Java Programming

U Hou Lok

Department of Computer Science and Information Engineering,
National Taiwan University

Java 272
8–19 Aug., 2016

Arithmetic Compound Assignment Operators

++	Increment
+=	Addition assignment
– =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

- Note that these shorthand operators are not common in languages whose level is higher than Java, say Matlab, R, and Python.

Example

```
1     ...
2     int x = 1;
3     System.out.println(x); // output 1
4     x = x + 1;
5     System.out.println(x); // output 2
6     x += 2;
7     System.out.println(x); // output 4
8     x++; // equivalent to x += 1 and x = x + 1
9     System.out.println(x); // output 5
10    ...
```

- The compound assignment operators are also useful for char values.[1]
- For example,

```
1  ...
2  char s = 'a';
3  System.out.println(s); // output 'a'
4  s += 1;
5  System.out.println(s); // output 'b'
6  s++;
7  System.out.println(s); // output 'c'
8  ...
```

[1] Contribution by Mr. Edward Wang (Java265) on May 1, 2016.

++x vs. x++

- The expression ++x first increments the value of x and then returns x.
- Instead, the expression x++ first returns the value of x and then increments itself.
- For example,

```
1  ...
2  int x = 1;
3  System.out.println(++x); // output 2; aka preincrement
4  System.out.println(x); // output 2
5
6  int y = 1;
7  System.out.println(y++);  // output 1; aka postincrement
8  System.out.println(y); // output 2
9  ...
```

- We will use these notations very often.

Bitwise Operators

- Java defines several bitwise operators that can be applied to the integer types: long, int, short, char, and byte.

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Example

```
1   ...
2   int a = 3; // 0 + 2 + 1 or 0011 in 2's complement
3   int b = 6; // 4 + 2 + 0 or 0110 in 2's complement
4   int c = a | b;
5   int d = a & b;
6   int e = a ^ b;
7   int f = (~a & b) | (a & ~b);
8   int g = ~a & 0x0f;
9   System.out.println(a >>= 2);
10  System.out.println(b <<= 2);
11  ...
```

- c = ? d = ? e = ? f = ? g = ? h = ? i = ? 
- In Line 9 and 10, what are the results?

Operator Precedence[2]

Precedence	*Operator*
↓	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!` (Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Comparison)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

[2]See Table3-10 in YDL, p. 116.

Using Parentheses

- Parentheses raise the precedence of the operations that are inside them.
- Parentheses remove ambiguity of your program.

Scanner Objects

- It is not convenient to modify the source code and recompile it for a different radius.
- Reading from the console enables the program to receive an input from the user.
- A **Scanner** object provides input methods, say the input received from the user or the database.
- Java uses **System.in** to refer to the standard input device, by default, the keyboard.

Example: Reading Input From The Console

Write a program which receives a number as input, and outputs the area of the circle.

```
1  import java.util.Scanner;  // import the Scanner class
2  ...
3  Scanner input = new Scanner(System.in); // create a Scanner
       object for receiving data from the keyboard
4  ...
5  r = input.nextInt(); // return a double value
6  ...
7  input.close();
8  ...
```

```
1   import java.util.Scanner;
2
3   public class ComputeAreaWithConsoleInput {
4   public static void main (String[] args) {
5   Scanner input = new Scanner(System.in);
6   System.out.println("Enter r?");
7   int r = input.nextInt();
8   double area = r * r * 3.14;
9   System.out.println(area);
10  input.close();
11  }
12  }
```

- In the listing, Line 5 is used to create a **Scanner** object by the new operator, and then return its memory address assigned to the variable *input*.[3]
- So the variable *input* is a reference variable.
- We will discuss the objects and reference variables in OO lectures.

[3] All objects are resided in the heap of the memory.

Methods Within Scanner Objects[4]

Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.
`next()`	reads a string that ends before a whitespace character.
`nextLine()`	reads a line of text (i.e., a string ending with the *Enter* key pressed).

[4] See Table 2-1 in YDL, p. 38.

Example: Sample Mean and Sample Standard Deviation

Write a program which calculates the sample mean and the sample standard deviation of 3 numbers.

- Sample mean of 3 numbers is given by $\overline{x} = \left(\sum_{i=1}^{3} x_i\right)/3$.
- Also, the sample standard deviation is given by

$$S = \sqrt{\frac{\sum_{i=1}^{3}(x_i - \overline{x})^2}{2}}.$$

- You may use these two methods:
  - Math.pow(double x , double y) for x^y
  - Math.sqrt(double x) for $\sqrt{x}$
- See more methods within Math class.

