# Program Flow Control

Vittayasak Rujivorakul

Lecture 3

# Program Flow Control

Vittayasak Rujivorakul

Lecture 3



# กฎการตั้งชื่อ Identifiers

- ชื่อของ Identifier ประกอบด้วยลำดับของ**ตัวอักษร, ตัวเลข, $** และ _ โดยที่**ตัวแรกของชื่อจะต้องเป็นตัวอักษร**
- ต้องไม่เป็น reserved word
- Java เป็น ภาษาที่ Case Sensitive (Identifier **J**ava ไม่เหมือนกับ **j**ava)
- สามารถใช้ identifier ในการตั้งชื่อ class, object, method และอื่น ๆ
  - ตัวอย่าง :

    FunTime

    area

    getName



# ตัวแปร (Variable)

- ตัวแปร = ชื่อที่ใช้ในการเก็บข้อมูล
- การตั้งชื่อตัวแปรเป็นไปตามกฎการตั้งชื่อของ Identifier
- Syntax: รูปแบบการประกาศตัวแปร

  **<data type> <variable>;**

  **<data type>** = ชนิดของตัวแปร

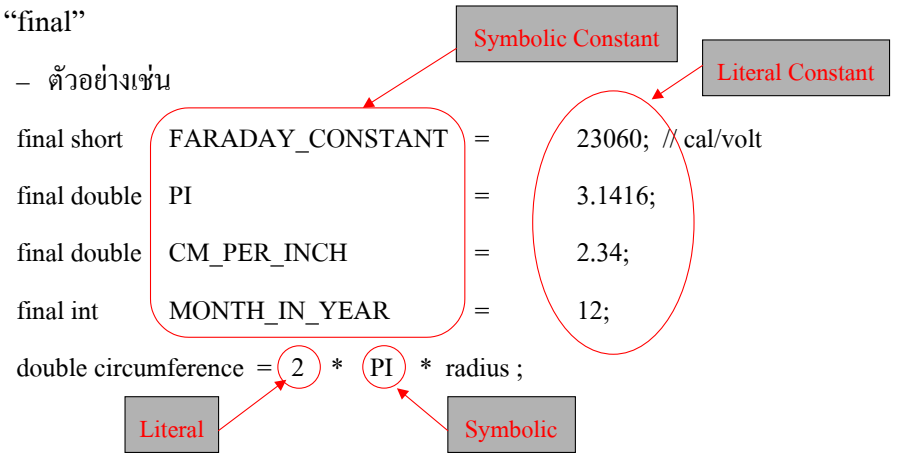
  **<variables>** = กลุ่มลำดับของตัวแปรแยกจากกันโดย “,”
  - ตัวอย่าง:

    int x , y ;



# ค่าคงที่ (Contant)

- การประกาศค่าคงที่ทำเช่นเดียวกับการประกาศ variable แต่เพิ่ม modifier “final”
  - ตัวอย่างเช่น



# Primitive Data Types

- Primitive Data Type 8 ชนิด
  - boolean
  - byte
  - short
  - int
  - long
  - float
  - double
  - char

**boolean :** true, false

**int :** 8, -123, 45678

**long :** 123L, 9876543L

**float :** 99.9f, -3.5f,

**double :**

99.9, -3.5, 76.5E25

**char :** 'a', '9', 'B'

**String :**

**"**This is a string**"**

- กับ 1 Literal พิเศษ: String



# ค่าของ Primitive Data Types

| Data Type | Content | Default Value | Min. Value | Max. Value | Length |
|---|---|---|---|---|---|
| byte | integer | 0 | -2^7 | 2^7 - 1 | 8 bit |
| short | integer | 0 | -2^15 | 2^15 - 1 | 16 bit |
| int | integer | 0 | -2^31 | 2^31 - 1 | 32 bit |
| long | integer | 0 | -2^63 | 2^63 | 64 bit |
| float | Real | 0 | -3.40282347E+38 | 3.40282347E+38 | 32 bit |
| double | Real | 0 | -1.79769...E+308 | 1.79769...E+308 | 64 bit |
| char | Text | 16 bit Unicode character enclosed in single quote(' ') | | | |
| boolean | Logical | Possible value: true, false | | | |

*use \u escapes to define any Unicode character ie.. Char c = '\u0041';



