ชนิดข้อมูลพื้นฐาน และ การดำเนินการที่เกี่ยวข้องกับตัวแปร (Primitive Data Type & Operations)

วัตถุประสงค์

- รู้จักชนิดข้อมูลพื้นฐาน (Primitive Data Type) ของตัวแปร
- รู้จักการประกาศตัวแปร (Variable) และ ตัวแปรคงที่ (Constant)
- รู้จักการเก็บข้อมูลในตัวแปร
- รู้จักความหมายของสัญพจน์ (Literal)
- รู้จักตัวดำเนินการ หรือ เครื่องหมายดำเนินการ (Operator)
- เข้าใจนิพจน์ (Expression) และ ลำดับการดำเนินการ (Precedence Order)
- Type Casting (การแปลงชนิด)
- สามารถเข้าใจโปรแกรมคำนวณอย่างง่าย รับข้อมูลเข้า และ ส่งข้อมูลออกอย่างง่าย ๆ ได้

ส่วนใหญ่ ในการเขียนโปรแกรมภาษาใด ๆ ผู้เขียนโปรแกรมจำต้องประกาศตัวแปร (หรือเรียกว่า จองหน่วยความจำ) เพื่อใช้ในการทดชั่วคราว ซึ่งตัวแปรในภาษา Java มีสองชนิดใหญ่ ดังนี้

1. ชนิดข้อมูลพื้นฐาน (Primitive Data Type) ใช้เก็บค่าพื้นฐานต่าง ๆ เช่น เลขจำนวนเต็ม เลขทศนิยม อักขระ และ ข้อมูลตรรกะ และ

2. ชนิดข้อมูลแบบอ้างอิง (Reference Data Type) ตัวแปรชนิดนี้ไม่ได้เก็บข้อมูลของการคำนวณเหมือนชนิดแรก แต่ถูกใช้เก็บค่าอ้างอิงตำแหน่งในหน่วยความจำ เช่น String และ ข้อมูล array

ในบทนี้เราจะมาพูดถึงข้อมูลชนิดพื้นฐานก่อน

ชนิดข้อมูลพื้นฐาน (Primitive Data Type)

การดำเนินการทางคอมพิวเตอร์อาจประกอบไปด้วยการดำเนินการย่อยมากมาย โปรแกรมอาจมีความจำเป็นต้องเก็บผลลัพธ์ของการดำเนินการย่อยในหน่วยความจำ เพื่อนำมาดำเนินการต่อในขั้นถัดไป นักพัฒนาโปรแกรมสามารถบอก Java ให้จองพื้นที่ในหน่วยความจำเพื่อเก็บข้อมูลได้หลายวิธี หนึ่งในนั้นคือ การประกาศตัวแปรขึ้นมาใช้

ในการประกาศตัวแปรในภาษา Java ผู้เขียนจำเป็นต้องระบุชนิดของตัวแปรทุกครั้ง เพื่อให้ compiler รู้ขนาดพื้นที่หน่วยความจำที่ต้องจอง ชนิดข้อมูลพื้นฐาน (Primitive Data Type) ของตัวแปรมีดังนี้:-

Primitive Data Type	คำอธิบาย	ขนาด [bits]	พิสัยของตัวแปร (ช่วงของค่าที่ตัวแปรแต่ละชนิดสามารถเก็บได้)
boolean	ใช้เก็บค่าจริงหรือเท็จ	8	true หรือ false
byte	ใช้เก็บค่าตัวเลขจำนวนเต็ม	8	-128 (-2^7) ถึง 127 ($2^7 - 1$)
short	ใช้เก็บค่าตัวเลขจำนวนเต็ม	16	– 32,768 (-2^{15}) ถึง 32,767 ($2^{15} - 1$)
int	ใช้เก็บค่าตัวเลขจำนวนเต็ม	32	–2,147,483,648 (-2^{31}) ถึง 2,147,483,647 ($2^{31} - 1$)
long	ใช้เก็บค่าตัวเลขจำนวนเต็ม	64	–9,223,372,036,854,775,808 (2^{63}) ถึง 9,223,372,036,854,775,807 ($2^{63} - 1$)
float	ใช้เก็บค่าตัวเลขทศนิยม	32	พิสัยด้านลบ: -3.40282346638528860e+38 ถึง -1.40129846432481707e-45 พิสัยด้านบวก: 1.40129846432481707e-45 ถึง 3.40282346638528860e+38
double	ใช้เก็บค่าตัวเลขทศนิยม	64	พิสัยด้านลบ: -1.79769313486231570e+308 ถึง -4.94065645841246544e-324 พิสัยด้านบวก: 4.94065645841246544e-324 ถึง 1.79769313486231570e+308
char	ใช้เก็บค่าตัวอักขระหนึ่งตัว	16	0 – 65,535

การประกาศตัวแปร (Variable Declaration)

การประกาศตัวแปร คือ การบอกให้ compiler รู้ว่า ผู้เขียนโปรแกรมต้องการจองพื้นที่หน่วยความจำไว้ใช้งาน และ ให้ compiler รู้ว่า ข้อมูลที่จะนำมาใส่เป็นชนิดใด ไวยากรณ์ (syntax) สำหรับการประกาศตัวแปรสามารถเขียนได้ดังนี้:-

[<variable_modifier>] [scope_modifier] <variable_type> <variable_name> [=initial_value];

ใน syntax ข้างต้น ส่วนที่อยู่ใน < > เป็นส่วนที่จำเป็นสำหรับการประกาศตัวแปร ถ้าไม่มี การประกาศตัวแปรจะไม่เกิดขึ้น ในที่นี้ การประกาศตัวแปรต้องการแค่ variable type และ ชื่อตัวแปร ดังตัวอย่างต่อไปนี้:-

```
int x;          // ประกาศตัวแปร x ชนิด int (integer) ใช้เก็บค่าจำนวนเต็ม
double radius;  // ประกาศตัวแปร radius ชนิด double ใช้เก็บค่าทศนิยม
char a;         // ประกาศตัวแปร a ชนิด char (character) ใช้เก็บค่าอักขระ
```

ส่วนที่มี [ ] ครอบอยู่ บางครั้งไม่มีก็ได้ แต่บางครั้งมันก็มีความสำคัญสำหรับการเขียนโปรแกรม ในที่นี้จะกล่าวถึงคร่าว ๆ เท่านั้น

