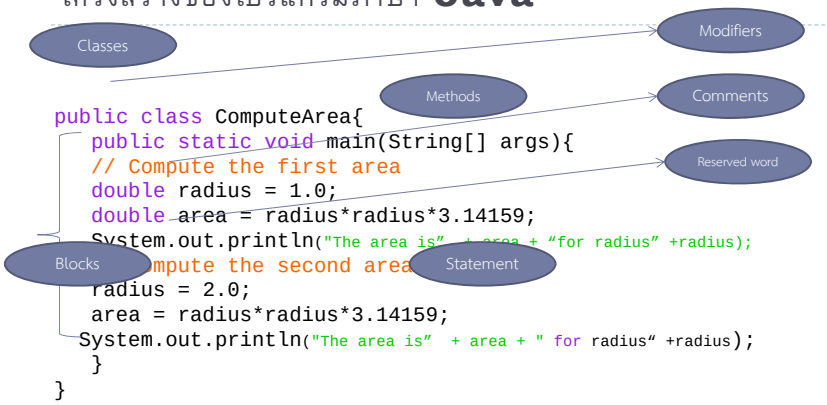
Object Oriented Programming

Lecture#1 Basic of OOP

โครงสร้างของโปรแกรมภาษา **Java**




```
public class ComputeArea{
  public static void main(String[] args){
  // Compute the first area
  double radius = 1.0;
  double area = radius*radius*3.14159;
  System.out.println("The area is"  + area + "for radius" +radius);
  // Compute the second area
  radius = 2.0;
  area = radius*radius*3.14159;
System.out.println("The area is"  + area + " for radius" +radius);
  }
}
```

ชื่อที่ตั้งขึ้นในภาษาจาวา - Identifiers

- identifier คือชื่อที่ตั้งขึ้นในภาษาจาวา ซึ่งอาจเป็นชื่อของคลาส ชื่อของตัวแปร ชื่อของเมธอด หรือชื่อของค่าคงที่ ซึ่งจะต้องเป็นไปตามกฎการตั้งชื่อ ดังนี้
  - ต้องขึ้นต้นด้วยอักขระ A-Z, a-z, _ หรือ $ เท่านั้น
  - ตัวอักขระที่มากกว่าหนึ่งตัว ตัวอักขระหลังจากตัวแรกจะต้องเป็นตัวอักขระหรือเป็นตัวเลข 0 ถึง 9 เท่านั้น
  - ต้องไม่ตรงกับคีย์เวิร์ด reserved word
  - ต้องไม่ใช้ true, false, หรือ null
  - ตัวอักษรพิมพ์ใหญ่และตัวอักษรพิมพ์เล็กต่างกัน(case sensitive) เช่น myVariable จะแตกต่างจาก MyVariable

ตัวแปร Variables

Variable – ใช้สำหรับเก็บข้อมูลในโปรแกรม สามารถเปลี่ยนค่าได้

การประกาศตัวแปร Declaring Variables

```
int x;         // Declare x to be an integer variable;
double radius; // Declare radius to be a double variable;
char a;        // Declare a to be a character variable;
```

การกำหนดค่าให้กับตัวแปร Assignment Statements

```
x = 1;          // Assign 1 to x;
radius = 1.0;   // Assign 1.0 to radius;
a = 'A';        // Assign 'A' to a;
```

การประกาศตัวแปรและกำหนดค่าให้กับตัวแปรใน 1 บรรทัด

```
int x = 1;
double d = 1.4;
float f = 1.4;
```

ข้อมูลค่าคงที่ Constants

- ข้อมูลค่าคงที่คือคำที่ใช้แสดงข้อมูลที่เป็นตัวเลข ตัวอักขระ ข้อความ หรือค่าทางตรรกะ ประกอบด้วย
  - ตรรกะ(boolean)
  - ตัวอักขระ(character)
  - ตัวเลขจำนวนเต็ม(integral)
  - ตัวเลขทศนิยม(floating point)
  - ข้อความ(string)

```
final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int SIZE = 3;
```