# ข้อมูลชนิดตัวเลขจำนวนเต็ม Byte, short, int, และ long

- กำหนดข้อมูลชนิดตัวเลขได้ 3 รูปแบบ :
  - **เลขฐานสิบ (Decimal)**
    - 12 เลขฐานสิบที่มีค่า สิบสอง
  - **เลขฐานแปด (Octal)**
    - 062 เลขฐานแปดขึ้นต้นด้วยเลขศูนย์ แสดงค่า (62) ฐาน 8 = 50
  - **เลขฐานสิบหก (Hexadecimal)**
    - 0xA2 เลขฐานสิบหกขึ้นต้นด้วยเลขศูนย์ "X" แสดงค่า (A2) ฐาน 16 = 162
- มี default เป็น int
- กำหนด long โดยใช้ตัวอักษร "L" หรือ "I" (12L, 062L, 0xA21)



# ข้อมูลชนิดตัวเลขจำนวนจริง float และ double

- กำหนดข้อมูลชนิดตัวเลขได้ใส่ " . " หรือใส่ :
  - **E หรือ e เพื่อแสดง exponential**
    - 2.3**E**+2 ซึ่งมีค่าเท่ากับ 230
  - **F หรือ f เพื่อแสดง float**
    - 3.14**F** เป็น float ซึ่งมีค่าเท่ากับ 3.14
  - **D หรือ d เพื่อแสดง double**
    - 1.65E3**D** เป็น double ซึ่งมีค่าเท่ากับ 1650
- มี default เป็น double

# Character

- **Character** เป็นได้ทั้งตัวอักษร ตัวเลข และ special character
- **Unicode:** กำหนดโดย \u ตามด้วยเลขฐาน 16
  - \u00AE (C) The copyright symbol
  - \u0022 “ The double quote
  - \u00BD 1/2 The fraction 1/2
- **Escape characters:**
  - \b backspace - \t tab
  - \n linefeed - \f formfeed
  - \r carriage return - \” double quote, ”
  - \’ single quote, ‘ - \ \ backslash, \



# boolean

- **boolean** เป็น primitive data type ที่มีค่าได้เพียง 2 ค่า : **true** และ **false**
- ค่าของ boolean แตกต่าง 0 หรือ 1
- ไม่สามารถใช้ค่า (0, 1) ทดแทนเพื่อกำหนด boolean ได้
- boolean เป็น primitive data type ซึ่งแตกต่างจาก Boolean ซึ่งเป็น Object data type (case sensitive)



# คำสั่งประเภท Assignment

- เป็นการกำหนดหรือให้ค่ากับตัวแปร
- Syntax:

  **<variable> = <expression>;**

  **<variable>** = lvalue

  **<expression>** = rvalue

  - ตัวอย่าง:

    myIntNumber = 25 ;
- ในกรณี primitive คือการ copy content ของ rvalue ไปใส่ใน lvalue



# นิพจน์คณิตศาสตร์ (Arithmetic Expression)

- นิพจน์คณิตศาสตร์ ประกอบด้วย
  - ตัวอย่าง

    x + y / z
  - ตัวแปร หรือ ตัวคงที่ เรียกว่า **Operand** (เช่น **x, y, z**)
  - สัญลักษณ์ที่ใช้ในการคำนวณ เรียกว่า **Operator** (เช่น **+, /**)

# Java Arithmetic Expression

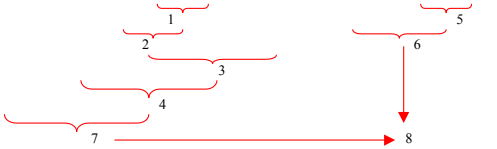
| Operation | Java Operator | Example | Value (x=10, y=7, z=2.5) |
|---|---|---|---|
| Addition | + | x + y | 17 |
| Subtraction | - | x - y | 3 |
| Multiplication | * | x * y | 70 |
| Division | / | x / y | 1 |
| | | x / z | 4 |
| Modulo division | % | x % y | 3 |



# ลำดับของ operator (Precedence Rules)

| Order | Group | Operator | กฎ |
|---|---|---|---|
| High | subexpression | ( ) | ทำเป็นอันดับแรก |
| ↓ | | | จากในสุดออกมาข้างนอก |
| ↓ | unary operator | - , + | ทำจากขวาไปซ้าย |
| ↓ | multiplicative op. | *, /, % | ทำจากซ้ายไปขวา |
| Low | / | +, - | ทำจากซ้ายไปขวา |

a * (b + -(c / d) / e) * (f - g % h)






