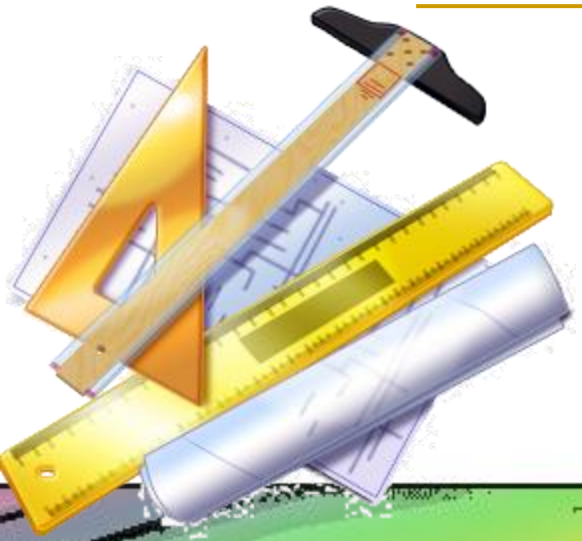
ทางเลือก และการทำซ้ำ

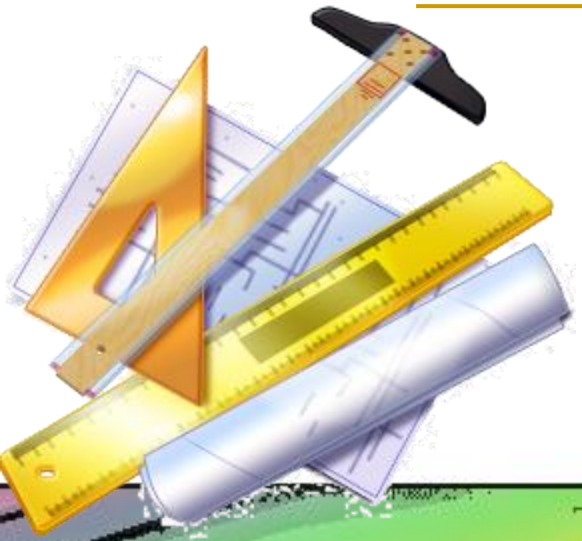


Lecture 4

Yaowadee Temtanapat

เยาวดี เต็มธนาภัทร์

ทางเลือก



Lecture 4

Yaowadee Temtanapat

เยาวดี เต็มธนาภัทร์

วัตถุประสงค์ของการเรียนในวันนี้

- สามารถควบคุมการเลือกการทำงานโดยการใช้คำสั่ง if หรือ switch
- เรียนรู้ในการเปรียบเทียบตัวเลข สายอักขระ และ objects
- เขียน Boolean expression โดยใช้ relational และ Boolean operators
- วิเคราะห์ค่าความจริงของ Boolean expression ได้อย่างถูกต้อง
- เขียนประโยค if ในลักษณะซ้อนกัน (Nested If) ได้อย่างถูกต้อง
- เข้าใจลำดับการทำงานในกรณีมีทางเลือกซ้อน
- เลือกคำสั่งที่เหมาะสมเพื่อควบคุมการเลือกได้อย่างเหมาะสมกับงาน

ลักษณะ Statements ในโปรแกรม

- คำสั่งเรียงลำดับ (Sequential Statements): ทำงานในลักษณะไหลเรียงลำดับ (Sequential control flow หรือ Sequential execution)
  - การทำงานในลักษณะต่อเนื่องจากคำสั่งหนึ่งไปอีกคำสั่งหนึ่งตามลำดับ
- คำสั่งในการควบคุม (Control statements): ประโยคที่ทำให้เกิดการเปลี่ยนทิศทางในการทำงาน แบ่งได้เป็น
  - คำสั่งทางเลือก (selection statements)
  - คำสั่งทำงานซ้ำ (repetition statements)

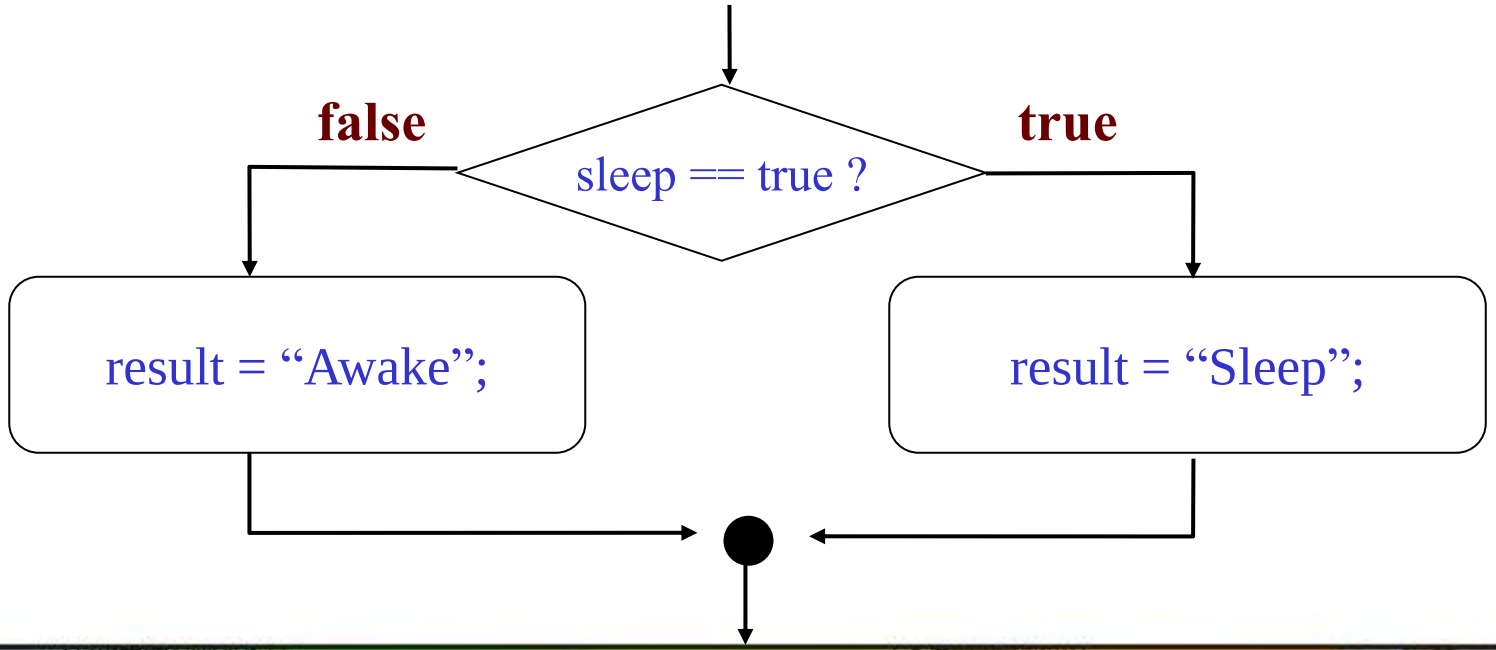
คำสั่ง if

- if สร้างทางเลือกการทำงานในโปรแกรม โดยกลุ่มคำสั่งจะถูกทำงานเมื่อเงื่อนไขที่ใช้ทดสอบเป็นจริง
- นิพจน์เงื่อนไขที่ใช้ทดสอบ เรียกว่า นิพจน์บูเลียน (Boolean Expression)
- Syntax