ชนิดข้อมูลแบบพื้นฐาน

- ภาษาจาวาเป็นภาษาที่ต้องระบุชนิดข้อมูลอย่างชัดเจน (strongly typed language)
- ชนิดข้อมูลแบบพื้นฐาน (primitive data type)
  - ชนิดข้อมูลตรรกะ
  - ชนิดข้อมูลอักขระ
  - ชนิดข้อมูลตัวเลขจำนวนเต็ม
  - ชนิดข้อมูลตัวเลขทศนิยม
- ชนิดข้อมูลแบบอ้างอิง (reference data type)
  - ชนิดข้อมูลที่เป็นคลาส
  - ชนิดข้อมูลที่เป็นอะเรย์

Numerical Data Types

ชนิดข้อมูล	จำนวนบิต
byte	8 bits
short	16 bits
int	32 bits
long	64 bits
float	32 bits
double	64 bits

Operators

- ในการคำนวณกับชนิดข้อมูลแบบตัวเลขจะมีการใช้เครื่องหมาย (operator) ดังนี้ คือ

  + - * / และ %

- เมื่อนำชนิดข้อมูลตัวเลขจำนวนเต็ม integer กระทำต่อกันจะได้ผลลัพธ์เป็นชนิดข้อมูลแบบ integer เช่น
  - 5/2 จะได้ 2
  - ส่วนกรณีที่ตัวใดตัวหนึ่งเป็น float จะทำให้ผลลัพธ์ที่ได้เป็นเลขทศนิยม เช่น 5.0/2 จะได้ 2.5
  - ส่วนการหารเอาเศษ(Modulation) จะกระทำกับชนิดข้อมูลจำนวนเต็มเท่านั้น เช่น 5%2 จะได้ผลลัพธ์เป็น 1

ข้อมูลค่าคงที่ - Number Literals

- ค่าคงที่คือตัวเลขที่ปรากฏโดยตรงในโปรแกรม เช่น 34, 1,000,000, และ 5.0 เป็นค่าคงที่ที่ปรากฏในคำสั่งต่อไปนี้ `int i = 34;` `long l = 1000000;` `double d = 5.0;`
- ค่าคงที่มี 5 ชนิดหลัก ๆ คือ Integer, floating point, string, logical, character.
  - สามารถกำหนดให้กับตัวแปรตัวเลขได้ ถ้าค่าคงที่ตัวเลขดังกล่าวมีขนาดหรือจำนวนไบต์ที่สามารถเก็บไว้ในตัวแปรได้
  - ข้อมูลแบบ byte จะสามารถเก็บข้อมูลในช่วง -2^7 -1 ถึง 2^7 -1 ค่าคงที่ตัวเลขจะกำหนดให้เป็น int เมื่อค่าของตัวเลขดังกล่าวอยู่ระหว่าง -2^{31} (-2147483648) ถึง 2^{31}-1 (2147483647)
  - ค่าคงที่ตัวเลขจะมีชนิดข้อมูลเป็น long เมื่อเติม L หรือ l ต่อท้าย
  - แปลงตัวเลขที่เป็น double ให้เป็น float ได้โดยการเติมตัวอักษร f หรือ F
  - แปลงตัวเลขให้อยู่ในรูปของ double ทำได้โดยการเติม d หรือ D
  - จะสามารถนำมาเขียนในรูปของสัญลักษณ์ทางวิทยาศาสตร์ได้ เช่น 1.23456e+2, หรือ 1.23456e2, จะมีค่าเท่ากับ 123.456

ตัวดำเนินการทางคณิตศาสตร์แบบย่อ

Shortcut Assignment Operators

Operator	Example	Equivalent
+=	i+=8	i = i+8
-=	f-=8.0	f = f-8.0
*=	i*=8	i = i*8
/=	i/=8	i = i/8
%=	i%=8	i = i%8

```
int i=10;
int newNum = 10*(++i);
```
Equivalent to
```
i = i + 1;
int newNum = 10*i;
```

```
int i=10;
int newNum = 10*i++;
```
Equivalent to
```
int newNum = 10*i;
i = i + 1;
```

ตัวดำเนินการเพิ่มค่าและลดค่า

Increment and Decrement Operators

Type Casting

- double
- float
- long
- int
- short
- byte

Implicit casting
```
double d = 3;
(type widening)
```

Explicit casting
```
int i = (int)3.0;
(type narrowing)
```

What is wrong?
```
int x = 5/2.0;
```

การแปลงข้อมูลที่กว้างขึ้น(widening conversion) คือการแปลงจากชนิดข้อมูลที่มีขนาดเล็กกว่าไปเป็นชนิดข้อมูลที่มีขนาดใหญ่กว่า