# Casting Conversion

- **การเปลี่ยนชนิด**ของ data type เมื่อทำ arithmetic operation ของ data type ต่างชนิดกัน
- **Casting Conversion เป็นในลักษณะ**
  - implicit หรือ numeric promotion เป็นการแปลงโดยอัตโนมัติ
  - explicit เป็นการเปลี่ยนแปลงโดยการใช้ type cast operator
  - ตัวอย่าง : ถ้า x เป็น short

    x + 3 - - implicit casting to int

    (float) x / 3 - - explicit casting to float



# Implicit Arithmetic Promotion

- **Unary**
  - ถ้า operand เป็นชนิด short, ผลลัพธ์จะถูกเปลี่ยนเป็น **int**
  - ถ้าไม่เช่นนั้น เป็นค่าชนิดเดิม
- **Binary**
  - ถ้า operand **ตัวใดตัวหนึ่ง**เป็นชนิด **double**, ผลจะถูกเปลี่ยนเป็น **double**
  - ถ้า operand **ตัวใดตัวหนึ่ง**เป็นชนิด **float**, ผลจะถูกเปลี่ยนเป็น **float**
  - ถ้า operand **ตัวใดตัวหนึ่ง**เป็นชนิด **long**, ผลจะถูกเปลี่ยนเป็น **long**
  - ถ้าไม่เช่นนั้น operand ทั้ง 2 ตัวถูกเปลี่ยนเป็น **int**

# Assignment Conversion

- การ assign arithmetic expression ให้กับตัวแปรที่มีชนิดต่างกัน จะทำให้เกิด implicit conversion เช่นกัน
  - การปรับทำโดยอัตโนมัติ ถ้าหากปรับแล้วชนิดข้อมูลเป็นชนิดที่มีค่าใหญ่ขึ้น

**byte**
**short**
**int**
**long**
**float**
**double**

เล็กสุด → ใหญ่สุด



# ตัวอย่าง implicit conversion

```
int  j;
long  k;
j = 88;
k = j; // valid upward cast

k = 88L;
j = k; // invalid cast

short a, b, c;
a = 1;
b = 2;
c = a + b; // invalid cast
```

a+b ผลลัพธ์เป็น int

Declaration & Assignment พร้อมกัน

```
double  y = 1.24F; //valid
float  x = 1.25; // invalid
int smallVal = 99L; //invalid
long  bigVal = 6; // legal
```

default type เป็น double



# Type Cast Operator

- ทำได้ทั้ง การแปลงเป็นชนิดที่มีค่าขนาดใหญ่ขึ้นหรือเล็กลง
- Syntax: **(<data type>) <expression>**
- type cast เป็น unary operator ที่มี precedence เหนือกว่า binary operator อื่นๆ

```
short  a = 1,  b = 2;
short  c;
c = a + b;     // invalid cast
c = (short) (a+b);  // ok
```

```
int c;
long  bigVal = 6;  // legal
c = bigVal;   // invalid
c  = (int) bigVal; // valid
```



# แบบฝึกหัด

พิจารณาว่า expression ต่อไปนี้ อันใดที่ valid

```
int   i = 1,  j = 2;
float  x = 4.0f,  y = 2.0f;
double  u = 5.0,  v = 3.0;
```

```
i = x;        → Illegal implicit conversion of float to int
x = u + y;    → Illegal implicit conversion of float to int
x = 23.4 + j * y;  → Illegal implicit conversion of float to int
v = (int) x;
y = j / i * x;
```

# แบบฝึกหัด (ต่อ)

```
int   i = 1,   j = 2;
float  x = 4.0f,  y = 2.0f;
double  u = 5.0,  v = 3.0;
```

i = (int) x; → 4
x = (float) u + y; → 7.0
x = 23.4F + j * y; → 27.4
v = (int) x; → 27.0
y = j / i * x; → 0.0



# แบบฝึกหัด

- **กำหนดให้**

```
final  int  x = 9;
double y;
```

- **พิจารณาว่า statement ต่อไปนี้ผิดที่ใด**

y = (1/2) * math.sqrt(x); → Miscalculation from (1/2) = int 0
y = sqrt(38.0); → Method not found
y = math.exp(2,3); → Wrong number of argument
y = math.sqrt(36); → Undefined class: math
y = math.sin(360); → สงสัยว่าอาจจะไม่ได้ใช้ degree ใน หน่วย radian i.e.. 4*PI)