- variable modifier ใช้ระบุถึงรายละเอียดการเข้าถึงตัวแปร ประกอบด้วย
    - public: ทำให้ทุก object สามารถเข้าถึงตัวแปรได้
    - protected: ทำให้ object สามารถเข้าถึงตัวแปรได้เฉพาะภายในคลาสและ sub class ที่สืบทอดกันมา
    - private: ทำให้ object สามารถเข้าถึงตัวแปรได้เฉพาะภายในคลาสเท่านั้น ไม่รวม sub class
- scope modifier มี 2 ชนิด ได้แก่
    - static: class ที่มีการประกาศตัวแปรแบบ static จะมีคำว่า static ขณะประกาศตัวแปร โดย object ของ class นี้ทุกตัวจะใช้ตัวแปรนี้ร่วมกัน ถ้า object ตัวใดเปลี่ยนค่าในตัวแปรนี้ object ตัวอื่นก็จะเห็นค่าที่ถูกเปลี่ยนในตัวแปรนี้ด้วย ในกรณีที่ผู้พัฒนาโปรแกรมไม่ระบุคำว่า static ตัวแปรจะเป็นแบบ non-static (หรือเรียกอีกแบบว่า instance) แต่ละ object ของ class นี้จะมีตัวแปรชื่อนี้เป็นของตัวเอง การเปลี่ยนค่าในตัวแปรของ object ตัวใด จะไม่ส่งผลต่อค่าในตัวแปรชื่อเดียวกันนี้ของ object อื่น ๆ
    - final: เป็นตัวแปรที่เก็บค่าคงที่ เมื่อตั้งค่าแล้วจะไม่สามารถเปลี่ยนค่าได้ เช่น เมื่อประกาศตัวแปรดังนี้ final double PI = 3.14159; ตัวแปร PI จะเก็บค่าคงที่และไม่สามารถถูกเปลี่ยนแปลงได้อีก

- initial value เป็นค่าตั้งต้นที่จะตั้งให้กับตัวแปรนั้นหลังจากการประกาศเสร็จแล้ว

ตัวอย่างการประกาศตัวแปรพร้อมทั้งการใส่ค่าตั้งต้นอย่างง่าย

double d_var = 10.0;

เป็นการจองหน่วยความจำ 8 bytes (64 บิต) เพื่อใช้เก็บค่าเลขทศนิยม โดยเมื่อประกาศตัวแปรนี้ (หรือจองหน่วยความจำ) เสร็จแล้ว จะตั้งค่าในหน่วยความจำให้เท่ากับ 10.0

การตั้งชื่อตัวแปร

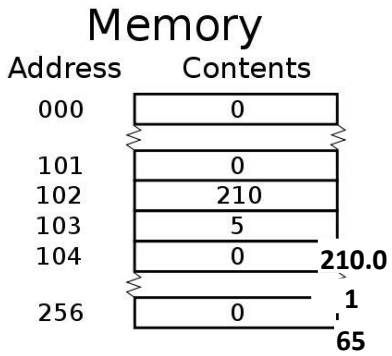
1. ชื่อตัวแปรเริ่มต้นด้วยตัวอักษรเสมอ เริ่มต้นด้วยตัวเลข หรือ $ ไม่ได้
2. ชื่อตัวแปรสามารถประกอบด้วยตัวอักษร ตัวเลข เครื่องหมาย $ หรือ _
3. ชื่อตัวแปรห้ามมีช่องว่าง และตัวพิมพ์เล็กพิมพ์ใหญ่จะต่างกัน
4. เพื่อให้เป็นมาตรฐานเดียวกันและเข้าใจความหมายของชื่อตัวแปรได้ง่าย ชื่อจะเริ่มต้นด้วยตัวพิมพ์เล็ก ถ้าชื่อประกอบด้วยคำหลายคำเมื่อขึ้นคำใหม่อักษรตัวแรกของคำจะใช้ตัวพิมพ์ใหญ่ ตัวต่อมาจะเป็นตัวพิมพ์เล็ก เว้นแต่กรณีที่เป็นค่าคงที่ (final) จะใช้ตัวพิมพ์ใหญ่ทั้งหมดแล้วคั่นระหว่างคำด้วย _ เช่น int studentID; หรือ final int SIZE = 3;
5. ห้ามใช้ true, false หรือ null
6. ห้ามใช้คำสงวน (reserved words) อันได้แก่:-

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

การตั้งค่าข้อมูลให้ตัวแปร

เมื่อประกาศตัวแปรแล้ว เราอาจใส่ค่าเริ่มต้น (initialize) ให้ตัวแปรได้เลย ดังที่ได้อธิบายข้างต้น หรือ ถ้าต้องการเปลี่ยนแปลงค่าภายในตัวแปรภายหลัง ก็สามารถทำได้โดยใช้เครื่องหมายดำเนินการ = (operator กำหนดค่า) ตัวอย่างเช่น เมื่อเราประกาศตัวแปร radius, x และ a แล้ว เราสามารถกำหนดค่าในตัวแปรเหล่านี้ได้อีก ดังตัวอย่าง:-

```
radius = 210.0;      // นำค่า 210.0 ไปใส่ไว้ในหน่วยความจำสำหรับตัวแปรชื่อ radius
                     // (หน่วยความจำที่ 102 ในรูปด้านล่าง)
x = 1;               // นำค่า 1 ไปใส่ไว้ในหน่วยความจำสำหรับ x
                     // (หน่วยความจำที่ 103 ในรูปด้านล่าง)
a = 'A';             // แปลงอักขระ A เป็นรหัสตัวเลข ASCII (65) แล้ว, นำไปใส่ไว้ใน
                     // หน่วยความจำที่ 104 สำหรับตัวแปรชื่อ a
```



ข้อควรระวัง 1 เราจะตั้งค่าข้อมูลให้ตัวแปรใด ๆ ได้ก็ต่อเมื่อ เราได้ประกาศตัวแปรนั้น ๆ แล้ว !!!!

ข้อควรระวัง 2 การใช้ตัวดำเนินการ = เป็นการบอกให้ Java นำค่าที่อยู่ทางขวามือของเครื่องหมาย = ไปใส่ไว้ในตัวแปร (หน่วยความจำ) ด้านซ้ายมือของเครื่องหมาย = ดังนั้น Java จะแจ้ง error ถ้าเราใช้คำสั่ง 1 = x; เนื่องจาก 1 ไม่ใช่ตัวแปรที่จะมารับค่าในตัวแปร x

ข้อควรระวัง 3 ดังได้อธิบายแล้วในเรื่องชนิดของข้อมูลพื้นฐาน ตัวแปรทุกตัวจะสามารถเก็บค่าที่อยู่ในช่วงที่ถูกต้องเท่านั้น เนื่องจากเนื้อหน่วยความจำของแต่ละตัวแปรนั้นมีขนาดจำกัด นักเขียนโปรแกรมต้องพึงระวังเรื่องขนาดของตัวแปรให้ดี เพื่อป้องกันการเกิด bug ที่สามารถเกิดได้ง่าย ๆ