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

Diagram: การควบคุมการไหลของคำสั่ง if

```
String result;
if (sleep == true)
    result = "Sleep";
else
    result = "Awake";
```



กฎในการเขียนกลุ่มคำสั่ง (block statements)

- กลุ่มคำสั่งภายใต้คำสั่งควบคุมการไหล (เช่น if…else, for, do, while)
  - กรณีที่เป็นกลุ่มของคำสั่ง ต้องมีวงเล็บปีกกาซ้าย { และ ขวา } คร่อมกลุ่มคำสั่ง
  - ในกรณีที่เป็นเพียงคำสั่งเดียวจะมีปีกกาหรือไม่ก็ได้
  - หลังวงเล็บปีกกาขวา } ไม่จำเป็นต้องมี ; ปิดท้าย
- โปรแกรมเมอร์ส่วนใหญ่นิยมครอบคำสั่ง ด้วยวงเล็บปีกกาเสมอ

Style ในการเขียน if-then-else

- Style 1

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

- Style 2

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

เครื่องหมายเปรียบเทียบ (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)

เครื่องหมายตรรกะ (Boolean หรือ Logical Operators)

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

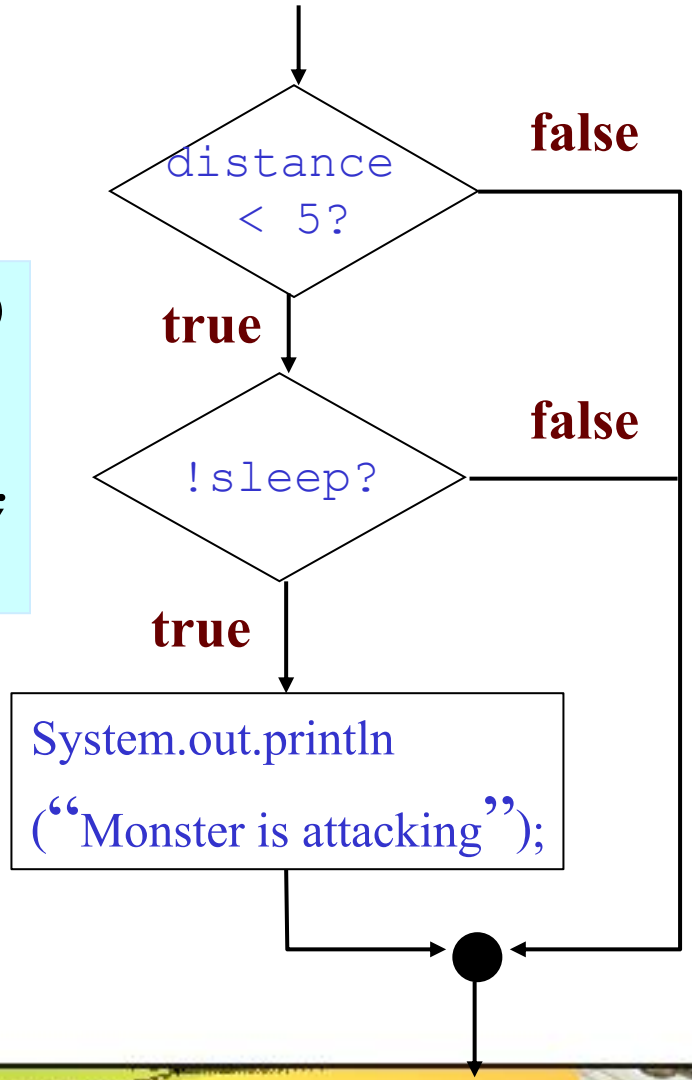
A	B	A&&B	A||B	!A
เท็จ	เท็จ	เท็จ	เท็จ	จริง
เท็จ	จริง	เท็จ	จริง	จริง
จริง	เท็จ	เท็จ	จริง	เท็จ
จริง	จริง	จริง	จริง	เท็จ

การประเมินแบบลัด (Short-circuit evaluation)

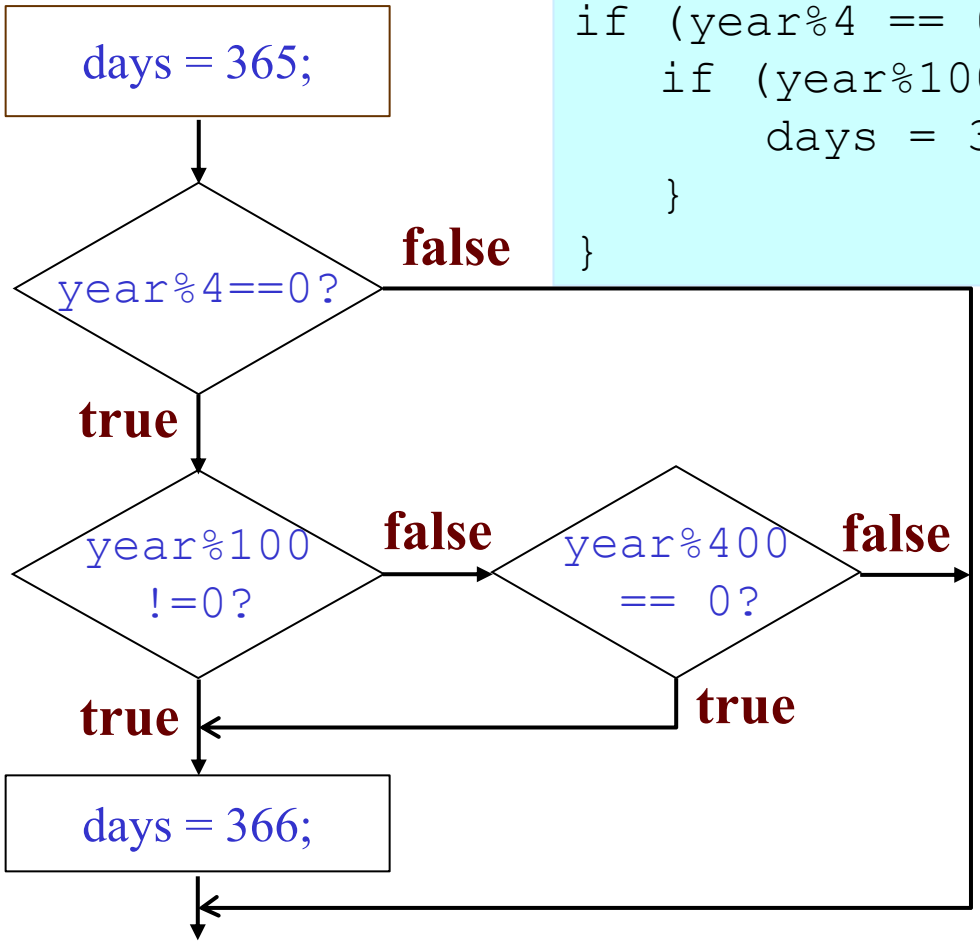
- And Operator (& &)
  - นิพจน์มีค่าความจริงเป็นเท็จหากนิพจน์ย่อยทางซ้ายเป็นเท็จ โดยไม่ต้องวิเคราะห์ค่าความจริงของนิพจน์ย่อยทางขวาของ and
- Or Operator (| |)
  - นิพจน์มีค่าความจริงเป็นจริงหากนิพจน์ย่อยทางซ้ายเป็นจริง โดยไม่ต้องวิเคราะห์ค่าความจริงของนิพจน์ย่อยทางขวาของ or