การแปลงข้อมูลที่แคบลง (narrowing conversion) คือการแปลงจากชนิดข้อมูลที่มีขนาดใหญ่กว่าไปเป็นชนิดข้อมูลที่มีขนาดเล็กลง ซึ่งอาจมีผลให้เสียความละเอียดของข้อมูลบางส่วนไป

การแปลงชนิดข้อมูล Numeric Type Conversion

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

int x = k; //(Wrong)
long k = x; //(fine, implicit casting)
```

- การแปลงข้อมูลระหว่างชนิดข้อมูลตัวอักษรและชนิดข้อมูลตัวเลข

  การแปลงชนิดข้อมูลตัวอักษรและชนิดข้อมูลตัวเลขสามารถทำได้โดยการระบุชนิดข้อมูลที่ต้องการหน้าตัวแปรที่ต้องการแปลง เช่น

  int i = 'a'; มีความหมายเดียวกับ int i = (int)'a';

  char c = 97; มีความหมายเดียวกับ char c = (char)97;

Character Data Type

- การเก็บชนิดข้อมูลแบบตัวอักษรอาจจะเก็บในรูปของรหัสแอสกี้หรือรหัส Unicode ก็ได้ หรืออาจจะเก็บตัวอักษรโดยตรง

```
char letter = 'A'; (ASCII)
char numChar = '4'; (ASCII)
char letter = '\u0041'; (Unicode)
char numChar = '\u0034'; (Unicode)
```

- กรณีที่ต้องการประกาศตัวอักขระพิเศษสามารถประกาศได้คล้ายกัน เช่น

```
char tab = '\t';
```

- การแปลงจาก String ให้เป็น ตัวเลข integer ใช้เมธอด static parseInt ที่อยู่ในคลาส Integer เช่น:
- int intValue = Integer.parseInt(intString);
- การแปลงจาก String ให้เป็น ตัวเลข double ใช้เมธอด static parseDouble ที่อยู่ในคลาส Double เช่น:
- double doubleValue =Double.parseDouble(doubleString);

Unicode Format

Description	Escape Sequence	Unicode
Backspace	\b	\u0008
Tab	\t	\u0009
Linefeed	\n	\u000a
Carriage return	\r	\u000d

ASCII Character Set

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

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		

The boolean Type and Operators

- ในภาษาจาวาจะมีชนิดข้อมูลแบบตรรกะโดยชนิดข้อมูลแบบนี้เป็นชนิดข้อมูลที่มีค่าความเป็นจริงอยู่ 2 ค่าคือ จริง (True) และเท็จ (False) เช่น

```
boolean lightsOn = true;
boolean lightsOn = false;
boolean b = (1 > 2);
```

- ในกรณีที่มีการเช็คเงื่อนไขมากกว่า 1 เงื่อนไขจะใช้เครื่องหมาย && (and) || (or) ! (not) เช่น

```
(1 < x) && (x < 100)
 (lightsOn) || (isDayTime)
!(isStopped)
```

Comparison Operators

Operator	Name
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Operand1	Operand2	!operand1	Operand1 && Operand2	Operand1 || Operand2	Operand1 ^ Operand2
false	false	true	false	false	false
false	true	true	false	true	true
True	false	fase	false	true	true
true	true	false	true	true	false

ลำดับความสำคัญของโอเปอร์เรเตอร์

Operator Precedence

- var++, var--
- +, - (Unary plus and minus), ++var,--var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and modulus)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Comparison)
- ==, !=; (Equality)
- & (Unconditional AND)
- ^ (Exclusive OR)
- | (Unconditional OR)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR
- =, +=, -=, *=, /=, %= (Assignment operator)

Operator Associativity

- เมื่อโอเปอเรเตอร์ที่มีลำดับความสำคัญเท่ากันเขียนต่อกันส่วนมากจะมีการเรียงลำดับการทำงาน จากซ้ายไปขวาที่เรียกว่า Left-associative ยกเว้นการใช้เครื่องหมายเท่ากับเมื่อกำหนดค่าจะทำจากขวาไปซ้ายที่เรียกว่า right-associative เช่น

  a - b + c - d จะมีค่าเท่ากับ ((a -b) + c)-d

  a = b += c = 5 จะมีค่าเท่ากับ a = (b += (c = 5))

- กรณีที่โอเปอแรนด์มี side effect คือมีการเปลี่ยนค่าของตัวแปรก่อนค่อยนำไปคำนวณในคำสั่งเดียวกัน หากมีการเรียงลำดับเครื่องหมายในการคำนวณไม่ถูกต้องจะทำให้ค่าของตัวแปรไม่ถูกต้องตามความต้องการจริง

ลำดับการคำนวณของโอเปอแรนด์

Operand Evaluation Order

- **ตัวอย่าง**