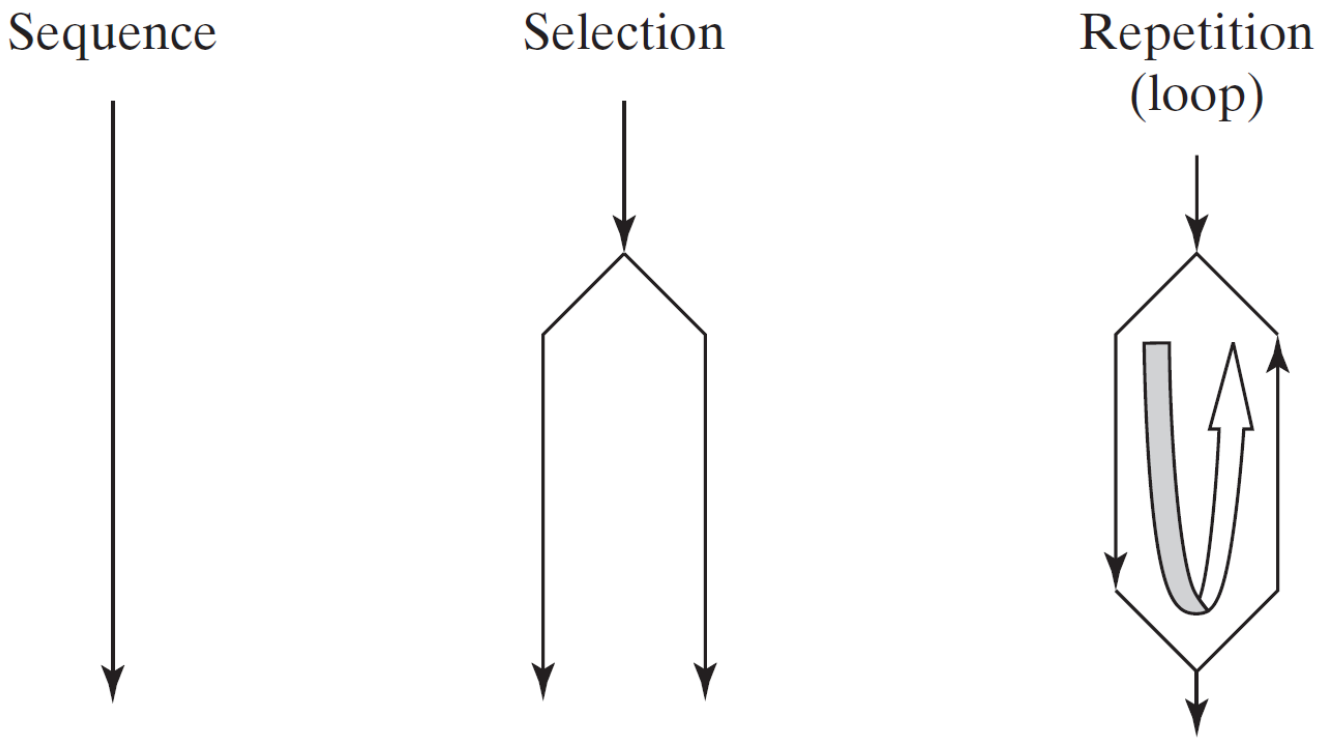
```
1     ...
2     public static void main (String[] args) {
3     Scanner input = new Scanner(System.in);
4     System.out.println("a = ?");
5     double a = input.nextDouble();
6     System.out.println("b = ?");
7     double b = input.nextDouble();
8     System.out.println("c = ?");
9     double c = input.nextDouble();
10
11    double mean = (a + b + c) / 3;
12    double std = Math.sqrt((Math.pow(a - mean, 2) +
13    Math.pow(b - mean, 2) +
14    Math.pow(c - mean, 2)) / 2);
15
16    System.out.println("mean = " + mean);
17    System.out.println("std = " + std);
18    }
19    ...
```

```
1    class Lecture3 {
2
3    "Selections"
4
5    }
6
7    // Keywords
8    if, else, else if, switch, case, default
```

Flow Controls

The basic algorithm (and program) is constituted by the following operations:

- Sequential statements: instructions executed in order.
- Selection: first check if the condition is satisfied, then select the next instruction based on boolean values.
- Repetition: repeat the execution of a block of instructions until the criterion is not satisfied.



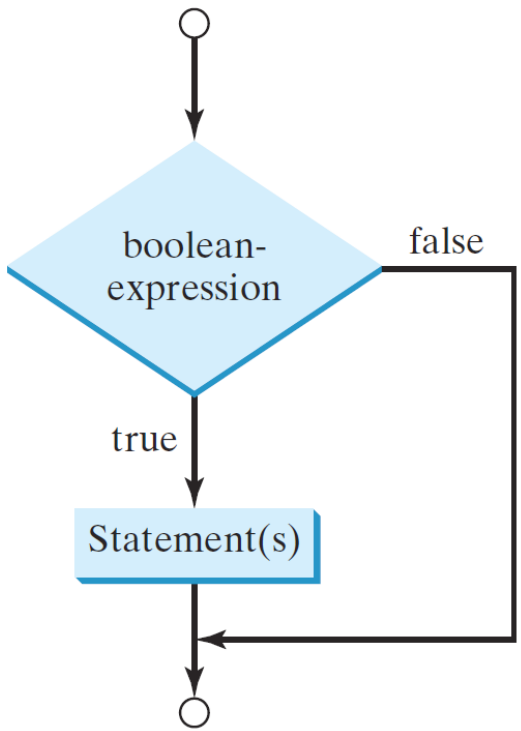
- Note that they are involved with each other in most cases,
- For example, recall how to find the highest score of the math quiz?