การลัดวงจรการทำงานของ &&

```
if (distance < 5 && !sleep)
{
   System.out.println(
    "Monster is attacking");
}
```



การลัดวงจรการทำงานของ ||




```
days = 365;
if (year%4 == 0) {
   if (year%100 != 0 || year%400 == 0){
       days = 366;
   }
}
```

กฎของลำดับในการทำงาน (Operator Precedence rules)

ลำดับ	กลุ่ม	Operator	กฎ
ก่อน ↓ หลัง	subexpression	( )	ทำจากในสุด ออกมาข้างนอก
	unary operators	-, +, !	ทำจาก ขวา ไป ซ้าย
	multiplicative op.	*, /, %	ทำจาก ซ้าย ไป ขวา
	additive operators	+, -	ทำจาก ซ้าย ไป ขวา
	comparison op.	<, <=, >, >=	ทำจาก ซ้าย ไป ขวา
	equality op.	==, !=	ทำจาก ซ้าย ไป ขวา
	“and” boolean op.	&&	ทำจาก ซ้าย ไป ขวา
	“or” boolean op.	||	ทำจาก ซ้าย ไป ขวา
	assignment op.	=	ทำจาก ขวาไปซ้าย

ตัวอย่าง

- ให้ x, y, z เป็นชนิดตัวเลข จง วิเคราะห์ค่าความจริงของนิพจน์ต่อไปนี้
  - `4 < 5 || 6 == 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;` assignment
  - `2 < 4 &&`
    `(3 < 5) + 1 == 3` Boolean VS arithmetic
  - `boolean quit;`
    `quit = true;`
    `quit == ( 34 == 20)`
    `&& quit;` Relational op.
  - `70 <= x <= 100`
    Rel. op. เป็น binary op.

การเปรียบเทียบเลขจำนวนจริง

- อาจเกิดข้อผิดพลาดจากการ roundoff

```
double r = Math.sqrt(2);
if (r*r == 2)
    System.out.println("sqrt(2) squared is 2");
else
    System.out.println("sqrt(2) squared is "+ r*r);
```

- ผลลัพธ์ที่ได้

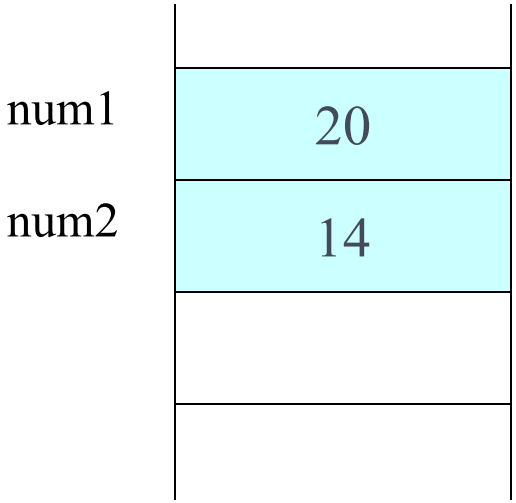
```
sqrt(2) squared is 2.0000000000000004
```

การเปรียบเทียบสายอักขระ

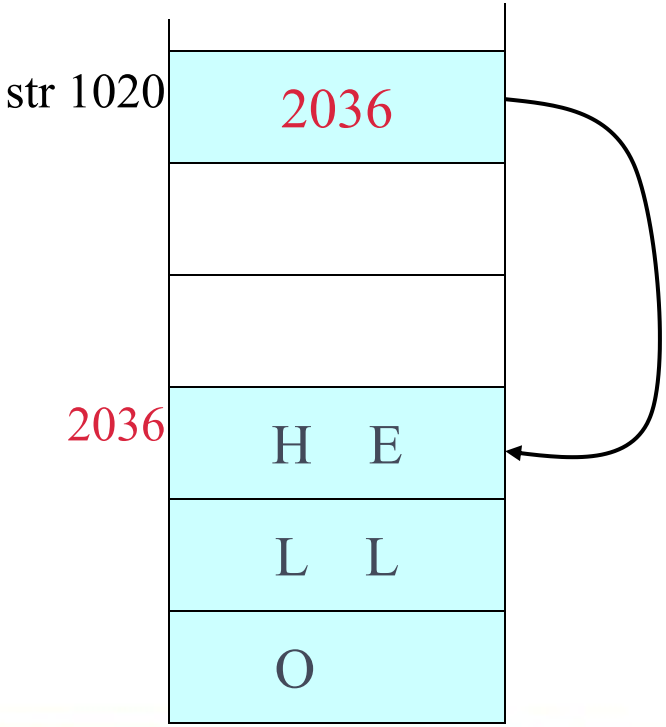
- การทดสอบโดย == เป็นการเปรียบเทียบตัวอ้างถึง
- การทดสอบโดย method equals: เป็นการเปรียบเทียบค่าของข้อมูล
- การทดสอบโดย method equalsIgnoreCase : เป็นการเปรียบเทียบค่าของข้อมูลสายอักขระโดยไม่คิดตัวพิมพ์ใหญ่หรือเล็ก
- การทดสอบโดย method compareTo : เป็นการเปรียบเทียบค่าของข้อมูลสายอักขระแบบมากกว่าหรือน้อยกว่า เรียงตามอักษรในลักษณะ unicode
  - คืน 0 ถ้าค่าข้อมูลเหมือนกัน
  - คืน < 0 ถ้าสายอักขระนั้นน้อยกว่าค่าพารามิเตอร์
  - คืน > 0 ถ้าสายอักขระนั้นมากกว่าค่าพารามิเตอร์

State ของหน่วยความจำระหว่าง Primitive VS Reference

```
int num1, num2;
num1 = 14;
num2 = num1;
num1 += 6;
```




```
String str;
str = "HELLO"
```

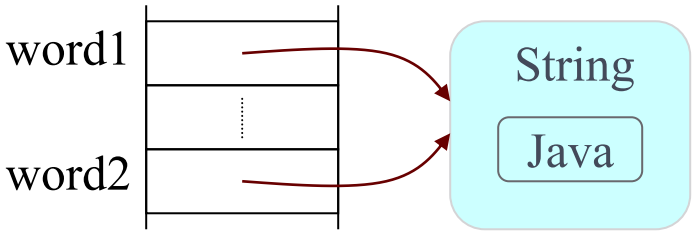


(str1 == str2) VS (str1.equals(str2))

- (str1 == str2) เป็นจริง เมื่อ contents ในหน่วยความจำสำหรับ 2 ตัวแปรนั้นมีค่าเท่ากัน
  - ในกรณีที่เป็น primitive ค่าในหน่วยความจำคือค่าของตัวแปรเอง
  - ในกรณีที่เป็น object ค่าในหน่วยความจำคือค่า reference ที่อ้างไปยัง object นั้น
- (str1.equals(str2))เป็นจริง เมื่อค่าของ 2 ตัวแปรนั้นเท่ากัน
  - เราเรียกการเปรียบเทียบแบบนี้ว่า ***equivalence test***