# Increment (++) และ Decrement (--) operators

- **Increment operator (++):** ทำการเพิ่มค่าของ variable นั้นขึ้นอีก 1
  - count++ มีความหมายเช่นเดียวกับ count = count + 1;
- **Decrement operator (- -):** ทำการลดค่าของ variable นั้นลงไป 1
  - count-- มีความหมายเช่นเดียวกับ count = count - 1;
- **Pre VS Post (increment/decrement);**
  - **Preincrement:** ++count
  - **Postincrement:** count++

```
int i = 1;
System.out.println("i : " +  i);
// pre-increment
System.out.println("i : " + ++i);
// post-increment
System.out.println("i : " +  i++);
System.out.println("i : " + i);
```



# Shorthand assignment operators

| Operator | Usage | Meaning |
|---|---|---|
| += | a += b; | a = a + b; |
| -= | a -= b; | a = a - b; |
| *= | a *= b; | a = a * b; |
| /= | a /= b; | a = a / b; |
| %= | a %= b; | a = a % b; |

# Relational operators

- < น้อยกว่า (less than)
- <= น้อยกว่าหรือเท่ากับ (less than or equal to)
- == เท่ากับ (equal to)
- != ไม่เท่ากับ (not equal to)
- > มากกว่า (greater than)
- >= มากกว่าหรือเท่ากับ (greater than or equal to)



# Operator == VS equals

- **(obj1 == obj2) เป็นจริงเมื่อ contents ใน memory location สำหรับ 2 ตัวแปรนั้นมีค่าเท่ากัน**
  - ในกรณีที่เป็น primitive ค่าใน memory location คือค่าของตัวแปรเอง
  - ในกรณีที่เป็น object ค่าใน memory location คือค่า reference ที่อ้างไปยัง object นั้น
- **(obj.equal(obj2)) เป็นจริงเมื่อ ค่าของ 2 ตัวแปรนั้นเท่ากัน**
  - เราเรียกการเปรียบเทียบแบบนี้ว่า **equivalence test**



# Boolean Operators หรือ Logical Operators

- && และ (and)
- || หรือ (or)
- ! ปฏิเสธ (not)

| A | B | A&&B | A\|\|B | !A |
|---|---|---|---|---|
| เท็จ | เท็จ | เท็จ | เท็จ | จริง |
| เท็จ | จริง | เท็จ | จริง | จริง |
| จริง | เท็จ | เท็จ | จริง | เท็จ |
| จริง | จริง | จริง | จริง | เท็จ |



# Short-circuit Evaluation

- **And Operator (&&)**
  - นิพจน์มีค่าความจริงเป็นเท็จหากนิพจน์ย่อยทางซ้ายเป็นเท็จ โดยไม่ต้องทำการวิเคราะห์ค่าความจริงของนิพจน์ย่อยทางขวาของ and
- **Or Operator (| |)**
  - นิพจน์มีค่าความจริงเป็นจริงหากนิพจน์ย่อยทางซ้ายเป็นจริง โดยไม่ต้องทำการวิเคราะห์ค่าความจริงของนิพจน์ย่อยทางขวาของ or

# Operator Precedence rules

| Order | Group | Operator | กฎ |
|---|---|---|---|
| High | subexpression | ( ) | จากในสุดออกมาข้างนอก |
| ↓ | unary operator | - , + | ทำจากขวาไปซ้าย |
| | multiplicative op. | *, /, % | ทำจากซ้ายไปขวา |
| | additive operators | + , - | ทำจากซ้ายไปขวา |
| | comparison op | < , <= , > , >= | ทำจากซ้ายไปขวา |
| | equality op | ==, != | ทำจากซ้ายไปขวา |
| | "and" boolean op | && | ทำจากซ้ายไปขวา |
| | "or" boolean op | \|\| | ทำจากซ้ายไปขวา |
| Low | assignment op | = | ทำจากขวาไปซ้าย |



# ตัวอย่าง

- สมมติว่า x,y,z มีค่าเป็นจำนวนเลข ให้วิเคราะห์ค่าความจริงของนิพจน์ต่อไปนี้

  4 < 5 || 5 == 6 T

  2 < 4 && (false || 5 <=4) F

  x <= y && !(z != z) || x > y T

  x < y || z < y && y <= z x<y=?

- วิเคราะห์ว่านิพจน์ต่อไปนี้ผิดที่ใด
  - boolean done;

    done = x = y; (assigment)

    (Boolean VS arithmetic)
  - 2 < 4 && (3 < 5) + 1 == 3
  - boolean quit;

    quit = true;

    quit == (34 == 20) && quit; (Relational op.)
  - 70 <= x <= 100

    Rel. op. Is a binary op.