Selections

- One-way if statements 
- Two-way if-else statements 
- Nested if statements
- Multiway if-else if-else statements 
- switch-case statements
- Conditional operators

One-Way if Statements

A one-way if statement executes an action if and only if the condition is true.



```
1    ...
2    if (condition) {
3    // selection body
4    }
5    ...
```

- The keyword if is followed by the parenthesized condition.
- The condition should be a boolean expression or a boolean value.
- It the condition is true, then the statements in the selection body will be executed once.
- If not, then the program won't enter the selection body.
- Note that the braces can be omitted if the block contains a single statement.

Example

Write a program which receives a nonnegative number as input for the radius of a circle, and determines the area of the circle.

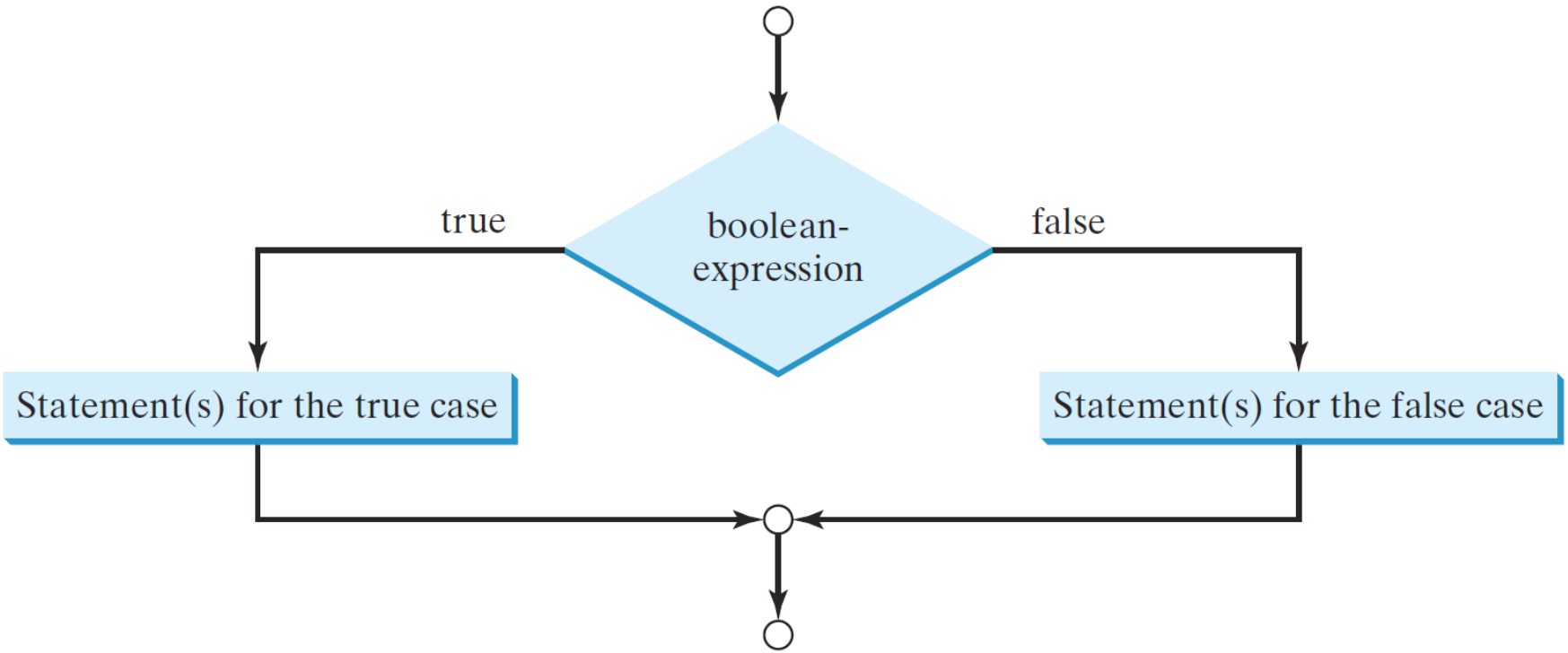
```
1  ...
2  double area;
3  if (r >= 0) {
4  area = r * r * 3.14;
5  System.out.println(area);
6  }
7  ...
```

- However, the world is not well-defined.

Two-Way if-else Statements

A two-way if-else statement decides which statements to execute based on whether the condition is true or false.

```
1  ...
2  if (condition) {
3  // body for the true case
4  } else {
5  // body for the false case
6  }
7  ...
```


true
boolean-
expression
false
Statement(s) for the true case
Statement(s) for the false case

Example

Write a program which receives a number as input for the radius of a circle. If the number is nonnegative, then determine the area of the circle; otherwise, output "Not a circle."

```
1    ...
2    double area;
3    if (r >= 0) {
4    area = r * r * 3.14;
5    System.out.println(area);
6    } else {
7    System.out.println("Not a circle.");
8    }
9    input.close();
10   }
11   ...
```

Nested if Statements

- For example,

```
1     ...
2     if (score >= 90)
3     System.out.println("A");
4     else {
5     if (score >= 80)
6     System.out.println("B");
7     else {
8     if (score >= 70)
9     System.out.println("C");
10    else {
11    if (score >= 60)
12    System.out.println("D");
13    else
14    System.out.println("F");
15    }
16    }
17    }
18    ...
```

Multi-Way if-else

- Let's redo the previous problem.

```
1   ...
2   if (score >= 90)
3   System.out.println("A");
4   else if (score >= 80)
5   System.out.println("B");
6   else if (score >= 70)
7   System.out.println("C");
8   else if (score >= 60)
9   System.out.println("D");
10  else
11  System.out.println("F");
12  ...
```

- An if-elseif-else statement is a preferred format for multiple alternatives, in order to avoid deep indentation and makes the program easy to read.