```
int a = 0;
int x = a + (++a);
```

- X จะกลายเป็น 1 เมื่อทำตามคำสั่งข้างต้น เนื่องจากเริ่มต้น a จะมีค่าเป็น 0 หลังจากนั้นจะนำค่าที่อยู่ใน a คือ 0 ไปบวกกับค่าภายหลังการเพิ่มค่า a จากคำสั่ง ++a ซึ่งจะได้ 1 ทำให้ผลที่ได้คือ 0+1 ซึ่งมีค่าเป็น 1 หลังจากนั้นจะนำเลข 1 ดังกล่าวไปเก็บในตัวแปร x
- **ตัวอย่าง**

```
int a = 0;
int x = ++a + a;
```

- X จะกลายเป็น 2 เมื่อทำตามคำสั่งข้างต้น เนื่องจากเริ่มต้น a มีค่าเป็น 0 หลังจากนั้นจะทำคำสั่ง ++a คือเพิ่มค่า a ก่อนทำให้ a มีค่าเป็น 1 แล้วนำค่า a มาบวกต่อซึ่งกลายเป็น 1+1 จะได้ค่าเป็น 2 หลังจากนั้นจะนำ 2 ดังกล่าวไปเก็บไว้ในตัวแปร x

Programming Errors

- Syntax Error รูปแบบไวยากรณ์ของโปรแกรมผิดพลาดเช่น พิมพ์ผิด เป็นข้อผิดพลาดที่ตรวจสอบได้จากขั้นตอนการคอมไพล์ โดยคอมไพเลอร์จะแสดงข้อความชี้แจงความผิดพลาดและระบุตำแหน่งที่ผิด
  - รูปแบบข้อความแสดง Syntax Error ของจาวา

ชื่อไฟล์ซอร์สโค้ด : เลขที่บรรทัด : ข้อความแสดงข้อผิดพลาด
โค้ดบรรทัดที่ผิดพลาด
^ ตัวชี้ตำแหน่งในบรรทัดที่ผิดพลาด
จำนวนข้อผิดพลาดที่พบ

- Runtime Errors ข้อผิดพลาดในขณะที่โปรแกรมทำงาน เป็นข้อผิดพลาดของการใช้คำสั่งที่ไม่ถูกต้อง เช่นการอ้างถึงหน่วยความจำที่ใช้ไม่ได้หรือไม่มี การเปิดไฟล์ที่ไม่มีอยู่
- Logic Errors ข้อผิดพลาดของวิธีการแก้ปัญหาที่หากนำมาเขียนโปรแกรมแล้วทำให้ได้ผลลัพธ์ไม่ตรงตามที่ต้องการ

Compilation Errors

```
public class ShowSyntaxErrors {
  public static void main(String[] args) {
    i = 30;
    System.out.println(i+4);
  }
}
```

Runtime Errors

```
public class ShowRuntimeErrors {
  public static void main(String[] args) {
    int i = 1 / 0;
  }
}
```

Logic Errors

```
public class ShowLogicErrors {
  // Determine if a number is between 1 and 100 inclusively
  public static void main(String[] args) {
    // Prompt the user to enter a number
    String input = JOptionPane.showInputDialog(null,
      "Please enter an integer:",
      "ShowLogicErrors", JOptionPane.QUESTION_MESSAGE);
    int number = Integer.parseInt(input);
    // Display the result
    System.out.println("The number is between 1 and 100, " +
      "inclusively? " + ((1 < number) && (number < 100)));
     System.exit(0);
  }
}
```

Object Oriented programming

Control statements

คำสั่งทดสอบเงื่อนไข **if Statements**

- กรณีที่ต้องการให้โปรแกรมมีการทำหรือไม่ทำคำสั่งที่ผู้ใช้กำหนดจะทำได้โดยการใช้คำสั่ง if โดยมีรูปแบบดังนี้