# Selection Statement: If

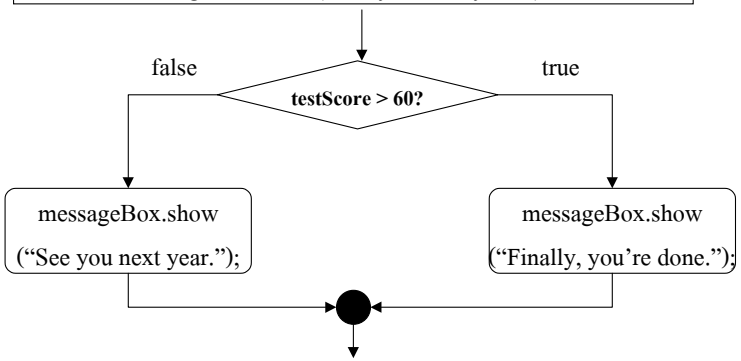
- if สร้างทางเลือกในการทำงานในโปรแกรม โดยที่กลุ่มของคำสั่งจะถูกทำงานก็ต่อเมื่อข้อแม้ที่ตั้งนั้น มีค่าความจริงเป็นจริง
- ข้อแม้ที่จะต้องทำการทดสอบก่อนนี้เรียกว่า **Boolean Expression**
- **Syntax**

```
if ( <boolean expression> ) {
    <then block statements>
}
else {
    <else block statements>
}
```



# Diagram: Control Flow of an if statement

```
If (testScore > 60)
        messageBox.show("Finally, you're done.);
else
        messageBox.show("See you next year.");
```



# กฎในการเขียน then และ else blocks

- ในกรณีที่กลุ่มของคำสั่ง วงเล็บปีกกาซ้าย { และ ขวา } จำเป็นต้องมี
- ในกรณีที่เป็นคำสั่งเพียงคำสั่งเดียวจะมีหรือไม่ก็ได้
- ไม่จำเป็นต้องมี ; ปิดท้ายหลังวงเล็บปีกกาขวา }
- โปรแกรมเมอร์ส่วนใหญ่นิยมครอบคำสั่งใน then หรือ else blocks ด้วยวงเล็บปีกกาเสมอ



# Style ในการเขียน if-then-else

- Style 1

```
if (<boolean expr) ) {
    ...
}
else {
    ..
}
```

- Style 2

```
if (<boolean expr) )
{
    ...
}
else
{
    ..
}
```



# Nested-If Statements: การซ้อน if ภายในอีก if

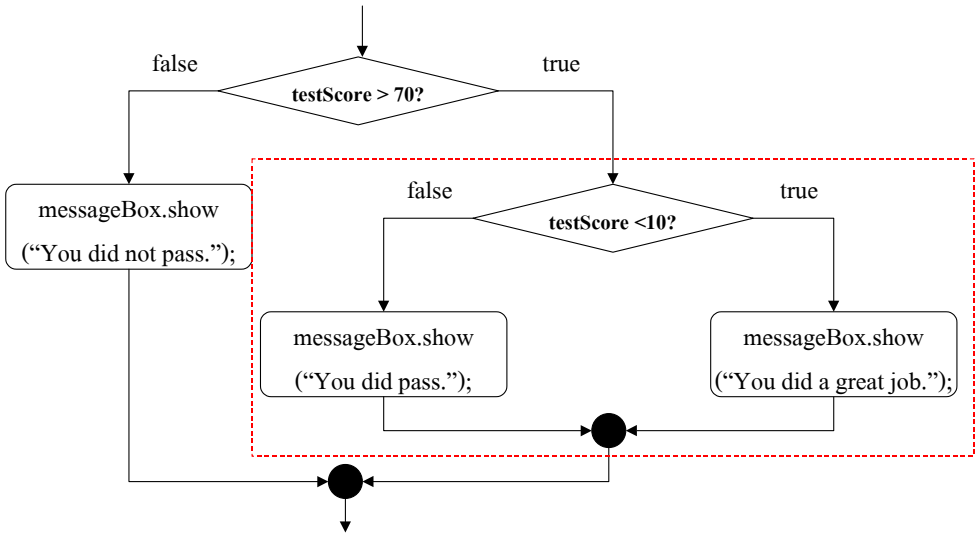
```
if ( testScore >= 70 ) {
    if( studentAge < 10) {
        mBox.show("You did an great job.");
    }
    else {
        // testScore < 70 and age >= 10
        mBox.show( "You did pass.");
    }
}
else {
    // testScore < 70
    mBox.show("You did not pass.");
}
```