- The order of conditions may be relevant. (Why?)

```
1   ...
2   if ((score >= 90) && (score <= 100))
3   else if ((score >= 80) && (score < 90))
4   ...
5   else
6   ...
```

- The performance may degrade due to the order of conditions. (Why?)

Common Errors in Selection Statements

- Fix the two bugs for the following code snippet:

```
1   ...
2   double area;
3   if (r > 0);
4   area = r * r * 3.14;
5   System.out.println(area);
6   ...
```

- Terrible code snippet if the curly braces are ignored, for example,

```
1    ...
2    int i = 1, j = 2, k = 3;
3    if (i > j)
4    if (i > k)
5    System.out.println("Max = " + i);
6    else if (j > k)
7    System.out.println("Max = " + j);
8    else
9    System.out.println("Max = " + k);
10   ...
```

Example

Generating random numbers

Write a program which generates 2 random integers and asks the user to answer the math expression.

- For example, the program shows $2 + 5 =$?
- If the user answers 7, then the program reports "Correct." and terminates.
- Otherwise, the program reports "Wrong answer. The correct answer is 7." for this case.
- You may use **Math.random**() for a random value between 0.0 and 1.0, excluding themselves.

```
1   ...
2   public static void main(String[] args) {
3
4   int x = (int) (Math.random() * 10); // integers 0 ~ 9
5   int y = (int) (Math.random() * 10);
6   int answer = x + y;
7
8   System.out.println(x + " + " + y + " =  ?");
9
10  Scanner input = new Scanner(System.in);
11  int z = input.nextInt();
12
13  if (z == answer)
14  System.out.println("Correct.");
15  else
16  System.out.println("Wrong. Answer: " + answer);
17  input.close();
18  }
19  ...
```

- Can you extend this program to include $+ - \times \div$ in the math expressions?

Exercise

Find Maximum

Write a program which determines the maximum value in 3 random integers whose range from 0 to 99.

- How many variables do we need?
- How to compare?
- How to keep the maximum value?

- In this case, a scalar variable is not convenient. (Why?)
- So we need two more elements: arrays and loops.

switch-case Statements

A switch-case statements executes statements based on the value of a variable or an expression.

```
1   ...
2   switch (target) {
3   case v1:
4   // statements
5   break;
6   case v2:
7   .
8   .
9   case vk:
10  // statements
11  break;
12  default:
13  // statements
14  }
15  ...
```

- A switch-case statement is more convenient than an if statement for multiple discrete conditions.
- The variable *target*, always enclosed in parentheses, must yield a value of char, byte, short, int, or **String** type.
- The value $v_1, \ldots,$ and v_k must have the same data type as the variable *target*.
- In each case, a break statement is a must.[5]
  - break is used to break a construct!
- The default case, which is optional, can be used to perform actions when none of the specified cases matches *target*.
  - Counterpart to else statements.

[5] If not, there will be a fall-through behavior.

Example

Write a program which picks a weekday randomly.

- How many discrete cases?

```
1   ...
2   int selectedWeekday = (int) (Math.random() * 7);
3   switch (selectedWeekday) {
4   case 0:
5   System.out.println("Monday");
6   break;
7   case 1:
8   System.out.println("Tuesday");
9   break;
10  case 2:
11  System.out.println("Wednesday");
12  break;
13  case 3:
14  System.out.println("Thursday");
15  break;
16  case 4:
```

```
17      System.out.println("Friday");
18      break;
19      case 5:
20      System.out.println("Saturday");
21      break;
22      case 6:
23      System.out.println("Sunday");
24      }
25      ...
```

- Can we do this by if statements?

Conditional Operators

A conditional expression evaluates an expression based on the specified condition.

```
1    ...
2    booleanExpr ? exprA : exprB;
3    ...
```

- This is the only ternary operator in Java.
- If the boolean expression is evaluated true, then return expr A; otherwise, expr B.

- For example,

```
1  ...
2  if (num1 > num2)
3  max = num1;
4  else
5  max = num2;
6  ...
```

- Alternatively, one can use a conditional expression like this:

```
1  ...
2  max = (num1 > num2) ? num1 : num2;
3  ...
```

Problem Set[6]

Exercise 3.1 (Roots of 2^{nd}-order polynomials)

Write a program which solves $y = ax^2 + bx + c$ for all $x, a, b, c \in \mathbb{R}$ using the following formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Note that a complex root and its conjugate are allowed in general.