ข้อแตกต่างระหว่าง equality test และ equals method

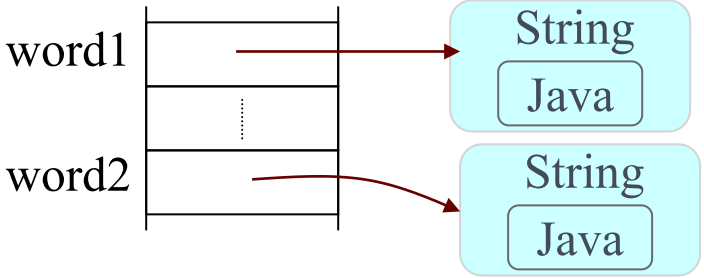
Case A: อ้างถึง object เดียวกัน



word1 == word2 เป็น จริง

word1.equals(word2) เป็น จริง

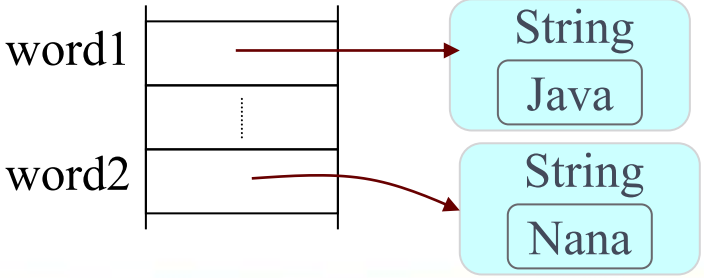
Case B: อ้างถึง object ต่างกันแต่มีค่าของ String เท่ากัน



word1 == word2 เป็น เท็จ

word1.equals(word2) เป็น จริง

Case C: อ้างถึง object ต่างกันและค่าของ String ต่างกัน

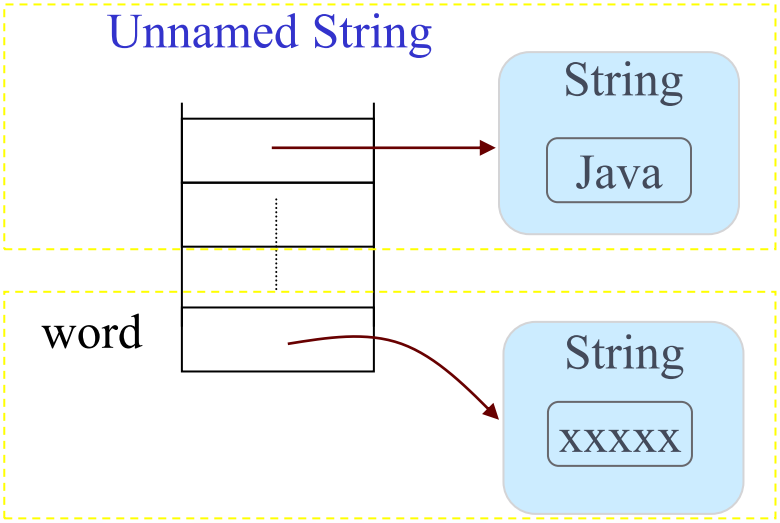


word1 == word2 เป็น เท็จ

word1.equals(word2) เป็น เท็จ

Unnamed String Object

- Unnamed String object จะถูกสร้างขึ้นโดยอัตโนมัติเมื่อมีการกำหนดค่าคงที่ (literal string) ใน Java Code
- จาวารู้เองว่าหน่วยความจำที่อ้างถึง Unnamed String นั้นอยู่ที่ใด นั่นคือ



word == “Java” เป็น เท็จ

word.equals(“Java”) เป็น ?

ตัวอย่างการเปรียบเทียบสายอักขระ

```
String s1 = new String("Hello");
String s2 = new String("Hello");
if (s1 == s2)
    System.out.println("s1 == s2");
else
    System.out.println("s1 != s2");
if (s1.equals(s2))
    System.out.println("s1 equals s2");
else
    System.out.println("s1 is not equals s2");
if (s1.compareTo("Hell") < 0)
    System.out.println("s1 < Hell");
else
    System.out.println("s1 >= Hell");
```

```
s1 != s2
```

```
s1 equals s2
```

```
s1 >= Hell
```

ตัวอย่างการเปรียบเทียบสายอักขระ (ต่อ)

```
String s3 = "Hello";
String s4 = "Hello";
if (s3 == s4)
    System.out.println("s3 == s4");
else
    System.out.println("s3 != s4");
if (s1 == s3)
    System.out.println("s1 == s3");
else
    System.out.println("s1 != s3");
if (s3 == "Hello")
    System.out.println("s3 == Hello");
else
    System.out.println("s3 != Hello");
```

```
S3 == s4
```

```
s1 != s3
```

```
s3 == Hello
```

การเปรียบเทียบ Objects

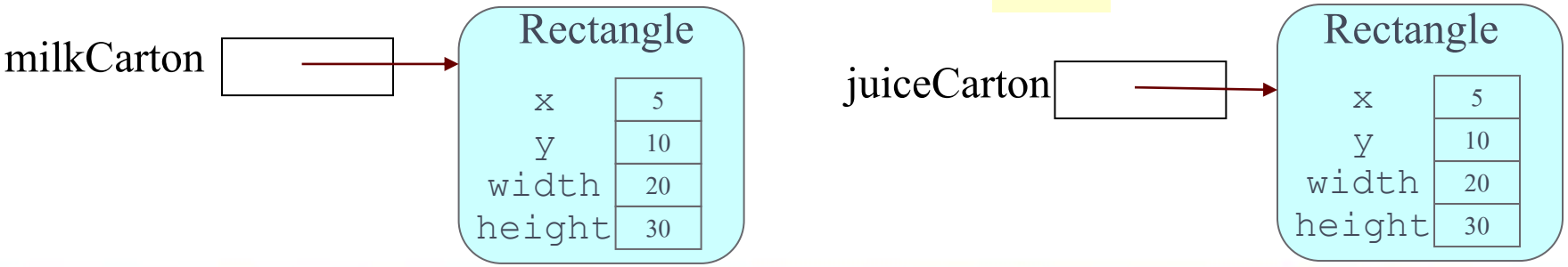
- การทดสอบโดย == เปรียบเทียบตัวแปรอ้างถึง objects
- การทดสอบโดย equals เป็นการเปรียบเทียบที่ผู้ใช้ต้องกำหนดขึ้นเอง สำหรับ class ที่ผู้ใช้สร้างขึ้น (เรียนต่อไปในบทที่ 9)

```
Rectangle milkCarton = new Rectangle(5, 10, 20, 30);
Rectangle juiceCarton = new Rectangle(5, 10, 20, 30);
```

```
milkCarton == juiceCarton
milkCarton.equals(juiceCarton))
```