```
if (booleanExpression) {
  statement(s);
}
```

- เช่น ต้องการทดสอบว่าค่า i มีค่ามากกว่า 0 และน้อยกว่า 10 ให้พิมพ์คำสั่ง i เป็นตัวเลขที่อยู่ระหว่าง 0 ถึง 10

```
public class ShowIf {
    public static void main(String[] args) {
        if ((i > 0) && (i < 10)) {
           System.out.println("i is an " +  "integer
  between 0 and 10");
        }
    }
  }
```

คำสั่งทดสอบเงื่อนไข **if...else Statement**

- มีรูปแบบดังนี้

```
if (booleanExpression) {
  คำสั่งกรณีที่เงื่อนไขถูกต้อง;
}
else {
  คำสั่งกรณีที่เงื่อนไขไม่ถูกต้อง;
}
```

```
public class ShowIf {
  public static void main(String[] args) {
      if (radius >= 0) {
         area = radius*radius*PI;
         System.out.println("The area for
the "
         + "circle of radius " + radius +
         " is " + area);
      }else {
         System.out.println("Negative
input");
      }
  }
}
```

คำสั่งทดสอบเงื่อนไข **if...else Statement**

```
public class ShowIf {
    public static void main(String[] args) {
        if (score >= 90)
           grade = 'A';
        else
           if (score >= 80)
              grade = 'B';
        else
           if (score >= 70)
              grade = 'C';
        else
           if (score >= 60)
              grade = 'D';
        else
         grade = 'F';
    }
  }
```

Multiple Alternative if Statements

```
public class ShowIf {
  public static void main(String[] args) {
      if (score >= 90)
          grade = 'A';
      else if (score >= 80)
         grade = 'B';
      else if (score >= 70)
         grade = 'C';
      else if (score >= 60)
         grade = 'D';
      else
         grade = 'F';
  }
}
```

Multiple Alternative if Statements

คำสั่ง else จะจับคู่กับ if ที่ใกล้ตัวมันมากที่สุดก่อน เช่น

```
int i = 1; int j = 2; int k = 3;
if (i > j)
  if (i > k)
    System.out.println("A");
else
  System.out.println("B");
```

จะมีผลการทำงาน

```
int i = 1; int j = 2; int k = 3;
if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

เมื่อต้องการให้โปรแกรมทำคำสั่ง if-else ตามที่เราต้องการให้ใส่ block ของปีกกาเปิดปิดเข้าไปเพื่อจัดลำดับการทำงาน

การทดสอบเงื่อนไข

- สามารถใช้เครื่องหมายทดสอบเงื่อนไขแบบย่อสำหรับทดสอบเงื่อนไขได้ เช่น

```
(booleanExp) ? exp1 : exp2
```

- เช่น

```
if (num % 2 == 0)
    System.out.println(num + “is even”);
  else
    System.out.println(num + “is odd”);
```

สามารถเปลี่ยนเป็น

```
System.out.println(
  (num % 2 == 0)? num + “is even” :
  num + “is odd”);
```

คำสั่งทดสอบเงื่อนไข switch Statements

- เมื่อมีตัวเลือกให้เลือกมากกว่า 1 ตัวเลือกอาจใช้คำสั่ง switch-case สำหรับทดสอบเงื่อนไข
- รูปแบบ

```
switch (switch-expression) {
case value1:
. . . // x is valueOne
break;
case value2:
. . . // x is valueTwo
break;
default:
. . . // other value of x
}
```