Exercise 3.2 (Game: scissor, rock, paper)

Write a program that plays the popular scissor-rockpaper game. The program randomly generates a number 0, 1, or 2 representing scissor, rock, and paper. The program prompts the user to enter a number 0, 1, or 2 and displays a message indicating whether the user or the computer wins, loses, or draws.

```
scissor (0), rock (1), paper (2): 1 ↵Enter
The computer is scissor. You are rock. You won
```

Exercise 3.3 (Sort three integers)

Write a program that sorts three integers. The integers are entered from the input dialogs and stored in 3 variables.

- For example, the input is $a = 1, b = 5, c = 2$. Then the program display the 3 numbers in ascending order, that is, $1 < 2 < 5$.

Exercise 3.4 (Compute the perimeter of a triangle)

Write a program that reads three edges for a triangle and computes the perimeter if the input is valid. Otherwise, display that the input is invalid. The input is valid if the sum of every pair of two edges is greater than the remaining edge.

Exercise 3.5 (Geometry: points in triangle?)

Suppose a right triangle is placed in a plane as shown below. The right-angle point is placed at $(0, 0)$, and the other two points are placed at $(200, 0)$, and $(0, 100)$. Write a program that prompts the user to enter a point with x- and y-coordinates and determines whether the point is inside the triangle.

Exercise 3.6 (Geometry: point in a circle?)

Write a program that prompts the user to enter a point (x, y) and checks whether the point is within the circle centered at $(0, 0)$ with radius 10.

[6] See Programming Exercises in YDL, p. 121-131.

```
1    class Lecture4 {
2
3    "Loops"
4
5    }
6
7    // keywords:
8    while, do, for, break, continue
```

Loops

A loop can be used to make a program execute statements repeatedly without having to code the same statements.

- For example, a program outputs "Hello, Java." for 100 times.

```
1    ...
2    System.out.println("Hello, Java.");
3    System.out.println("Hello, Java.");
4    .
5    . // copy and paste for 100 times
6    .
7    System.out.println("Hello, Java.");
8    ...
```

```
1     ...
2     int cnt = 0;
3     while (cnt < 100) {
4     System.out.println("Hello, Java.");
5     cnt++;
6     }
7     ...
```

- This is a simple example to show the power of loops.
- In practice, any routine which repeats couples of times[7] can be done by folding them into a loop.

[7] I'd like to call them "patterns."

- Loops provide significant computational power.
- Loops bring an efficient way of programming.
- Loops could consume a lot of time.[8]

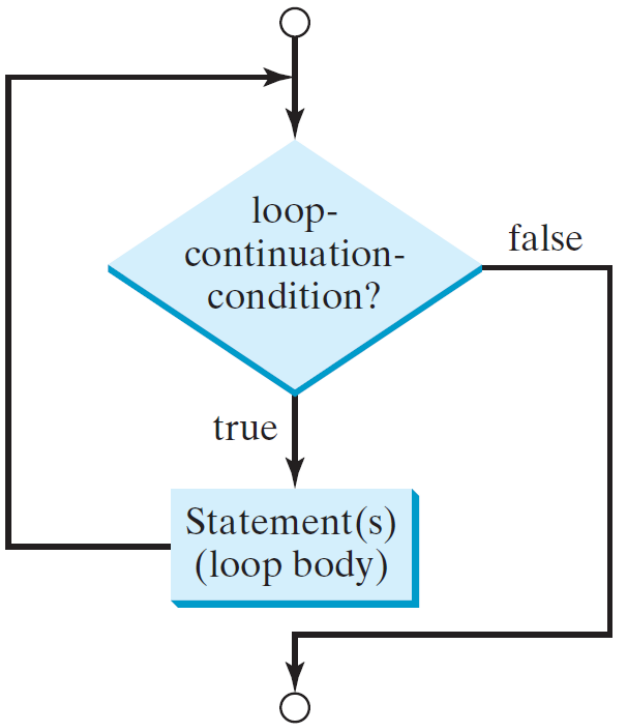
[8] We will visit the analysis of algorithms in the end of this lecture.