False

True



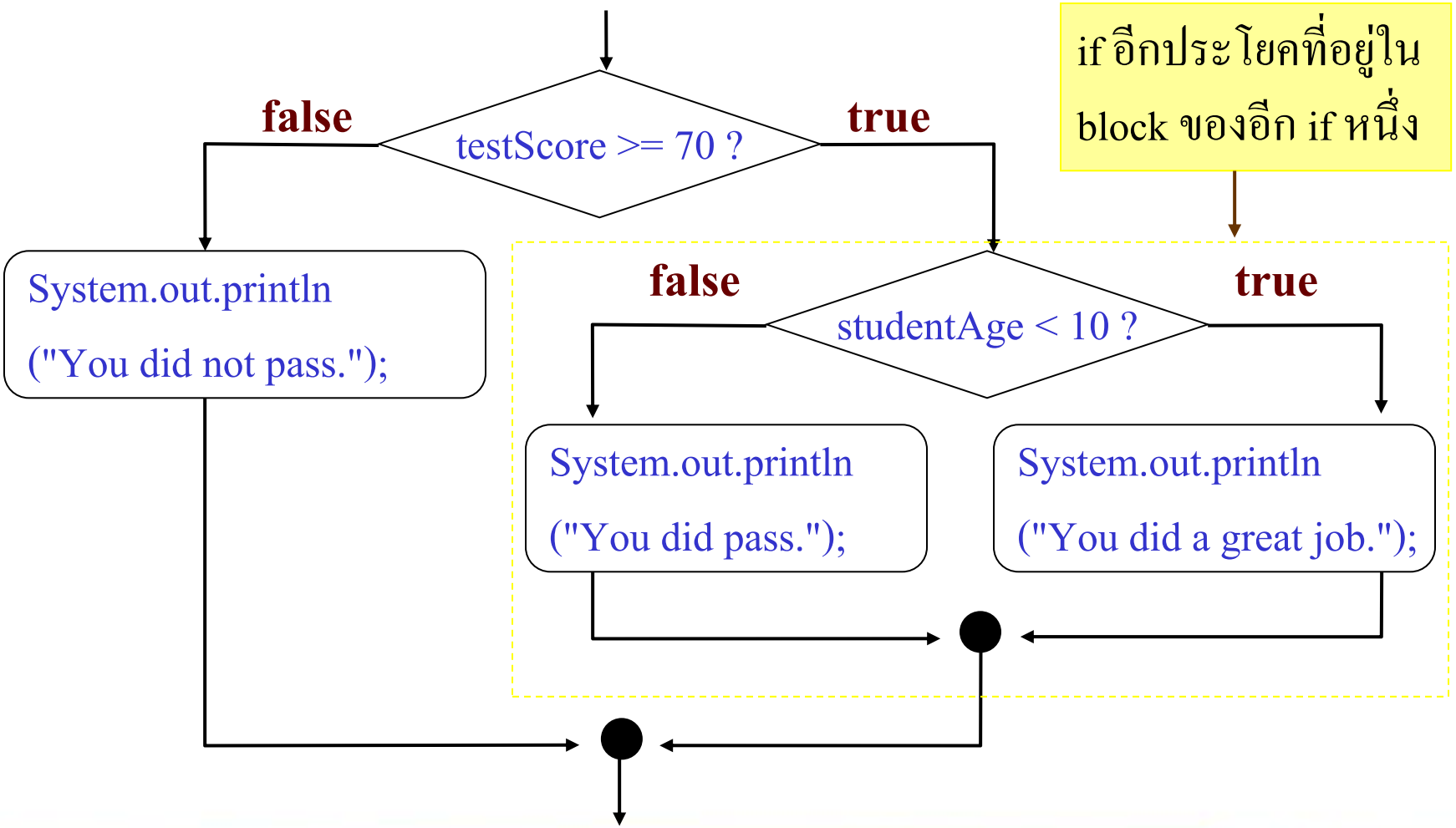
Nested-If Statements: การซ้อน if ภายในอีก if

```
if ( testScore >= 70) {
    if (studentAge < 10) {
        System.out.println("You did a great job.");
    }
    else {
        // testScore >= 70 and age >= 10
        System.out.println("You did pass.");
    }
}
else {
    // testScore < 70
    System.out.println("You did not pass.");
}
```

เขียนในอีกรูป

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

Diagram ของ if แบบซ้อน (Nested-If)



อีกตัวอย่าง

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

ระวังปัญหาของการไม่ครบคู่ของ else (Dangling else)

```
if (distance > 50)
    if (direction == 1)
        System.out.println("Long Walk Vertically");
else
    System.out.println("Normal Walk");
```

- Java แปลข้างต้นว่า

```
if (distance > 50)
    if (direction == 1)
      System.out.println("Long Walk Vertically");
    else
      System.out.println("Normal Walk");
```

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

- Increment operator (++) : เพิ่มค่าของ variable นั้นขึ้นอีก 1
  - count++ มีความหมายเช่นเดียวกับ count = count + 1;
  - สามารถใส่หน้า หรือหลังตัวแปรได้ แต่ให้ความหมายต่างกัน เช่น ++count หรือ count++
- Decrement operator (--) : ลดค่าของ variable นั้นลงไป 1
  - count-- มีความหมายเช่นเดียวกับ count = count - 1;
  - สามารถใส่หน้า หรือหลังตัวแปรได้ แต่ให้ความหมายต่างกัน

Shorthand assignment operators

Operator	การใช้	ความหมาย
+=	`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);`

คำสั่ง Switch

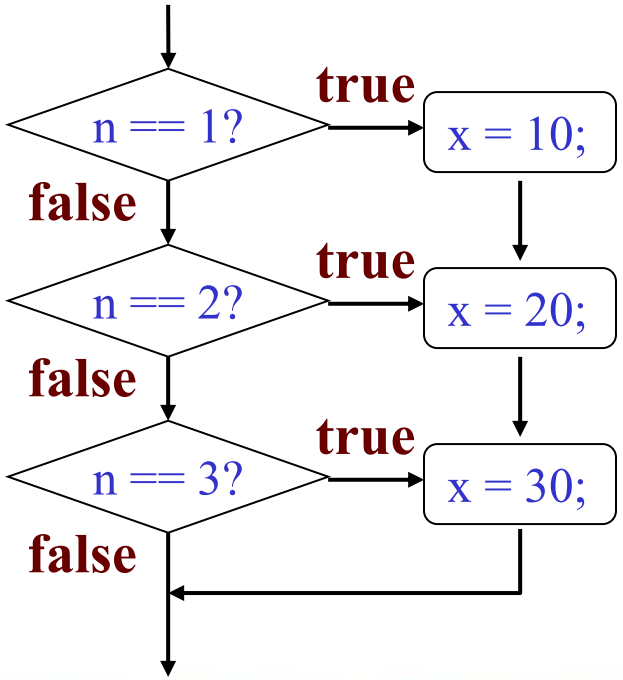
- ทางเลือกโดยใช้ Switch กรณีที่มีหลาย ๆ กรณี
- Syntax