ข้อควรระวัง 4 การตั้งค่าให้ตัวแปรมีข้อควรระวังอีกข้อคือ ชนิดของค่าเริ่มต้นจะต้องเป็นชนิดเดียวกันกับตัวแปรนั้น หรือ ชนิดของค่านั้นต้องเป็นชนิดที่สามารถถูกเปลี่ยน (cast) ให้เป็นชนิดเดียวกับชนิดของตัวแปรได้ (รายละเอียดเกี่ยวกับ casting จะกล่าวภายหลัง) เช่น int my_int_var = 10.0; ไม่ถูกต้อง เพราะ my_int_var เป็นชนิด int แต่ 10.0 เป็นชนิด double compiler จะแจ้งว่ามีความผิดพลาดขึ้น

*ดูรายละเอียดเพิ่มเติมเกี่ยวกับการจัดเก็บจำนวนเต็มรวมทั้งความรู้พื้นฐานเลขฐานสอง ได้ที่ www.informatics.buu.ac.th/~jira/886200/Lecture3.pdf

สัญพจน์ (literal)

สัญพจน์ (literal) เป็นส่วนหนึ่งของ source code ที่มีค่าตามตัวอักษร (Java ไม่ต้องคำนวณหรืออ่านมาจากหน่วยความจำ) สัญพจน์ต่างจากตัวแปรตรงที่ compiler ต้องตีความค่าของตัวแปรก่อนการดำเนินการ ตัวอย่างของ literal เช่น true, 'C' หรือค่าตัวเลขต่าง ๆ เช่น 100, 10000, หรือ 5.0 ซึ่ง literal นี้สามารถถูกนำมากำหนดค่าให้กับตัวแปรต่าง ๆ ได้ดังตัวอย่างด้านล่าง:

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short x = 10000;
double d = 5.0;
```

สัญพจน์ในตัวอย่างข้างต้นมีหลายชนิดได้แก่:-

- สัญพจน์ชนิดตรรกะแบบบูล (Boolean literal) คือ ค่าทางตรรกะ true หรือ false
- สัญพจน์ชนิดตัวอักขระ (Character liter) คือ ค่าที่เป็นตัวอักขระหนึ่งตัว โดยใช้เครื่องหมาย ' ' กำกับ เช่น สัญพจน์ตัวอักขระ C จะต้องถูกเขียนเป็น 'C'
- สัญพจน์ชนิดตัวเลข (Number Literals) คือค่าคงที่ที่ปรากฏใน source code เช่น 100, 10000 ที่อยู่ในตัวอย่างข้างต้นเป็น integer literal (สัญพจน์ที่เป็นจำนวนเต็ม) ส่วน 5.0 เป็น floating-point literal (สัญพจน์ที่เป็นจำนวนทศนิยม)

ข้อควรระวัง 1 เมื่อเราใช้ System.out.println(x); เนื่องจาก x เป็นตัวแปรไม่ใช่สัญพจน์ Java จะนำค่าที่เก็บไว้ใน x ออกมาแสดงผลทางหน้าจอ หากเราต้องการระบุให้ Java แสดงตัวอักษร x ออกทางหน้าจอ เราต้องใช้คำสั่งต่อไปนี้แทน System.out.println("x"); หรือ System.out.println('x');

นั่นคือ ถ้าเราพิมพ์ตัวเลข, true, false, ตัวอักษรตัวเดียวมี ' ' กำกับ, หรือ สายอักขระมี " " กำกับ Java จะมองว่าเป็นสัญพจน์ (และจะตีความตามตัวอักษร) แต่ ถ้าเราพิมพ์ตัวอักษรหรือสายอักขระที่ไม่มี ' ' หรือ " " กำกับ Java จะมองว่า เราหมายถึงตัวแปรและจะต้องนำค่าในตัวแปรออกมาก่อนประมวลผลต่อไป

ข้อควรระวัง 2 ถ้าเราพยายามตั้งค่า literal ที่มากเกินกว่าที่ตัวแปรจะเก็บได้ เราจะได้รับ error ตอน compile เช่นตัวอย่างต่อไปนี้ byte b = 1000;

Java จะให้สัญพจน์ที่เป็นจำนวนเต็ม (integer literal) ทุกตัวเป็นชนิด int โดยอัตโนมัติ ไม่ใช่ชนิด byte, short หรือ long ถ้าเราอยากให้สัญพจน์ที่เป็นจำนวนเต็มเป็นชนิด long ให้เติม L หรือ l ต่อท้ายสัญพจน์นั้น (ข้อแนะนำ: ใช้ L ดีกว่าเพราะ l คล้าย I และ 1 เกินไป)

Java จะให้สัญพจน์ที่เป็นจำนวนทศนิยม (floating-point literal) ทุกตัวเป็นชนิด double โดยอัตโนมัติ ไม่ใช่ชนิด float ถ้าเราอยากให้สัญพจน์ที่เป็นจำนวนทศนิยมเป็นชนิด float ให้เติม F หรือ f ต่อท้ายสัญพจน์นั้น

เช่น Java จะตีความว่าสัญพจน์ 5.0 เป็นชนิด double ถ้าเราจะทำให้ Java ตีความว่าเป็น float ให้เติม f หรือ F ต่อท้าย ดังนี้ 5.0f หรือ 5.0F (ในทำนองเดียวกัน เราใช้ d หรือ D ต่อท้ายสัญพจน์เพื่อให้ Java ตีความสัญพจน์ให้เป็นชนิด double ดังนี้ 100.2d หรือ 100.2D)

สัญพจน์ที่เป็นจำนวนทศนิยม (floating-point literal) สามารถถูกเขียนได้แบบวิทยาศาสตร์ เช่น 1.23456e+2 หรือ 1.23456e2 ซึ่งหมายถึง 123.456 และ 1.23456e-2 ซึ่งหมายถึง 0.0123456 (จะใช้ e หรือ E ก็ได้)

ตัวดำเนินการ หรือ เครื่องหมายดำเนินการ (operator)

ตัวดำเนินการอาจถูกแบ่งได้เป็นหลายประเภท ดังนี้:-

1. ตัวดำเนินการตัวเลข (Numeric Operators)

ชื่อ	ความหมาย	ตัวอย่าง	ผลลัพธ์
+	การบวก	34 + 1	35
-	การลบ	34.0 – 0.1	33.9
*	การคูณ	300 * 30	9000
/	การหาร	1.0 / 2.0	0.5
%	การหารหาเศษ	20 % 3	2

ในตัวอย่างการคำนวณเลขข้างต้น แต่ละตัวอย่างมีการใช้ตัวถูกดำเนินการ (operand) 2 ตัว เช่น 34 +1 (+ เป็น ตัวดำเนินการ (operator), 34 และ 1 คือ ตัวถูกดำเนินการ (operand)) เราเรียก operator ที่มี 2 operands แบบนี้ว่า Binary operator โดยชนิดของ operands ทั้งสองจะเป็น byte, short, int, long, float หรือ double ก็ได้

บางครั้ง operator + และ – ถูกใช้กับ operand แค่ตัวเดียวได้ (เรียกว่า Unary Plus และ Unary Minus) เช่น –a จะให้ค่าติดลบของค่าที่อยู่ในตัวแปร a

<u>ข้อควรระวัง</u> ชนิดของผลลัพธ์ของการดำเนินการตัวเลขจะเป็นชนิดเดียวกับชนิดของ operand ที่ใช้หน่วยความจำใหญ่กว่า ตัวอย่างเช่น

- 5 / 2 จะให้ผลลัพธ์เป็น 2 ไม่ใช่ 2.5 เนื่องจาก operand 5 และ 2 ต่างก็เป็น literal ที่เป็น integer ดังนั้นชนิดของผลลัพธ์ที่ได้จะเป็น 2 ซึ่งเป็น integer แทนที่จะเป็น 2.5 ซึ่งเป็น double
- 5.0 / 2 จะให้ผลลัพธ์เป็น 2.5 เนื่องจาก operand 5.0 เป็น literal ชนิด double แม้ 2 จะเป็น literal ชนิด int ชนิดของผลลัพธ์ที่ได้จะเป็น double ด้วย (สังเกตว่า double ใช้หน่วยความจำมากกว่า integer)

การหารหาเศษโดยการใช้ % (Remainder Operator, ส่วนใหญ่อ่านว่า mod) ถูกพบบ่อยมากในการเขียนโปรแกรม เช่น

- ในกรณีการตรวจสอบว่าตัวเลขเป็นเลขคู่ (หาร 2 ลงตัว) หรือไม่ เรามักจะนำตัวเลขนั้นมาหารแบบหาเศษ (mod) ถ้าได้ผลลัพธ์เป็น 0 แสดงว่าเลขนั้นเป็นเลขคู่
- สมมตว่า วันนี้เป็นวันเสาร์แล้วเราอยากรู้ว่าอีก 10 วันเป็นวันอะไร เราสามารถหาได้โดยใช้นิพจน์ด้านล่างนี้: (6 + 10) % 7 ได้ 2 โดยวันเสาร์ถูกแทนด้วย 6 สัปดาห์หนึ่งมีแค่ 7 วัน ดังนั้นหลังจากวันเสาร์ไป 10 วัน คือ 2 (ซึ่งใช้แทนวันอังคาร)
- ในกรณีที่เราต้องการแยกตัวเลขแต่ละหลักออกมา เราสามารถทำได้โดยการใช้ % และ / สลับกัน เช่น ถ้าเรารับ 1234 มา แล้วเราอยากแยกมันออกมาเป็น 1, 2, 3, และ 4 เพื่อดำเนินการต่อ เราทำได้ดังนี้:-

    --> 1234 % 10 ได้ผลลัพธ์ 4

    --> 1234 / 10 ได้ผลลัพธ์ 123

    --> 123 % 10 ได้ผลลัพธ์ 3

    --> 123 / 10 ได้ผลลัพธ์ 12

    --> 12 % 10 ได้ผลลัพธ์ 2

    --> 12 / 10 ได้ผลลัพธ์ 1

    --> 1 % 10 ได้ผลลัพธ์ 1

2. ตัวดำเนินการเพื่อการตั้งค่า (Assignment Operators)

การตั้งค่าให้กับตัวแปรสามารถทำได้โดยใช้เครื่องหมาย = ซึ่งค่าที่อยู่ทางขวามือของ = จะถูกนำมากำหนดให้กับตัวแปรทางซ้ายมือ และ ถ้าคำสั่งในการกำหนดค่าเป็นการนำตัวแปรที่อยู่ทางซ้ายมือมาดำเนินการร่วมด้วย เราสามารถเขียนในรูปย่อได้ ดังต่ออย่างต่อไปนี้

ตัวดำเนินการ	ตัวอย่างอย่างย่อ	ตัวอย่างแบบเต็ม
=	i = 8	i = 8
+=	i += 8	i = i + 8
-=	i -= 8	i = i - 8
*=	i *= 8	i = i * 8
/=	i /= 8	i = i / 8
%=	i %= 8	i = i % 8

3. ตัวดำเนินการเพิ่มและลด (Increment and Decrement Operators) มีการใช้งานมากในโครงสร้างควบคุมการทำงานชนิดทำซ้ำ

ตัวดำเนินการ	ชื่อ
++ (var)	prefix increment
-- (var)	prefix decrement
(var) ++	postfix increment
(var)--	postfix decrement

increment ++ เพิ่มค่าของตัวแปรขึ้น 1 decrement -- ลดค่าของตัวแปรลง 1 operator ทั้งสองนี้มีการใช้งาน 2 รูปแบบ คือใช้หน้าตัวแปร (prefix) และใช้หลังตัวแปร (postfix) ซึ่งมีความหมายต่างกัน ขอให้ศึกษาการใช้งานจากตัวอย่างต่อไปนี้

- แบบ prefix ปรับค่าของตัวแปรก่อน แล้วจึงนำค่าของตัวแปรนั้นไปใช้

```
int a = 9, b;
b = ++a;
```

เป็นการเพิ่มค่าของตัวแปร a ขึ้นอีก 1 ดังนั้นค่าของตัวแปร a จึงเปลี่ยนเป็น 10 จากนั้นจึงค่าของตัวแปร a ไปกำหนดให้แก่ตัวแปร b ดังนั้นตัวแปร b จึงมีค่าเป็น 10 สรุปว่าเมื่อทำงานเสร็จค่าของตัวแปร a = 10 และตัวแปร b = 10

- แบบ postfix นำค่าของตัวแปรนั้นไปใช้ก่อน แล้วจึงปรับค่าของตัวแปร