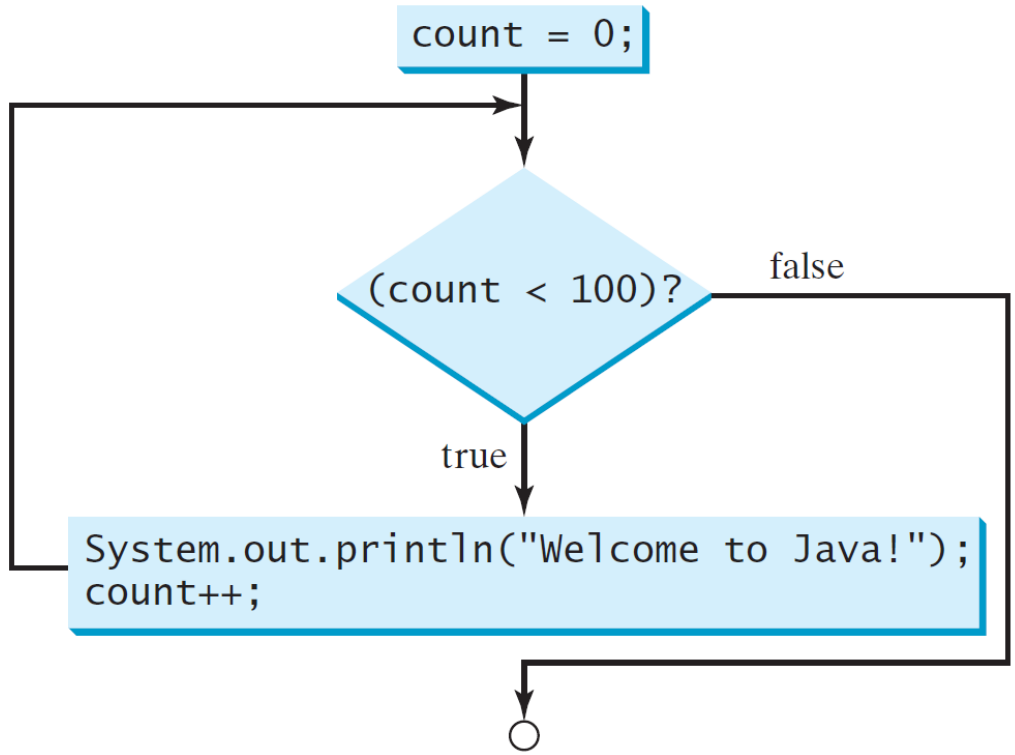
while Loops

A while loop executes statements repeatedly while the condition is true.

```
1  ...
2  while (condition) {
3  // loop body
4  }
5  ...
```

- The condition is a Boolean expression which controls the execution of the body.
- It is evaluated each time to determine if the loop body is executed.
- If true, the loop body is executed.
- Otherwise, the entire loop terminates.


loop-
continuation-
condition?
false
true
Statement(s)
(loop body)


count = 0;
(count < 100)?
false
true
System.out.println("Welcome to Java!");
count++;

Example

Write a program which sums up all integers from 1 to 100.

- In math, the question can be:

$$\text{sum} = 1 + 2 + \cdots + 100.$$

- But the form is not doable in the machine.
- Try computational thinking.[9]

[9] We are familiar with mathematical thinking since we learn math before we learn how to code.

- Normally, the computer executes the instructions sequentially.[10]
- So one needs to decompose the math equation into several steps, like:

```
1  ...
2  int sum = 0;
3  sum = sum + 1;
4  sum = sum + 2;
5  .
6  .
7  .
8  sum = sum + 100;
9  ...
```

- Cons: not efficient, not general (what if sum up to 10^{10}?)

[10] If we are talking about the parallel computing, then it is a different story.

- Using a while loop, the program looks like this:

```
1  ...
2  int sum = 0;
3  int i = 1;
4  while (i <= 100) {
5  sum = sum + i;
6  ++i;
7  }
8  ...
```

- Make sure that the condition eventually becomes false so that the loop will terminate.
- It is really easy to make an infinite loop.

```
1  ...
2  while (true);
3  ...
```

- Besides, replacing 100 by n determined by the user makes this program more general.

```
1    ...
2    Scanner input = new Scanner(System.in);
3    int n = input.nextInt();
4    int sum = 0;
5    int i = 1;
6    while (i <= n) {
7    sum = sum + i;
8    i = i + 1;
9    }
10   ...
```

- In practice, the number of loop steps is unknown until the input data is given.

Example

Write a program which sums two random integers and lets the user repeatedly enter a new answer until it is correct.

```
1   ...
2   Scanner input = new Scanner(System.in);
3   int x = (int) (Math.random() * 10);
4   int y = (int) (Math.random() * 10);
5   int ans = x + y;
6
7   System.out.println(x + " + " + y + " = ? ");
8   int z = input.nextInt();
9
10  while (z != ans) {
11  System.out.println("Try again? ");
12  z = input.nextInt();
13  }
14  System.out.println("Correct.");
15  input.close();
16  ...
```

Loop Design Strategy

Writing a correct loop is not an easy task for novice programmers. Consider 3 steps when writing a loop:

- Find the pattern: identify the statements that need to be repeated.
- Wrap: put these statements in a loop.
- Set the continuation condition: translate the criteria from the real world problem into computational conditions.[11]

[11] Not unique.

Sentinel-Controlled Loop

Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.

- This special input value, known as a sentinel value, signifies the end of the loop.
- For example, the operating systems and the GUI apps.