# เขียนในอีกรูป

```
if (testScore >= 70 && studentAge < 10) {
    messageBox.show("You did a great job.");
}
else {
    // either testScore > 70 or age >= 10
    if (testScore >= 70) {
        messageBox.show("You did pass.");
    }
    else{
        messageBox.show("You did not pass.");
    }
}
```

# Diagram ของ Nested-If






# อีกตัวอย่าง

```
if (num1 < 0)
    if(num2 < 0)
          if(num3 < )
             negativeCount = 3;
          else
             negativeCount = 2;
    else negativeCount = 1;
else
    if (num2 < 0)
          if(num3 < 0)
            negativeCount = 2;
          else
            negativeCount = 1;
    else
          if(num3 < 0)
            negativeCount = 1;
          else
            negativeCount = 0;
```

- เขียนได้เท่ากับ

```
negativeCount = 0;
if(num1 < 0)
    negativeCount++;
if(num2 < 0)
    negativeCount++;
if(num3 < 0)
    negativeCount++;
```



# ระวังปัญหาของ Dangling else

```
if (x < y)
    if(x < z)
        messageBox.show("Hello World!!");
else
        messageBox.show("Good bye ja!!");
```

- **Java แปลข้างต้นว่า**

```
if(x < y)
    if(x < z)
        messageBox.show("Hello World!!");
    else
        messageBox.show("Good bye ja!!");
```



# Switch Statements

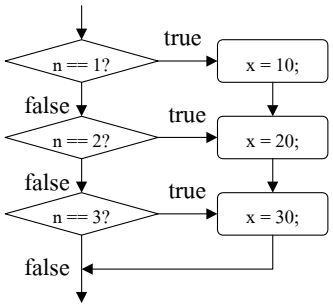
- **ทางเลือกโดยใช้ Switch กรณีที่มีหลายๆ กรณี**
- **Syntax**

```
Switch ( <arithmetic expression> ) {
      case <label 1> : <case body 1>
      .....
      case <label n>: <case body n>
            default  : <default body>
}
```

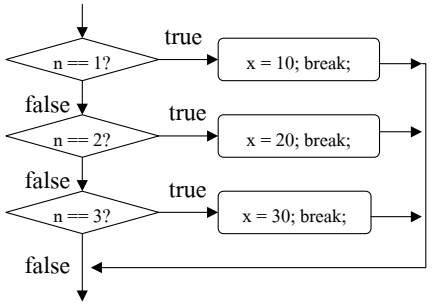
  - <arithmetic expression> เป็นชนิด char, short, byte, หรือ int
  - <label i> เป็นค่าคงที่

# Diagram: Control Flow of a switch statement

```
Switch(n) {
        case 1: x = 10;
        case 2: x = 20;
        case 3: x = 30;
}
```




```
Switch(n) {
        case 1: x = 10;
        case 2: x = 20;
        case 3: x = 30;
}
```






# ตัวอย่าง

```
int  ranking;  // valid value is 10 to 4
ranking = inputBox.getInteger();
switch (ranking) {
    case 10:
    case 9:
    case 8:  messageBox.show("Master"); break;
    case 7:
    case 6: messageBox.show("Journemar"); break;
    case 5:
    case 4: messageBox.show("Apprentire"); break;
    default: messageBox.show("Invalid"); break;
}
```



# ?: operator

- conditional operator,?: ใช้เป็น shorthand สำหรับการกำหนดค่าภายใต้เงื่อนไข
  - ตัวอย่าง

```
max = (a>b) ? a : b;
```