```
int     a = 9, b;
        b = a++;
```

เป็นการนำค่าปัจจุบันของตัวแปร a ไปกำหนดให้แก่ตัวแปร b เนื่องค่าปัจุบันของตัวแปร a มีค่าเป็น 9 ดังนั้นตัวแปร b จึงมีค่าเป็น 9 ด้วย จากนั้นจึงทำการเพิ่มค่าของตัวแปร a ขึ้นอีก 1 ทำให้ตัวแปร a มีค่าเป็น 10 สรุปว่าเมื่อทำงานเสร็จค่าของตัวแปร a = 10 และตัวแปร b = 9

4. ตัวดำเนินการเกี่ยวกับการเปรียบเทียบ (Comparison Operators) หลังจากการเปรียบเทียบแล้ว จะให้ผลลัพธ์เป็น true หรือ false (รายละเอียดจะกล่าวถึงภายหลัง)

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

5. ตัวดำเนินการเกี่ยวกับตรรกะแบบบูล (Boolean Operators) หลังจากการดำเนินการแล้ว จะให้ผลลัพธ์เป็น true หรือ false (รายละเอียดจะกล่าวถึงภายหลัง)

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

นิพจน์ (expression)

เรารู้จัก ตัวแปร และ ตัวดำเนินการ (operators) กันแล้ว ต่อมาเราจะมาเรียนเกี่ยวกับเรื่องนิพจน์ (expression) รวมทั้ง เพิ่มความเข้าใจเกี่ยวกับ statement กับ block ให้มากขึ้น

โครงสร้างของ expression ประกอบด้วย ตัวแปร, operator, และ การเรียก method ซึ่งจะเป็นไปตามไวยากรณ์ (syntax) ของภาษาจาวา

expression ที่เขียนถูกไวยกรณ์จะถูก Java ตีความออกมาเป็นค่าค่าหนึ่ง โดยที่ค่านี้อาจเป็นชนิดใดก็ได้ (boolean, char, int, ... ฯลฯ) ขึ้นกับ expression นั้น เช่น 1 * 2 * 3 จะถูก Java คำนวณและตีค่าเป็น 6 ซึ่งเป็นชนิด int

<u>ข้อควรระวัง</u> ลำดับการเขียนของ ตัวแปร, operator และ method มีความสำคัญ ถ้าทำไม่ชัดเจน เราอาจได้ผลลัพธ์ที่ไม่ต้องการได้ เช่น expression 2 + 6 / 4 นั้นไม่ชัดเจนว่า เราอยากจะให้ Java บวกหรือหารก่อน

เราสามารถใช้ () ระบุให้ชัดเจนว่าจะให้ Java ทำส่วนไหนก่อน เช่น (2 + 6) / 4 เป็นการระบุให้ Java ทำการดำเนินการบวกก่อน และ 2 + (6 / 4) เป็นการระบุให้ Java ทำการดำเนินการหารก่อน

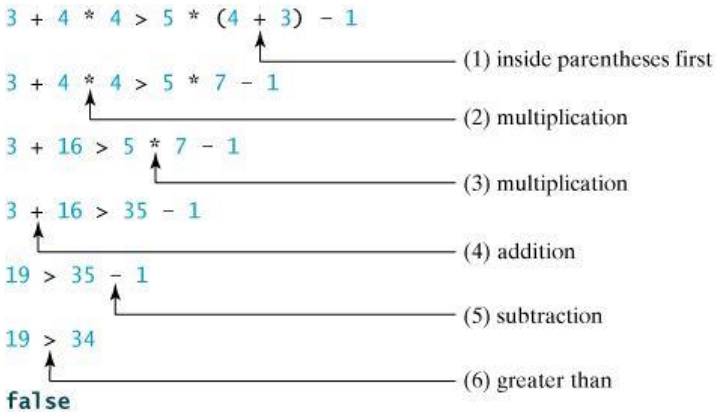
<u>ลำดับความสำคัญ (Precedence Order)</u>

ถ้าผู้เขียนโปรแกรมไม่ได้ระบุลำดับการดำเนินการใน expression โดยใช้ ( ) แล้ว Java จะไม่แจ้ง error แต่จะดำเนินการตาม**กฎการทำก่อน** (Precedence, เป็นลำดับการดำเนินการก่อนหลัง คล้ายกับลำดับการดำเนินการทางคณิตศาสตร์) ดังนี้:-

1) นิพจน์ที่อยู่ในวงเล็บในสุดจะถูกตีค่าก่อน
2) การตีค่าจะต้องเลือกทำตัวดำเนินการตามลำดับที่แสดงในตารางด้านล่าง โดย operator ที่อยู่ในลำดับบนของตารางจะถูกดำเนินการก่อน operator ที่อยู่ลำดับต่ำกว่า
3) ในกรณีที่ expression หนึ่งมี operator ลำดับเดียวกันหลายตัว Java จะเลือกทำจาก operator ตัวซ้ายสุดก่อน เช่น a – b + c – d จะถูกดำเนินการเหมือน ((a – b) + c) – d โดยการดำเนินการต้องคำนึงถึง Associativity ด้วยว่าจะทำจากซ้ายไปขวา หรือ ขาวไปซ้าย

Operator Precedence			
Operators	**Expression**	**Precedence**	**Associativity**
วงเล็บ, ดัชนีของ array, การเลือกสมาชิก	( ) [ ] .	*Highest*	ซ้ายไปขวา
postfix	*var*++ *var*--		ขวาไปซ้าย
prefix, unary plus/minus, นิเสธ, คอมพลีเมนต์บิต, การ cast ชนิด	++*var* --var +*expr* -*expr* ~var !var *(type) expr*		ขวาไปซ้าย
การคูณ การหาร	* / %		ซ้ายไปขวา
การบวก การลบ	+ -		ซ้ายไปขวา
การเลื่อนบิต	<< >> >>>		ซ้ายไปขวา
ความสัมพันธ์	< > <= >= instanceof		ซ้ายไปขวา
การตรวจสอบความเท่ากัน	== !=		ซ้ายไปขวา
bitwise AND	&		ซ้ายไปขวา
bitwise exclusive OR	^		ซ้ายไปขวา
bitwise inclusive OR	|		ซ้ายไปขวา
logical AND	&&		ซ้ายไปขวา
logical OR	||		ซ้ายไปขวา
Ternary	? :		ขวาไปซ้าย
การกำหนดค่า	= += -= *= /= %= &= ^= |= <<= >>= >>>=	*Lowest*	ขวาไปซ้าย

ตัวอย่างการคำนวณตามลำดับ precedence



ความแตกต่างระหว่าง Statement กับ Expression

เราได้รู้ความหมายของ statement ไปแล้วในบทที่แล้ว นั่นคือ statement จะถูกใช้แทน action หรือการกระทำ ที่ผู้เขียนตั้งใจจะให้โปรแกรมทำ (execute) ซึ่งทุก ๆ statement จะจบลงด้วยเครื่องหมาย ; แต่เราจะมาเรียนรู้รายละเอียดเพิ่มอีกนิดเพื่อจะได้ทราบความแตกต่างระหว่าง statement กับ expression

เราสามารถกล่าวได้ว่า statement เหมือนประโยคทั่ว ๆ ไปในภาษาพูด แต่ในภาษา Java แล้ว statement จะถูกเขียนแทนการกระทำการทางคอมพิวเตอร์ที่สมบูรณ์ (complete execution) แบ่งเป็นรูปแบบต่าง ๆ ดังนี้

1. *expression statement*: บางครั้ง statement ก็ประกอบไปด้วย expression แล้วตามด้วย ; เท่านั้น เช่น
    - aVar = 89; // ซึ่งก็คือ assignment expression ตามด้วย ;
    - aVar++; // ซึ่งก็คือ increment expression ตามด้วย ;
    - System.out.println(“Hello”); // ซึ่งก็คือ การเรียก method ตามด้วย ;
    - Bicycle myBike = new Bicycle(); // ซึ่งก็คือ การสร้าง object ตามด้วย ; (รายละเอียดจะอธิบายภายหลัง)
2. *declaration statement*: เช่น int aVar; หรือ int aVar = 89;

3. *control flow statement*: เป็น statement ที่ควบคุมลำดับการทำงานของโปรแกรม โดยใช้คำสั่งทางเลือก (if, if-else, switch) หรือ สั่งให้โปรแกรมทำงานแบบทำซ้ำ (loop for, while, do-while) หรือ สั่งให้หยุดทำงานและไปทำงานอีกที่หนึ่ง (break, continue, return)

Blocks เป็นกลุ่มของ statement ตั้งแต่ 0 statement ขึ้นไปที่อยู่ระหว่าง { } ซึ่งเราสามารถใช้ block ได้ทุก ๆ แห่งที่ Java ยอมให้ใส่ statement เราสร้าง Block เพื่อกำหนดขอบเขต (scope) ของการใช้งานตัวแปร และ/หรือ กำหนดขอบเขตของการใช้คำสั่งต่าง ๆ ที่อยู่ภายใต้คำสั่งทางเลือก หรือภายใต้คำสั่งทำซ้ำ

ตัวอย่างของ block