Example: Cashier Problem

Write a program which sums positive integers from the input except for -1 to exit, and displays the sum.

```
1   ...
2   Scanner in = new Scanner(System.in);
3   System.out.println("Enter a positive integer (-1 to exit): "
        );
4   int x = in.nextInt();
5   int sum = 0;
6   while (x != -1) {
7   if (x > 0) sum += x;
8   System.out.println("Enter a positive integer (-1 to exit): "
        );
9   x = in.nextInt();
10  }
11  System.out.println("Sum = " + sum);
12  in.close();
13  ...
```

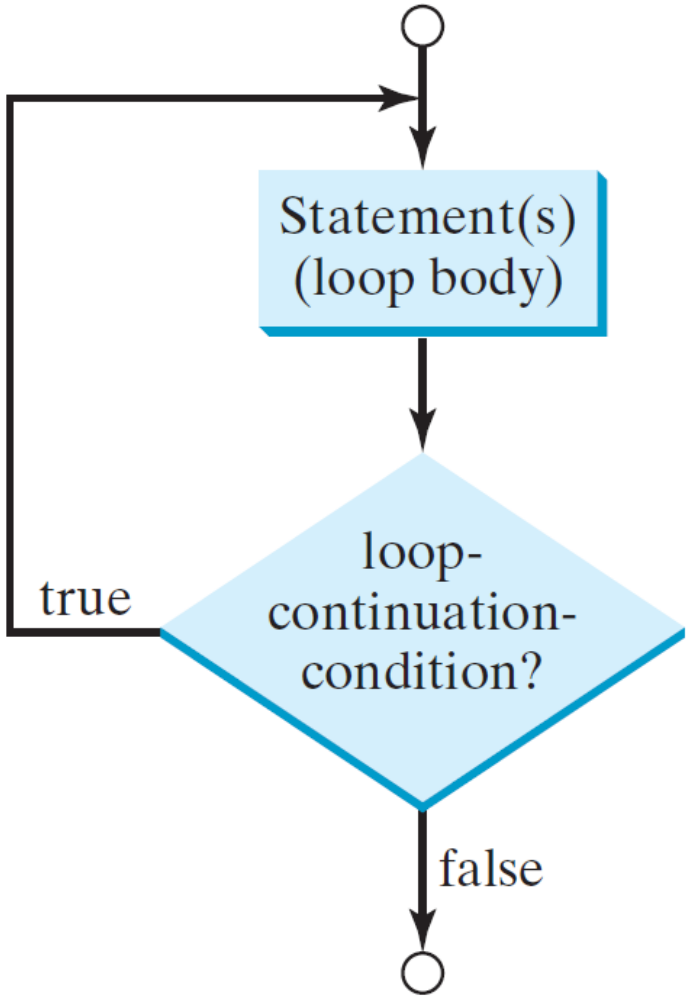
- Line 8 and 9 are the recurrence of Line 3 and 4?!

do-while Loops

A do-while loop is the same as a while loop except that it does execute the loop body first and then checks the loop continuation condition.

```
1  ...
2  do {
3  // loop body
4  } while (condition); // Do not miss the semicolon!
5  ...
```

- Note that there is a semicolon at the end the do-while loop.
- The do-while loops are also called posttest loop, in contrast to while loops, which are pretest loops.


Statement(s)
(loop body)
loop-
continuation-
condition?
true
false

Example (Revisted)

Write a program which sums positive integers from the input except for -1 to exit, and displays the sum.

```
1   ...
2   Scanner in = new Scanner(System.in);
3   int sum = 0;
4   int x = 0;
5
6   do {
7   System.out.println("Enter a positive integer (-1 to exit): "
        );
8   x = in.nextInt();
9   if (x >= 0) sum += x;
10  } while (x != -1);
11  System.out.println("Sum = " + sum);
12  in.close();
13  ...
```

Exercise

Write a program which allows the user to enter positive integers except for -1 to exit, and displays the maximum.

```
Please enter a real number (-1 to exit):
5
Max = 5.0
Please enter a real number (-1 to exit):
2
Max = 5.0
Please enter a real number (-1 to exit):
7
Max = 7.0
Please enter a real number (-1 to exit):
-1
```

```
1     ...
2     Scanner in = new Scanner(System.in);
3     int max = 0, x;
4     do{
5     System.out.println("Enter a positive integer (-1 to exit): "
          );
6     x = in.nextInt();
7     if(max < x) {
8     max = x;
9     }
10    System.out.println("Max = " + max);
11    } while (x != -1);
12    in.close();
13    ...
```

for Loops

A for loop generally uses a variable to control how many times the loop body is executed and when the loop terminates.

```
1  ...
2  for (init; condition; increment) {
3  // loop body
4  }
5  ...
```

- *init*: declare and initialize a variable
- *condition*: a continuation criterion
- *increment*: how the variable changes after each iteration
- Note that the three terms are separated by semicolons.