- ซึ่งจะเปรียบได้กับการทำ

```
if(a > b) {
   max = a;
}
else{
  max = b;
}
```

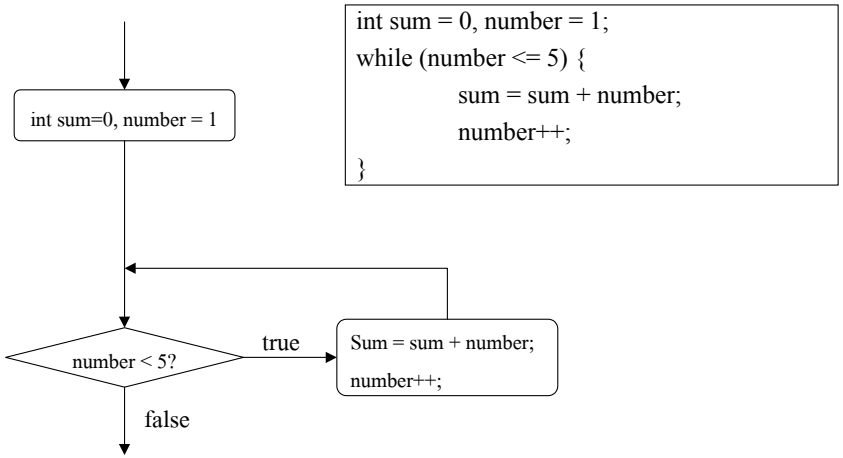


# Repetition Statements: while

- ใช้ควบคุมให้ทำกลุ่มของคำสั่งซ้ำตามจำนวนครั้งที่กำหนด หรือทำจนกว่าจะตรงกับข้อแม้ที่กำหนดให้
- while loop ตรวจสอบข้อแม้ก่อนที่จะเริ่มทำงาน (pretest loop)
- การทำซ้ำกระทำโดยใช้คำสั่ง while
- Syntax:

```
while ( <boolean expression> ){
    <statements>
}
```

# Diagram: Control Flow of a while statement






# ลักษณะการทำ loop

- Count controlled loop
  - นับจำนวนการ loop จนกว่าจะครบตามจำนวนที่กำหนด

```
while (number <= 100) {
  sum = sum + number;
  number = number + 1;
}
```

- Sentinel controlled loop
  - การ loop จนกระทั่งตรงกับข้อแม้ที่กำหนด

```
while (number <= 0) {
  sum = sum + number;
  number = inputBox.getInteger();
}
```



# Pitfalls in writing Repetition Statements

- Infinite Loop

Example:
```
int  product = 0;
while (product < 50000)
   product = product * 5;
```

Example: Overflow Error
```
int  count = 1;
while ( count != 10 )
  count = count + 2;
```

- Imprecise Loop Counter

Example: หลีกเลี่ยงการใช้ ค่า real เป็น counter
```
float count = 0.0f;
while( count != 1.0f )
  count = count + 0.33333f;
```

Example: ต้องการวน loop 10 ครั้ง
```
int  count = 1;
while ( count < 10 )
  count++;
```

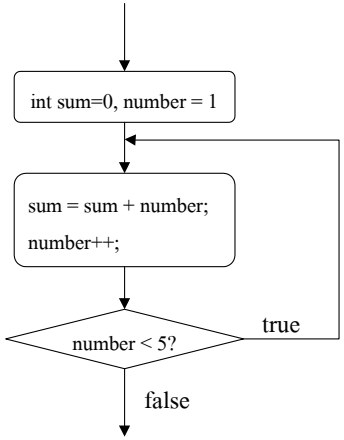


# Do-while Statements

- do -while loop เริ่มทำงานใน block ก่อนหนึ่งครั้งแล้วจึงตรวจสอบข้อแม้ที่กำหนด (Posttest loop)
- Syntax:

```
do {
  <statements>
} while ( <boolean expression> );
```

# Diagram: Control Flow of a while statement




```
int sum = 0, number = 1;
do{
        sum = sum + number;
        number++;
}while (number <= 5)
```



# แนะนำให้ใช้ตัวแปร Boolean ช่วยในการออก loop

```
sum = 0;
do {
    num = inputBox.getInteger( );
    if (num % 2 == 0)
        messageBox.show("Error: entered even number");
    else if (num == 0)
        messageBox.show("Sum = "+sum);
    else {
        sum += num;
        if(sum > 1000)
            messageBox.show("Sum is too large now.");
    }
} while ( !(num%2 == 0 || num == 0 || sum > 1000) )
```



# ใช้ตัวแปร Boolean

```
boolean repeat = true;
sum = 0;
do {
    num = inputBox.getInteger( );
    if ( num % 2 == 0) {
        messageBox.show("Error");
        repeat = false;
    }
    else if (num == 0) {
        messageBox.show("Sum = "+sum);
        repeat = false;
    }
    else{
        sum += um;
        if( sum > 1000) {
            messageBox.show("too large sum");
            repeat = false;
        }
    }
} while (repeat)
```