```
class BlockDemo {                          <---------------------- begin block class
    public static void main(String[] args) {   <---------------------- begin block method
        boolean condition = true;
        if (condition) {                       <---------------------- begin block 1
            System.out.println("Condition is true.");
        }                                      <---------------------- end block 1
        else {                                 <---------------------- begin block 2
            System.out.println("Condition is false.");
        }                                      <---------------------- end block 2
    }                                          <---------------------- end block method
}                                              <---------------------- end block class
```

การแปลงชนิดตัวเลข (Numeric Type Conversion)

เมื่อเราใช้ตัวดำเนินการแบบ binary (มีตัวถูกดำเนินการ 2 ตัว เช่น x + y, a * b) แต่ตัวถูกดำเนินการทั้ง 2 มีชนิดต่างกัน, Java จะแปลงชนิดของตัวถูกดำเนินการโดยอัตโนมัติ (การแปลงชนิดของข้อมูลนี้ ผู้เขียนโปรแกรมไม่ต้องแปลงเองเรียกว่า implicit casting) ตามกฎการแปลง (Conversion Rules) ต่อไปนี้:

1. ถ้าตัวถูกดำเนินการตัวใดตัวหนึ่งเป็นชนิด double, อีกตัวก็จะถูกแปลงเป็น double
2. ถ้าไม่ใช่ข้อข้างต้น, ถ้าตัวถูกดำเนินการตัวใดตัวหนึ่งเป็นชนิด float, อีกตัวก็จะถูกแปลงเป็น float
3. ถ้าไม่ใช่ข้อข้างต้น, ถ้าตัวถูกดำเนินการตัวใดตัวหนึ่งเป็นชนิด long, อีกตัวก็จะถูกแปลงเป็น long
4. ถ้าไม่ใช่ข้อข้างต้น, ตัวถูกดำเนินการทั้งสองจะถูกแปลงเป็น int

พิจารณาตัวอย่าง statement ต่อไปนี้:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

ตามที่ได้กล่าวมาแล้ว Java มองสัญพจน์ที่เป็นตัวเลขจำนวนเต็มทุกตัวเป็นชนิด int ดังนั้น 100 จะถูกมองว่าเป็นชนิด int ก่อนถูกนำไปกำหนดค่าให้ i, แต่ i เป็นตัวแปรชนิด byte ดังนั้นก่อนที่ Java จะนำค่า 100 ไปใส่ในตัวแปร i ได้ Java ต้องทำการแปลงชนิดของตัวเลข 100 นี้ให้เป็น byte ก่อน

ในบรรทัดที่สอง Java จะทำการคำนวณค่าทางขวามือของ = ให้เสร็จก่อนการกำหนดค่าให้ตัวแปร k ที่อยู่ด้านซ้ายมือของเครื่องหมาย = โดยการคำนวณค่าดังกล่าว Java จะต้องแปลงชนิดของค่าในตัวแปร i จาก byte เป็น int ก่อน จึงสามารถนำมาคูณกับสัญพจน์ตัวเลข 3 (ชนิด int) ได้ ผลลัพธ์จากการคูณคือ 300 (ชนิด int) จากนั้น Java ทำการบวก 300 ด้วย 4 ได้ผลลัพธ์ 304 (ชนิด int) ลำดับสุดท้าย Java ต้องทำการเปลี่ยนชนิด int ของ 304 ให้เป็นชนิด long ก่อนจึงค่อยนำ 304 ไปกำหนดค่าให้ k (ชนิด long)

ในบรรทัดที่สาม Java ทำขั้นตอนดังนี้ 1) เปลี่ยนชนิดของค่าในตัวแปร i จาก byte ให้เป็น double แล้วนำไปคูณกับ 3.1 (ชนิด double) ได้ผลลัพธ์เป็น 310.0 (ชนิด double) 2) เปลี่ยนชนิดของตัวเลข 2 จาก int เป็น long แล้วนำ 2 ไปหาร k ได้ผลลัพธ์เป็น 152 (ชนิด long) 3) เปลี่ยนชนิดของ 152 จาก long เป็น double ก่อนที่จะนำไปบวกกับ 310.0 ได้ผลลัพธ์การบวกเป็น 462.0 (ชนิด double) 4) กำหนดค่า 462.0 (ชนิด double) ให้ตัวแปร d

การแปลงชนิดของข้อมูลชนิดต่าง ๆ (Type Casting)

นอกจากการแปลงชนิดข้อมูลที่เป็นตัวเลขแล้ว Java ยังยอมให้มีการแปลงชนิดข้อมูลชนิดต่าง ๆ อีก โดยการแปลงชนิดข้อมูลใน Java มีสองแบบ

1. การแปลงโดยปริยาย (Implicit casting) ในการแปลงชนิดนี้ โปรแกรมไม่ต้องสั่งใน Java ทำการแปลง เพราะ Java ฉลาดพอที่จะแปลงชนิดของข้อมูลให้อัตโนมัติ เช่น