- เมื่อ switch-expression คือ เงื่อนไขที่จะทดสอบค่า มีชนิดข้อมูลเป็นแบบ byte, short, int, long และ char เท่านั้น
- ค่า value1 ถึง valueN จะต้องเป็นชนิดข้อมูลเดียวกันกับตัวแปรที่อยู่ใน switch-expression
- คำสั่งที่อยู่ใน Block ของค่าที่ตรวจสอบในแต่ละกรณี หากตรงจะทำงานในบล็อกของคำสั่งแต่ละ case เมื่อสิ้นสุดการทำงานจะใช้คำสั่ง break เพื่อบอกจุดสิ้นสุดการทำงานในแต่ละ case โดยไม่ต้องตรวจสอบเงื่อนไขอื่น ๆ ที่เหลือ กรณีที่ไม่ใส่ break จะทำให้ case ต่อไปใน switch case ถูกประมวลผลโดยอัตโนมัติ

switch Statement Rules

- **ข้อควรระวัง**
- ควรจะใช้ break เมื่อสิ้นสุดการทำงานในแต่ละ case เพื่อให้ผลการทำงานถูกต้อง เช่นตัวอย่างโปรแกรมต่อไปนี้ หากไม่มี switch case หาก numberofyear เป็น 15 ผลลัพธ์ที่ได้จะกำหนดเป็น 8.50 และ 9.0 และพิมพ์ wrong ในลำดับต่อไป เช่น

```
public class ShowSwitchCase {
    public static void main(String[] args) {
        switch (year) {
           case 7:  annualInterestRate = 7.25;
             break;
           case 15: annualInterestRate = 8.50;
             break;
           case 30: annualInterestRate = 9.0;
             break;
           default: System.out.println(
                 "Wrong number of years, enter 7, 15, or 30");
        }
    }
}
```

คำสั่งการวนซ้ำ **(Repetitions)**

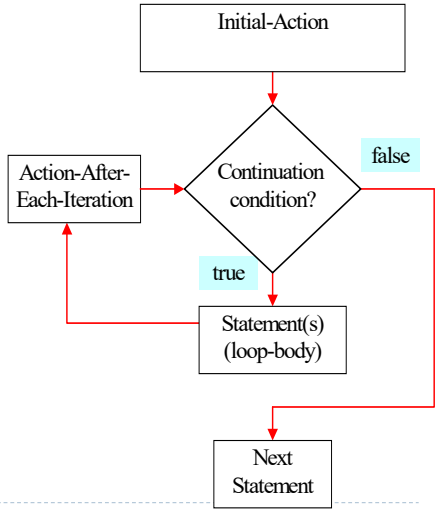
- การวนซ้ำ หมายถึง การควบคุมให้การกระทำการบางคำสั่งซ้ำหลายรอบ ซึ่งจะช่วยให้การเขียนโปรแกรมทำได้ง่ายสะดวก ไม่ต้องเขียนคำสั่งเดิมหลายครั้ง ทำให้โปรแกรมมีความกระชับ สามารถตรวจสอบความผิดพลาดได้ง่าย คำสั่งการวนรอบมี 3 ชนิดหลัก ๆ คือ คำสั่ง for, คำสั่ง do-while และคำสั่ง while
- โดยที่แต่ละคำสั่งมีรูปแบบและวิธีการใช้งานที่แตกต่างกัน สามารถเลือกใช้ตามความเหมาะสมของการใช้งานในโปรแกรม
- **ส่วนประกอบของคำสั่งแบบวนซ้ำ** ประกอบด้วย
  - ส่วนของการตรวจสอบ (loop test) เป็นเงื่อนไขเพื่อทดสอบว่าจะทำวนซ้ำอีกหรือไม่
  - ส่วนของการทำวนซ้ำ (loop body) เป็นชุดคำสั่งที่จะถูกดำเนินการ
- **รายละเอียดของการสร้าง loop มีขั้นตอนการทำงานดังนี้**
  - ระบุส่วนของการทำงานที่ต้องทำซ้ำ (loop body) ในส่วนนี้จะเป็นการระบุเงื่อนไข (loop test) ที่จะทำซ้ำหรือเลิกทำซ้ำ
  - ระบุชนิดของ loop ที่จะใช้ ซึ่งชนิดของการทำซ้ำจะมีประเภทที่แตกต่างกันหลัก ๆ 2 ประเภทคือPre-test loop และ Post-test loop

คำสั่ง **for**

- คำสั่ง for เป็นคำสั่งที่สั่งให้ทำคำสั่ง หรือกลุ่มของคำสั่งวนซ้ำหลายรอบ โดยมีจำนวนรอบในการวนซ้ำที่แน่นอน
- รูปแบบ

for (ค่าเริ่มต้น; เงื่อนไข; ค่าเพิ่มหรือค่าลด)

