลัพธ์เปลี่ยนไป เปลี่ยนแต่ขนาดเท่านั้น แต่ถ้าบิตส่วนหน้าที่ตัดทิ้งไปมีบิตใดบิตหนึ่งมีค่าเป็น 1 จะทำให้ค่าที่ได้ผิดไปจากเดิม

การหล่อสามารถนำไปใช้ในการปัดเศษทิ้งได้ เช่น

```
int i;  float f = 7.6f;
i = (int) f;
```

จะได้ i มีค่าเป็น 7 แต่ถ้าต้องการปัดเศษขึ้น สามารถเขียนเป็นนิพจน์ได้ดังนี้

```
i = (int) ( f + 0.5);
```

จะหาค่า `f + 0.5` ก่อน ได้เป็น 8.1 ต่อจากนั้นจะถูกหล่อให้เป็น `int` จึงเหลือเป็น 8