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

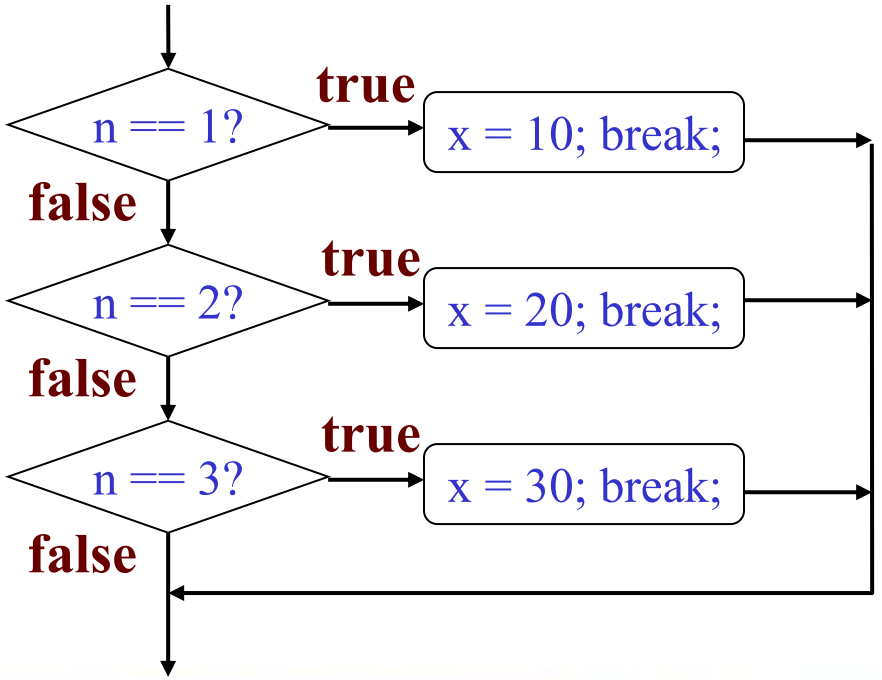
  - <arithmetic expression> เป็นชนิด char, short, byte, int หรือ String
  - <label i> เป็นค่าคงที่

Diagram: การควบคุมการไหลของ switch statement

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




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



ตัวอย่าง

```
int number; // valid value is 5 to 0
number = scanner.nextInt();
switch (number) {
    case  5:
            System.out.println("Very good"); break;
    case  4:
    case  3:
            System.out.println("Good"); break;
     case  2:
            System.out.println("Fair"); break;
    case  1:
    case  0:
            System.out.println("Poor"); break;
    default: System.out.println("N/A"); break;
}
```

นิพจน์ เงื่อนไข ?

- นิพจน์ เงื่อนไข ? สำหรับให้ค่า
- Syntax
  - *condition*? *trueValue*: *falseValue*;

```
public static int max(int x, int y) {
     return x > y? x: y;
}
public static void lessThan5(int x) {
     String s = x < 5? x+ " less than": x + " not less than";
     System.out.println(s);
}
```

นิพจน์ Switch (v.13)

- นิพจน์ switch สำหรับให้ค่า (ไม่ต้องมีประโยค `break;`)
- Syntax

```
switch ( arithmeticExpression ) {
        case label_1 -> value1;
        case label_2, label_3 -> { blockStmt };
        ...
        default -> defaultValue;
}
```

- ตัวอย่าง

```
public static String expresionSwitch(int num) {
        return switch(num) {
                case 1 -> "Hello";
                case 2, 3 -> "Good";
                case 4 -> { System.out.println(4);
                yield "Bye"; }
                default -> "Unknown";
        };
}
```

หรือจะใช้ประโยค `yield` แทน

```
return switch (num) {
        case 1: yield "Hello";
        case 2, 3:
        yield "Good";
        case 4: System.out.println(4);
                yield "Bye";
        default: yield "Unknown";
};
```

ตัวอย่างการใช้ประโยคทางเลือก

- การเขียนโปรแกรม ที่ใช้ถามตอบจากผู้ใช้

```
Do you want to continue?
```

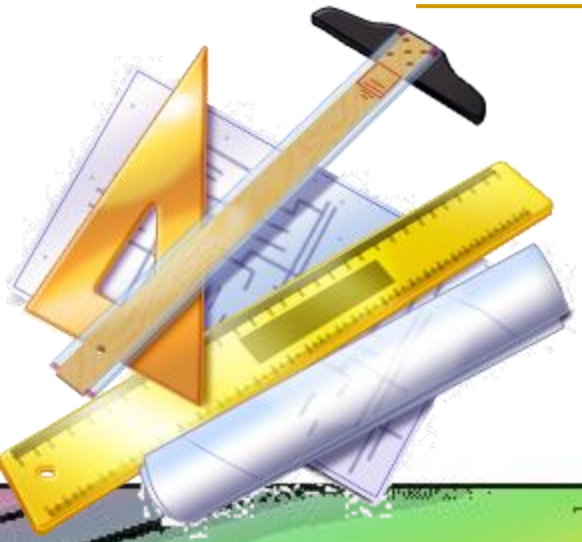
- ถ้าผู้ใช้ตอบ Y, Yes, Ok, หรือ Sure เขียนข้อความ

```
Sorry, it's still under construction!! Come back
   later.
```

- ถ้าผู้ใช้ตอบ N, หรือ No เขียนข้อความ

```
See you next time.
```

คำสั่งเพื่อให้ทำงานซ้ำ (Repetition Statements)



Lecture 4

Yaowadee Temtanapat

เยาวดี เต็มธนาภัทร์

วัตถุประสงค์ของการเรียนในวันนี้

- ควบคุมการทำงานซ้ำโดยการใช้คำสั่ง while, for, do-while
- เลี่ยงการวนซ้ำแบบไม่รู้จบ และความผิดพลาดของการวน
- เรียนรู้การทำซ้ำในลักษณะซ้อนกัน (Nested Loop)
- เรียนรู้การใช้ประโยค break และ continue ร่วมกับคำสั่งการทำงานซ้ำ
- เรียนรู้การอ่านข้อมูลเข้าด้วย String
- เรียนรู้การใช้วัตถุ StringTokenizer และ Random