```
double d = 3; // Java จะแปลงชนิด int ของ 3 ให้กลายเป็นชนิด double อัตโนมัติ
```

2. การแปลงโดยโปรแกรมเมอร์ตั้งใจ (Explicit casting) บางครั้ง การแปลงชนิดข้อมูลอาจทำให้เสียความแม่นยำได้ Java จึงไม่ทำการแปลงชนิดของข้อมูลให้อัตโนมัติ โปรแกรมเมอร์ต้องระบุถึงการแปลงชนิดข้อมูลให้ชัดเจน เช่น

```
int i = 3.1; // Java จะแจ้ง error เนื่องจาก Java ไม่แปลง 3.1 (ชนิด double) ให้
             //    เป็น 3 (ชนิด int) ให้โดยอัตโนมัติ เพราะ การแปลงค่าทำให้สูญเสีย
             // ความแม่นยำของข้อมูล
```

ถ้าโปรแกรมต้องการระบุให้ Java แปลงชนิดข้อมูลในคำสั่งข้างต้น โปรแกรมเมอร์ต้องพิมพ์ (int) นำหน้าข้อมูลที่อยากจะให้ Java แปลงชนิดข้อมูลให้ ดังนี้

```
int i = (int) 3.1; // โปรแกรมเมอร์ระบุให้ Java แปลง 3.1 ให้เป็น 3 โดยที่ Java จะ
                   // แปลงชนิดของข้อมูลให้โดยไม่ฟ้อง error เนื่องจาก โปรแกรมเมอร์
                   // ระบุบอก Java อย่างชัดแจ้ง ด้วยคำสั่งการ cast (int)
```

ชนิดข้อมูลตัวอักขระ (Character Data Type)

หน่วยความจำสามารถเก็บข้อมูลเป็นตัวเลขฐานสองเท่านั้น ดังนั้น หาก Java ต้องการเก็บข้อมูลที่เป็นตัวอักขระ เช่น อักษร A-Z Java จะต้องทำการเข้ารหัสอักษรเหล่านั้นให้เป็นตัวเลขก่อน ซึ่งการเข้ารหัสที่ Java ใช้มีสองรูปแบบ คือ รูปแบบของรหัส ASCII และ รูปแบบมาตรฐาน Unicode ตัวอย่างด้านล่างแสดงการประกาศและกำหนดค่าตัวแปรชนิดอักขระ

```
char letter1 = 'A';
char numChar1 = '5';
char letter2 = '\u0041';
char numChar2 = '\u0034';
```

การประกาศตัวแปร 4 ตัวข้างต้น เป็นการประกาศตัวแปร (หรือจองหน่วยความจำ) เพื่อใช้เก็บข้อมูลชนิดตัวอักษร (ชนิด char) การกำหนดค่าให้ตัวแปร letter1 และ numChar1 เป็นการกำหนดค่าในรูปแบบรหัส ASCII ผู้เขียนโปรแกรมต้องใส่ตัวอักษรในสัญลักษณ์ ' ' (สัญลักษณ์ฝนทอง, หรือ single quote) เพื่อระบุตัวอักษรที่จะ

นำมากำหนดค่าให้ตัวแปรในตัวอย่างข้างต้น ตัวแปร letter1 เก็บค่าตัวเลขที่ใช้แทนอักษร A ส่วนตัวแปร numChar1 เก็บค่าตัวเลขที่ใช้แทนอักษร 5 (ข้อสังเกตุ: numChar1 ไม่ได้เก็บค่าเลข 5 แต่เก็บค่าตัวเลขที่ใช้แทนอักษร 5)

ส่วนการกำหนดค่าให้ตัวแปร letter2 และ numChar2 เป็นการกำหนดค่าในรูปแบบ Unicode ผู้เขียนโปรแกรมต้องใส่ตัวเลขรหัสมาตรฐานเป็นตัวเลขฐานสิบหก (ค่า 0000 ถึง FFFF) ตามหลัง \u ในสัญลักษณ์ ' ' เพื่อระบุตัวอักษรที่จะนำมากำหนดค่าให้ตัวแปรในตัวอย่างข้างต้น ตัวแปร letter2 เก็บค่าตัวเลขรหัส Unicode \u0041 ที่ใช้แทนอักษร A ส่วนตัวแปร numChar2 เก็บค่าตัวเลขรหัส Unicode ที่ใช้แทนอักษร 5 (ข้อสังเกตุ: numChar2 ไม่ได้เก็บค่าเลข 5 แต่เก็บค่าตัวเลขรหัส Unicode \u0034 ที่ใช้แทนอักษร 5)

จริง ๆ แล้วรหัส ASCII เป็นชุดรหัสที่เป็น subset ของรหัส Unicode โดยรหัส ASCII มีค่าตั้งแต่ 0 ถึง 128 หรือ ค่าตั้งแต่ \u0000 ถึง \u007f ในรหัส Unicode (สาเหตุที่ Java ใช้ชุดรหัส Unicode เพราะ Unicode สามารถครอบคลุมรหัสของตัวอักษรภาษาต่าง ๆ และ สัญลักษณ์ต่าง ๆ ได้มากกว่า ASCII มาก) ตารางด้านล่างแสดงสัญลักษณ์และตัวอักขระในรหัส ASCII ในรูปแบบเลขฐานสิบ

TABLE B.1 ASCII Character Set in the Decimal Index

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

ก่อนที่จะขึ้นเรื่องข้อมูลสายอักขระ เรามาดูเรื่องการเปลี่ยนชนิดของตัวแปรอีกครั้ง ดังได้อธิบายก่อนหน้านี้ว่า Java จำเป็นต้องเปลี่ยนตัวอักษรให้เป็นรหัสตัวเลขก่อนที่จะกำหนดค่าให้ตัวแปรชนิด char ได้ เช่นในการดำเนินการคำสั่ง char letter1 = 'A'; Java ต้องเปลี่ยนอักขระ A ให้เป็นเลข 65 (เท่ากับ /u0041 ในเลขฐานสิบหก) ก่อนที่จะกำหนดค่าให้ตัวแปร letter1 ขั้นตอนที่ว่านี้ก็เป็น Implicit casting คือ การเปลี่ยนชนิดของ literal ที่เป็นชนิดตัวอักษร (char) ให้เป็นชนิดเลขจำนวนเต็ม (int) โดยที่ Java ทำให้อัตโนมัติ ซึ่งการทำ Implicit casting โดย Java นี้ให้ผลลัพธ์เช่นเดียวกับเมื่อโปรแกรมเมอร์ใช้ statement ต่อไปนี้:-

char letter1 = 65; หรือ char letter1 = (char) 65;

ในทำนองเดียวกัน Java จะทำ Implicit casting จาก char มาเป็น integer ให้อัตโนมัติ นั่นคือ Java จะไม่แจ้ง error สำหรับ statement นี้ int i = 'A'; ซึ่งเหมือนกับ statement int i = (int) 'A';

ชนิดข้อมูลสายอักขระ (String Type)

ตัวแปรชนิด char เก็บข้อมูลได้แค่หนึ่งตัวอักขระ ถ้าเราจะเก็บข้อมูลที่เป็นสายอักขระเราต้องใช้ตัวแปรชนิด String เช่น String message = "Welcome to Java";

ข้อสังเกตุ 1: ข้อมูลสายอักขระต้องอยู่ในเครื่องหมาย " " (ฟันหนู หรือ double quote) ในขณะที่ ข้อมูลอักขระเดียวต้องอยู่ในเครื่องหมาย ' ' (ฝนทอง หรือ single quote)

ข้อสังเกตุ 2: บางครั้งสายอักขระอาจมีตัวอักษรเดียว เราอาจเห็น expression ลักษณะนี้ "A" ซึ่งไม่ได้ผิดอะไร โดย แทนที่ "A" จะหมายถึงข้อมูลอักขระ A (ชนิด char) แต่หมายถึงข้อมูลสายอักขระที่มีอักษร A ตัวเดียว (ชนิด string)

Java อำนวยความสะดวกให้โปรแกรมเมอร์ในกรณีที่ โปรแกรมเมอร์อยากนำสายอักขระมาต่อกัน (Concatenate) โดยให้ใช้ operator + เป็นตัวเชื่อม เช่น statement ต่อไปนี้:-

String message = "Chapter " + 2;

เมื่อ Java พบ operator + ถูกใช้กับตัวถูกดำเนินการสองชนิด (โดยหนึ่งในนั้นเป็นชนิด string) Java จะฉลาดพอ และ ตีความโดยอัตโนมัติว่า โปรแกรมเมอร์ต้องการต่อสายอักขระ Chapter กับ เลข 2 ให้กลายเป็น "Chapter 2" ดังนั้น แทนที่ Java จะนำค่า Chapter ไปบวก 2 Java จะทำ Implicit casting เปลี่ยนชนิดของ 2 (ชนิด int) ให้เป็นชนิด String "2" จากนั้นจะทำการต่อสายอักขระทั้งสอง

เกร็ดความรู้: Java มีคำสั่ง Integer.parseInt() และ Double.parseDouble() ช่วยแปลง string ให้เป็น integer และ double ตามลำดับ ตัวอย่างการใช้มีดังนี้:-