- โดยที่ ค่าเริ่มต้น, เงื่อนไข, ค่าเพิ่มหรือค่าลด เป็นนิพจน์
- คำสั่ง หมายถึง คำสั่งที่จะถูกกระทำซ้ำ ซึ่งอาจจะมีเพียงคำสั่งเดียว หรือหลายคำสั่งก็ได้



คำสั่ง **for**

- คำสั่ง for มีขั้นตอนการทำงานดังนี้
- 1. คำนวนหาค่าเริ่มต้นของตัวแปรที่ใช้ควบคุมการวนซ้ำ ซึ่งอยู่ในรูปของคำสั่งกำหนดค่า
- 2. คำนวนหาผลลัพธ์จากเงื่อนไข ที่อยู่ในรูปของนิพจน์ความสัมพันธ์ ซึ่งจะให้ผลเป็นเท็จ(0) หรือจริง (ค่าที่ไม่เป็น 0)
- 3. ถ้าผลลัพธ์จากข้อ 2 มีค่าเป็นเท็จหรือ 0 ไปที่ 7
- 4. ถ้าผลลัพธ์จากข้อ 2 มีค่าเป็นจริง หรือค่าที่ไม่ใช่ 0 ข้อความสั่งที่อยู่ภายในคำสั่ง for จะถูกกระทำ
- 5. คำนวณหาค่าใหม่ของตัวแปรที่ใช้ควบคุมการวนซ้ำ
- 6. กลับไปที่ข้อ 2
- 7. จบการกระทำการคำสั่ง for และข้อความแรกที่อยู่ถัดจากคำสั่ง for จะถูกทำในลำดับต่อไป

```
public class ShowForLoop {
  public static void
main(String[] args) {
   int i;
   for (i = 0; i < 100; i++) {

System.out.println("Welcome to
Java! " + i);
   }
  }
}
```

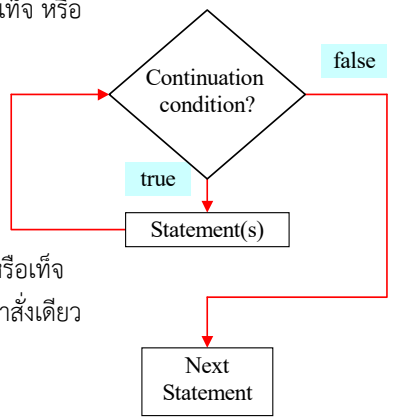
while Loop Flow Chart

คำสั่ง while เป็นคำสั่งวนซ้ำ ที่สั่งให้ทำคำสั่งที่อยู่ภายในคำสั่ง while หลายรอบจนกระทั่งเงื่อนไขเป็นเท็จ หรือ 0 จึงจะจบการวนซ้ำ

รูปแบบ

```
while (เงื่อนไข){
        คำสั่ง
}
```

โดยที่ เงื่อนไข เป็นนิพจน์ที่ให้ผลลัพธ์เป็นจริงหรือเท็จ และคำสั่งอยู่ภายในคำสั่ง while อาจมีเพียงคำสั่งเดียว หรือหลายคำสั่ง



while Loop Flow Chart

คำสั่ง while มีขั้นตอนการทำงานดังนี้

1 คำนวณหาค่าของเงื่อนไข

2. ถ้าค่าของเงื่อนไข มีค่าเป็นเท็จหรือศูนย์ ไปที่ข้อ 5

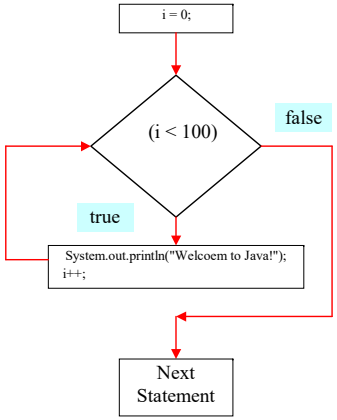
3. ถ้าค่าของเงื่อนไข มีค่าเป็นจริงหรือค่าที่ไม่ใช่ศูนย์ คำสั่งที่อยู่ภายในคำสั่ง while จะถูกกระทำ