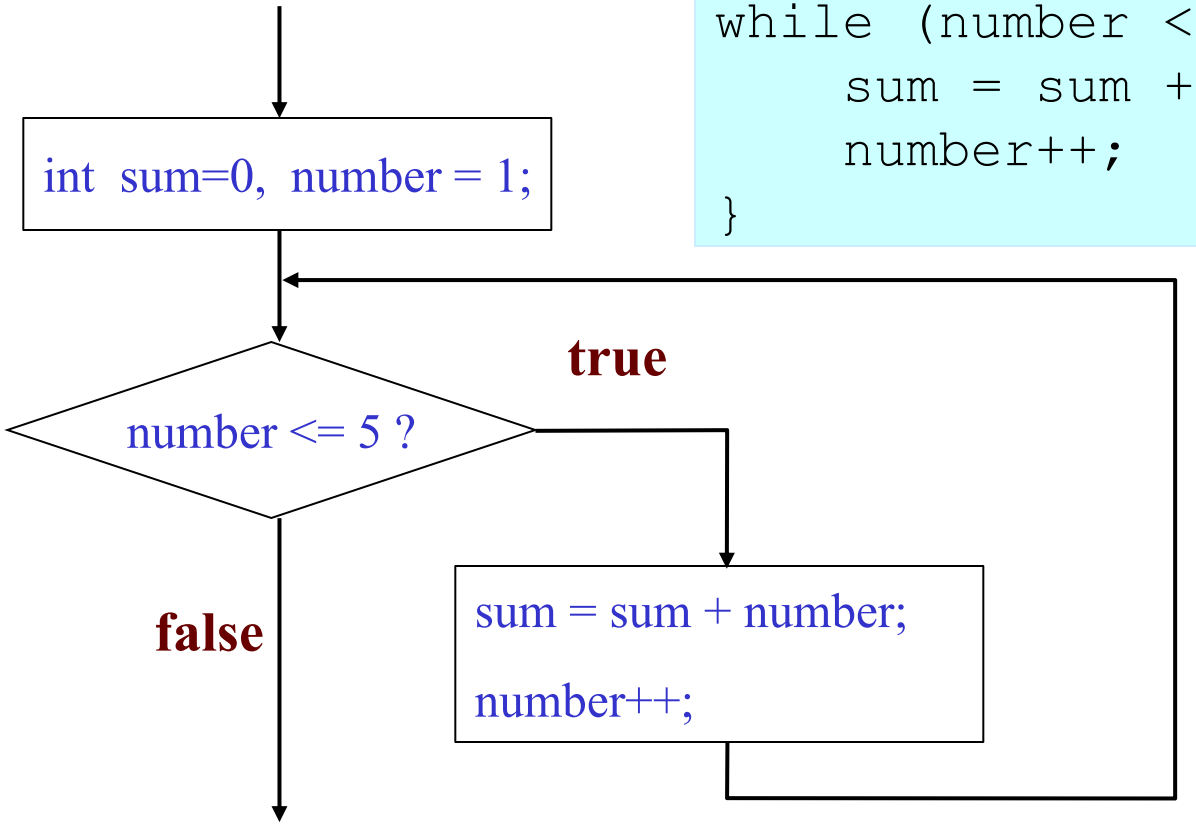
ประโยค while

- ใช้ควบคุมให้ทำกลุ่มคำสั่งซ้ำตามจำนวนครั้งที่กำหนดหรือทำจนกว่าจะไม่ตรงกับเงื่อนไขที่กำหนดให้
- while loop ตรวจสอบเงื่อนไขก่อนที่จะเริ่มทำงาน (pretest loop)
- การทำซ้ำ ทำโดยใช้คำสั่ง while
- Syntax:

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

Diagram: การควบคุมการไหลของประโยค while

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



ลักษณะการทำ loop

- Count controlled loop
  - นับจำนวนการ loop จนกว่าจะครบตามจำนวนที่กำหนด

```
int number = 0;
while (number < 100) {
    System.out.println("round "+number+":My message.");
    number++;
}
```

- Sentinel controlled loop
  - การ loop จนกระทั่งไม่ตรงกับเงื่อนไขที่กำหนด

```
int sum = 0, number = 0;
while (number >= 0) {
    sum = sum + number;
    number = scanner.nextInt();
}
```

ข้อผิดพลาดที่พบบ่อยในการเขียนโค้ดการทำซ้ำ

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:

หลีกเลี่ยงการใช้ค่า real เป็น counter

```
float count = 0.0f;
while (count != 1.0f)
    count = count + 0.333333f;
```

Off-by-One error

Example:

ต้องการวน loop 10 ครั้ง

```
int count = 1;
while (count < 10)
    count++;
```

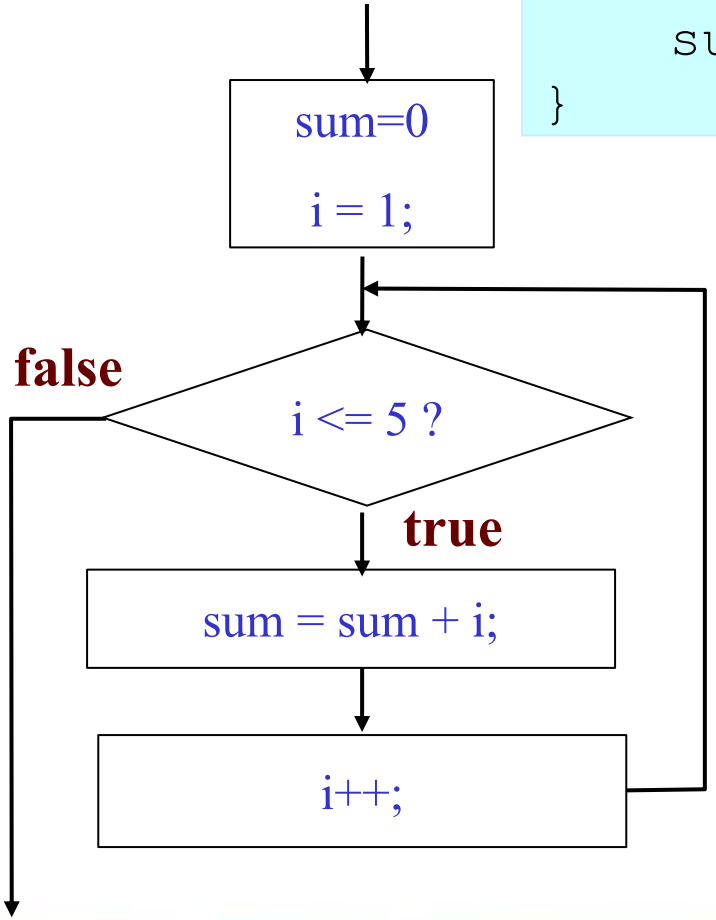
ประโยค for

- ทั่วไปเหมาะสำหรับการทำซ้ำแบบ Count-controlled loop
- Syntax:

```
for (<initial>; <condition>; <update>) {
    <statements>
}
```

Diagram: การควบคุมการไหลของประโยค for

```
int sum = 0;
for (int i=1; i<=5; i++)
{
    sum = sum + i;
}
```



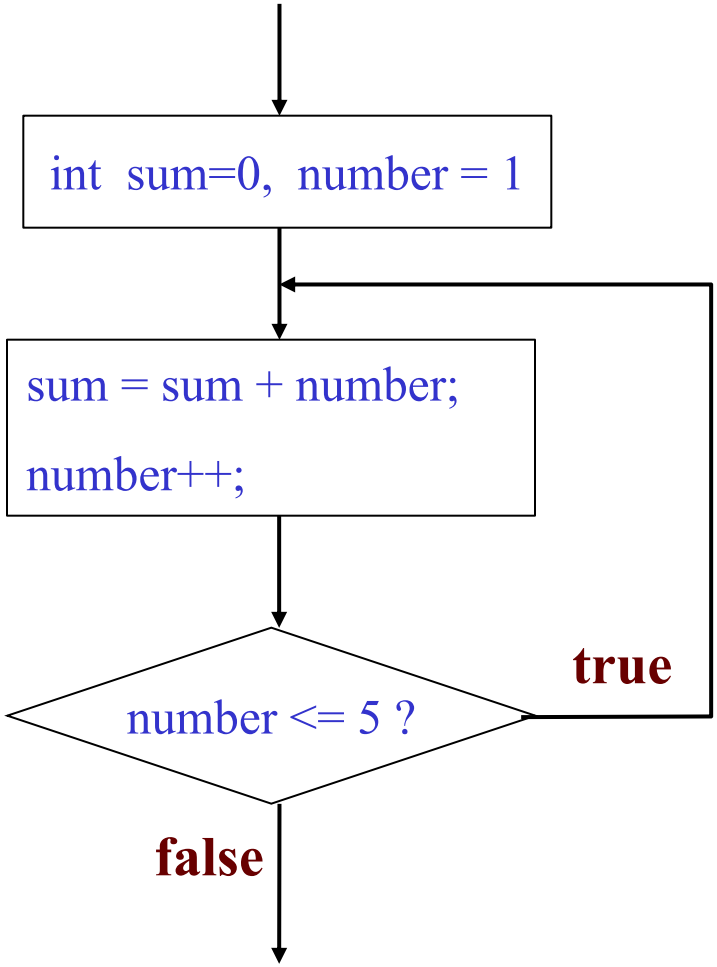
หมายเหตุ Scope ของตัวแปรที่กำหนดภายใน loop: สิ้นสุดอยู่เพียงใน loop และจะไม่รู้จักภายนอก loop