```
int intValue = Integer.parseInt(“123”);
double doubleValue =Double.parseDouble(“123.45”);
```

ตัวอย่างโปรแกรมคำนวณอย่างง่าย

```
01:  public class ComputeArea {
02:    /** Main method */
03:    public static void main(String[] args) {
04:       double radius;
05:       double area;
06:       // Assign a radius
07:       radius = 2.0;
08:       // Compute area
09:       area = radius * radius * 3.14159;
10:       // Display results
11:       System.out.println("The area for the circle of radius " +
12:                                  radius + " is " + area);
13:     }
14: }
```

บรรทัดที่

2, 6, 8, 10: เป็น comment ไม่ได้ถูกใช้ในการ compile และไม่เกี่ยวกับการทำงานของโปรแกรม ถูกเขียนเพื่ออธิบาย source code เท่านั้น มีความสำคัญเมื่อผู้เขียนโปรแกรมต้องการทำความเข้าใจโปรแกรมภายหลัง

1: เป็นการประกาศ class ชื่อ ComputeArea (เครื่องหมาย { แสดงว่า block ของ class ComputeArea เริ่มที่นี่)

3: เป็นการประกาศ method ชื่อ main ซึ่งมี argument เป็น array ของ String ซึ่งรายละเอียดของ argument นี้จะกล่าวภายหลัง(เครื่องหมาย { แสดงว่า block ของ method main เริ่มที่นี่)

4, 5: เป็นการประการศตัวแปรชนิด double เพื่อเก็บข้อมูลเลขทศนิยม 2 ตัวชื่อ radius และ area

7: เป็น statement การตั้งค่า (assignment statement) CPU จะกระทำการตั้งค่าให้ ตัวแปร radius กล่าวโดยละเอียดคือ compiler จะเปลี่ยน 20 ซึ่งเป็นเลขจำนวนเต็มให้เป็น 20.0 (ชนิด double ซึ่งเป็นชนิดเดียวกับตัว operand ด้านซ้ายมือของ operator =)

9: เป็น assignment statement ที่มี expression อยู่ด้านขวามือของ operator = ซึ่งตามลำดับ precedence ลำดับการทำการ statement นี้มีดังนี้

9.1 compiler จะเลือกทำ * ทั้งสองก่อน = โดยจะทำจากซ้ายไปขวา ได้ 4.0 * 3.14159

9.2 compiler ทำ * ที่เหลือ ได้ 12.56636

9.3 compiler ทำ = นั่นคือ ทำการตั้งค่า 12.56636 ให้ตัวแปรชื่อ area (พูดอีกอย่างว่า นำค่า 12.56636 ไปใส่ในหน่วยความจำที่ถูกตั้งชื่อว่า area)

11: พิมพ์ค่าที่อยู่ในวงเล็บออกหน้าจอ

13: เครื่องหมาย } เป็นการบอก compiler ว่า block ของ method main ได้สิ้นสุดที่ตำแหน่งนี้

14: เครื่องหมาย } เป็นการบอก compiler ว่า block ของ class ComputeArea ได้สิ้นสุดที่ตำแหน่งนี้

การรับ input จาก keyboard โดยใช้ Scanner

ใน Java เราสามารถรับข้อมูลจากแป้นพิมพ์เพื่อนำมาประมวลผลต่อไปโดยมีลำดับขั้นตอนต่อไปนี้:-

1. สร้าง object Scanner ดังตัวอย่าง

```
Scanner scanner = new Scanner(System.in);
```

2. เรียกใช้ methods next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() เพื่อรับค่า string, byte, short, int, long, float, double, หรือ boolean ตามลำดับ เช่น

```
System.out.print("Enter a double value: ");
double d = scanner.nextDouble(); // scanner คือ object ที่เราสร้างขึ้นในขั้นตอนที่ 1
```