# for statements

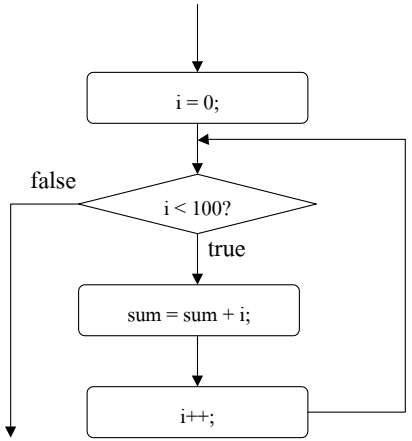
- ทั่วไปเหมาะสำหรับการทำซ้ำแบบ Count-controlled loop
- Syntax:

```
for (<initial> ; <boolean expr> ; <step> ){
    <statements>
}
```

- ใช้ comma (,) เพื่อแยก operation ได้
  - ตัวอย่าง

```
for ( int i = 0, j = 0 ; i < 10 ; i++,j=j+5 ) {
    System.out.println("i+j = " + (I+j));
}
```

# Diagram: Control Flow of a for statement




```
int i;
for ( i = 0; i < 100; i ++){
        sum = sum + i;
}
```



# Nested for Statements

```
for (int width = 11; width <= 20; width++) {
    oB.print(width + "  ");
    for (int length = 5; length <=25; length +=5){
        price = width * length * 19;
        oB.print("     " + price);
    }
    // finished one row then move to the next row
    oB.skipLine(1);
}
```



# ตัวอย่าง

- Code ต่อไปนี้ผิดที่ใด

```
int  sum = 0;
for (int i = 0; i <= 5; i++) {
   sum = sum + i;
   for (int i = 5; i > 0; i--){
        sum = sum + j;
   }
}
```



# break

- break statement ใช้ช่วยในการออกจาก loop ก่อนที่ fail condition

```
class CountWheat {
    public static void main( String args[ ] ) {
        int i, j = 1, k = 0;
        for ( i=1; i<=64; i++){
            j *= 2;
            if(j <= 0){
                System.out.println("Error: Overflow");
                break;
            }
            k += j;
            System.out.print( k + "\t  ");
            if ( i%4 == 0) System.out.println( );
        }
        System.out.println("All done!");
    }
}
```

# continue

- Continue ใช้ในกรณีที่ต้องการเริ่มต้น ของ innermosst loop ใหม่ เช่น อาจไม่ต้องการทำบางส่วนต่อจากนั้น โดยให้ไปเริ่มต้นใหม่
  - ตัวอย่าง

```
for (int i = 0; i < m.length; i++) {
   if (m[i] % 2 == 0) continue;
   // process odd elements...
}
```

- โดยทั่วไปจะใช้น้อยมาก



# Labeled loop

- การใช้ continue หรือ break จะเริ่มต้นใหม่หรือออกจาก innermost loop ในกรณีที่ต้องการให้ออกไปยัง loop นอกอื่น อาจใช้ labeled loop
  - ตัวอย่าง

```
iloop: for(int i=1; i < 3; i++){
   for(int j=1; j < 4; j++){
     if( j == 2) break  iloop;
     System.out.println( i + “,   ” + j);
   }
}
```



# Operator Summary

| Op | Purpose | Op | Purpose |
|---|---|---|---|
| + | addition of numbers, concatenation of String | - | bitwise NOT |
| += | add and assign numbers,concatenate and assign String | \| | bitwise OR |
| - | subtraction | \|= | bitwise OR and assign |
| -= | subtrac and assign | ^ | bitwise XOR |
| * | multiplication | ^= | bitwise XOR and assign |
| *= | multiply and assign | & | bitwise AND |
| / | division | &= | bitwise AND and assign |
| /= | devide and assign | ! | boolean NOT |
| % | take remainder | != | not equal to |
| %= | take remainder and assign | ++ | increment by one |
| > | greater than | -- | decrement by one |
| >= | greater than or equal to | >> | shift bits right with sign extension |
| < | less than | >>= | shif bits right with sign extension and assign |
| <= | less than or equal to | << | shif bits left |
| \|\| | boolean OR | <<= | shift bits left and assign |
| == | boolean equals | >>> | unsigned bit shift right |
| = | assignment | >>>= | unsigned bit shift right and assign |
| | | && | unsigned bit shift right and assign |



# โปรแกรมแรก

- **HelloWorld Application**

```
class HelloWorld{
   public static void main(String args[ ]){
     System.out.println(“Hello World”);
   }
}
```

- **Compile**

  javac HelloWorld.java

- **Run**

  java HelloWorld