4. กลับไปที่ข้อ 1

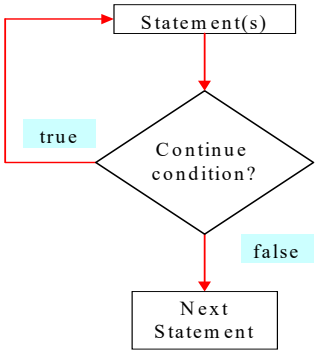
5. จบการกระทำการคำสั่ง while และข้อความแรกที่อยู่ถัดจากคำสั่ง while จะถูกทำในลำดับต่อไป

ตัวอย่าง

```
public class ShowWhileLoop {
  public static void main(String[] args) {
   int i = 0;
   while (i < 100) {
       System.out.println(
       "Welcome to Java!");
       i++;
   }
  }
}
```



do-while Loop



- คำสั่ง do-while เป็นคำสั่งวนซ้ำ ที่สั่งให้ทำคำสั่งที่อยู่ภายในคำสั่ง do-while หนึ่งรอบ แล้วจึงจะตรวจสอบเงื่อนไข ถ้าเงื่อนไขเป็นเท็จจะจบการทำงานทันที
- รูปแบบ

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

- โดยที่ เงื่อนไข เป็นนิพจน์ที่ให้ผลลัพธ์เป็นจริงหรือเท็จ และคำสั่งอยู่ภายในคำสั่ง do-while อาจมีเพียงคำสั่งเดียว หรือหลายคำสั่ง

Which Loop to Use?

- เราสามารถใช้รูปแบบของการวนซ้ำในรูปแบบใดก็ได้ คือ while, do, and for, โดยผลจากการทำงานจะมีลักษณะคล้ายกัน ให้เลือกใช้ตามที่ผู้ใช้ถนัดมากที่สุด โดยทั่วไป
  - for จะใช้ในกรณีที่ทราบจำนวนของรอบที่ต้องการประมวลผลอย่างชัดเจน
  - while จะใช้ในกรณีที่ไม่ทราบจำนวนของรอบที่ต้องการประมวลผลอย่างชัดเจน
  - do-while จะใช้ในกรณีที่ไม่ทราบจำนวนของรอบที่ต้องการประมวลผลอย่างชัดเจน และจำเป็นต้องมีการทำงานก่อนการเช็คเงื่อนไข

ข้อควรระวัง

การเติม Semicolon ที่ส่วนท้ายของ for จะทำให้เกิดความผิดพลาดเกิดขึ้น ดังแสดงด้านล่าง

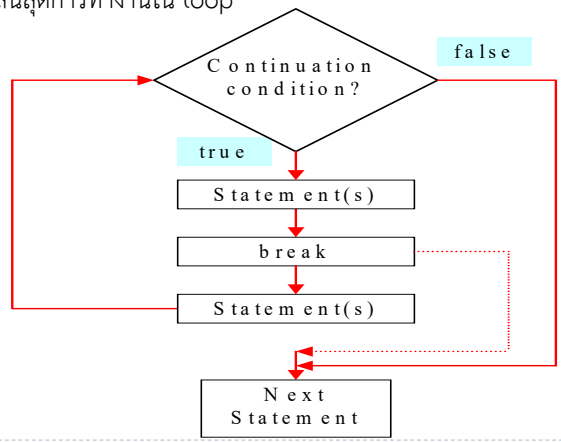
```
for (int i=0; i<10; i++);
{
  System.out.println("i is " + i);
}
```

และในทำนองเดียวกันการเติม Semicolon ที่ส่วนท้ายของ while จะทำให้เกิดความผิดพลาดเกิดขึ้น

```
int i=0;
while (i<10);
{
  System.out.println("i is " + i);
  i++;
}
```

คำสั่ง break

เป็นคำสั่งที่ทำให้หยุดสิ้นสุดการทำงานของโครงสร้างแบบทำซ้ำหรือเป็นคำสั่งให้สิ้นสุดการทำงานใน loop



The continue Keyword

คำสั่งที่จะข้ามการทำงานคำสั่งที่เหลือภายในบล็อก { } โดยไปเริ่มการทำซ้ำในรอบต่อไป