ประโยค do-while

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

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

Diagram: การควบคุมการไหลของประโยค do-while




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

การซ้อนประโยค for

```
for (int width = 11; width <= 20; width++) {
    System.out.print(width + "  ");

    for (int length = 5; length <= 25; length += 5) {
        price = width * length * 19;
        System.out.print("    " + price);
    }

    // finished one row then move to the next row
    System.out.println();
}
```

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 + i;
    }
}
```

Average.java (คำนวณค่าเฉลี่ยของ Input ที่ป้อนให้)

```
public class Average {
  public static void main(String[] args) {
    // create and initial
    Scanner console = new Scanner(System.in);
    System.out.println("Enter data.");

    double sum = 0;
    int count = 0;

    // compute sum of all input values
    boolean done = false; // stop flag
```

Average.java (คำนวณค่าเฉลี่ยของ Input ที่ป้อนให้)

```
    while (!done) {
        Scanner inputLine = new Scanner(scan.nextLine());
        if (!inputLine.hasNext())
            done = true;
        else {
            double x = inputLine.nextDouble();
            sum = sum + x;
            count++;
        }
     }
    // compute average
    if (count == 0)
       System.out.println("No data");
    else
       System.out.printf("%s = %g", "Average",
          sum/count);
  }
}
```

break

- ประโยค break ใช้ช่วยให้ออกจาก loop ก่อนเงื่อนไขของ loop เป็นเท็จ

```
while (true) {
    Scanner inputLine = new Scanner(scanner.nextLine());
   // if found blank line then done
   if(!inputLine.hasNext())
     break;
   else{
     // process each number in the input line
     while(inputLine.hasNext()){
            double x = inputLine.nextDouble();
            sum = sum+x;
            count++;
     }
   }
}
```

continue

- continue ใช้ในการกระโดดไปยังตอนจบของ loop ปัจจุบัน ในกรณีที่ต้องการเริ่มต้น innermost loop รอบถัดไปเลย เช่นอาจต้องการข้ามคำสั่งบางส่วน
  - ตัวอย่าง

```
for (int i = 0; i < 20; i++) {
  int value = (int)(Math.random()*100);
  if(value % 2 == 0)
        continue;
  // process odd elements…
  i++;
  // continue jumps here
}
```

- โดยทั่วไป ใช้ไม่บ่อย

Labeled loop

- การใช้ continue หรือ break จะเริ่มต้นใหม่หรือออกจาก loop ชั้นในที่ใกล้ที่สุด ในกรณีที่ต้องการให้ออกที่ไปยัง loop นอกอื่น อาจใช้ labeled loop โดยกำหนด label ไว้ที่ loop นอก
  - ตัวอย่าง

```
iloop:
for (int i = 0; i < 100; i++) {
   jLoop:
   for (int j = 0; j < 100; j++) {
     if (someCondition) break iloop;
   }
}
```

StringTokenizer

- StringTokenizer ทำหน้าที่ในการแตกสายอักขระออกเป็น สายของอักขระย่อย (tokens) โดยสามารถแยก words, ตัวเลข และตัวอักขระพิเศษ
  - hasMoreTokens( ) : ตรวจสอบว่ามี token เหลืออีกหรือไม่
  - nextToken( ) : คืนค่า token ตัวต่อไป
  - countTokens( ) : นับจำนวน tokens

```
String inputLine = "Mary has a little lamb.";
StringTokenizer st = new StringTokenizer(inputLine);
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

```
// Words.java
import java.util.StringTokenizer;
import java.util.Scanner;
public class Words {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter Words:");
    int count = 0;
    boolean done = false;
    while (!done) {
      String inputLine = scanner.nextLine();
      if(inputLine.length() == 0)
         done = true;
      else{
         // break up input line into words
         StringTokenizer tokenizer = new StringTokenizer(inputLine);
         // add number of words in the line
         count += tokenizer.countTokens();
      }
    }
    System.out.println(count + "words");
  }
}
```

Methods บาง method ของ String

- `length()` : ใช้ในการหาค่าความยาวของ String
- `charAt(index)` : คืนค่า character ในตำแหน่งที่ระบุ
  - โดยที่ index จะมีค่าระหว่าง 0 ถึง length( ) - 1
- `toLowerCase()` : คืนค่า String เป็นตัวพิมพ์เล็ก
- `toUpperCase()` : คืนค่า String เป็นตัวพิมพ์ใหญ่

ตัวอย่างการทำ reverse คำ

```
public class Reverse {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     System.out.println("Please enter a string:");
     String s = scanner.nextLine();
     String r = "";
     for (int i = 0; i < s.length(); i++) {
       char ch = s.charAt(i);
       r = ch + r; // add ch in front
     }
     System.out.println(s + " reversed is " + r);
  }
}
```

การสร้างเลขสุ่ม (Random Number)

- การสร้างเลขสุ่ม โดยใช้ Random Class ใน java.util:

```
import java.util.Random;
public class Random100 {
    public static void main(String[] args) {
        Random generator = new Random();
        // generate random number ten times
        for (int i = 1; i <= 10; i++) {
          int d = 1 + generator.nextInt(99);
          System.out.print(d + " ");
        }
        System.out.println();
    }
}
```

สร้าง Object เลขสุ่ม

สร้างเลขสุ่มค่าระหว่าง 0 ถึง 99

สรุปการเรียนในวันนี้

- เรียนรู้ประโยคควบคุมทางเลือกโดยการใช้คำสั่ง if หรือ switch
- เรียนรู้การเปรียบเทียบตัวเลข สายอักขระ และ objects
- เขียน Boolean expression โดยใช้ relational และ Boolean operators
- วิเคราะห์ค่าความจริงของ Boolean expression ได้อย่างถูกต้อง
- เขียนประโยค if ในลักษณะซ้อนกัน (Nested If) ได้อย่างถูกต้อง
- เข้าใจลำดับการทำงานในกรณีมีทางเลือกซ้อน
- เลือกคำสั่งที่เหมาะสมเพื่อควบคุมการเลือกได้อย่างเหมาะสมกับงาน

สรุปการเรียนในวันนี้

- เรียนรู้ประโยคการทำงานซ้ำ ด้วยคำสั่ง while, for, do-while
- เลี่ยงการวนซ้ำแบบไม่รู้จบ และความผิดพลาดของการวน
- เรียนรู้การทำซ้ำในลักษณะซ้อนกัน (Nested Loop)
- เรียนรู้การใช้ประโยค break และ continue ร่วมกับคำสั่งการทำงานซ้ำ
- เรียนรู้การอ่านข้อมูลเข้าด้วย String
- เรียนรู้การใช้วัตถุ StringTokenizer และ Random