Example

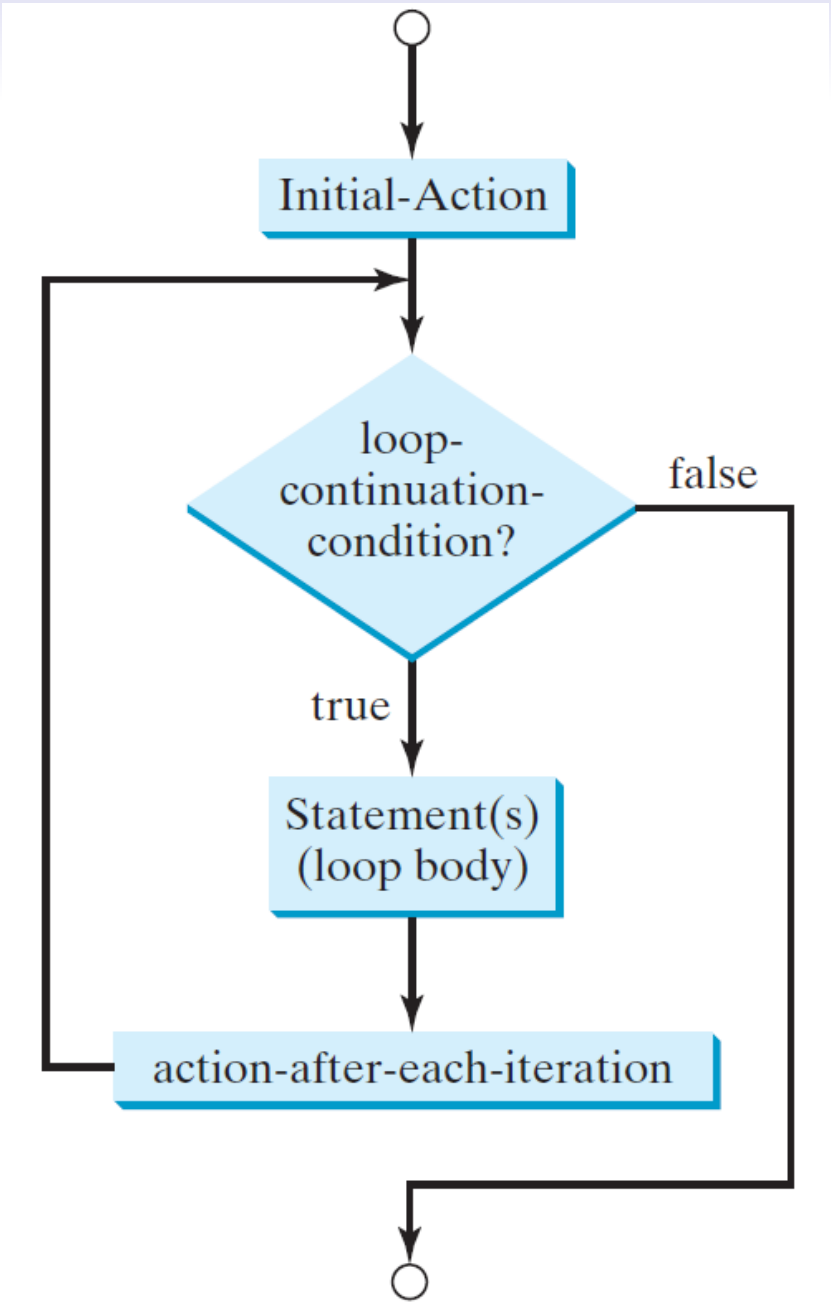
Sum from 1 to 100

Write a program which sums from 1 to 100.

```
1  ...
2  int sum = 0;
3  for (int i = 1; i <= 100; ++i)
4  sum = sum + i;
5  ...
```

- Compared to the while version,

```
1  ...
2  int sum = 0;
3  int i = 1;
4  while (i <= 100) {
5  sum = sum + i;
6  ++i;
7  }
8  ...
```


Initial-Action
loop-continuation-condition?
false
true
Statement(s) (loop body)
action-after-each-iteration

Example: Selection Resided in Loop

Display all even numbers

Write a program which displays all even numbers smaller than 100.

- An even number is an integer of the form $x = 2k$, where k is an integer.

- You may use modular operator (%).

```
...
for (int i = 1; i <= 100; i++) {
if (i % 2 == 0)
System.out.println(i);
}
...
```

- You may consider this alternative:

```
...
for (int i = 2; i <= 100; i += 2) {
System.out.println(i);
}
...
```

- How about odd numbers?

Jump Statements

The keywords, break and continue, are often used in loop structures to provide additional controls.

- break: the loop is terminated right after a break statement is executed.
- continue: the loop skips this iteration right after a continue statement is executed.
- In practice, jump statements are placed within selection structures in loops. (Why?)

Example

isPrime problem

Write a program which determines if the input integer is a prime number.

- Recall that a prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.
- Let x be any natural number.
- The most naive approach is to divide x by all natural numbers smaller than x.
- A better approach is to divide x by all natural numbers smaller than $\sqrt{x}$. (Why?)

```
1     ...
2     Scanner input = new Scanner(System.in);
3     System.out.println("Please enter an integer: ");
4     int x = input.nextInt();
5     boolean isPrime = true;
6     in.close();
7
8     for (int i = 2; i <= Math.sqrt(x); i++) {
9     if (x % i == 0) {
10    isPrime = false;
11    break;
12    }
13    }
14
15    if (isPrime) {
16    System.out.println("Prime");
17    } else {
18    System.out.println("Composite");
19    }
20    ...
```

Exercise (Revisited)

- Redo the cashier problem by using an infinite loop with a jump statement.

```
1   ...
2   while (true) {
3   System.out.println("Enter a positive integer (-1 to exit): "
        );
4   x = input.nextInt();
5   if (x >= 0) {
6   sum += x;
7   } else if (x == -1) {
8   System.out.println("Sum = " + sum);
9   break;
10  } else {
11  System.out.println("Try again.");
12  }
13  }
14  in.close();
15  ...
```

Equivalence: while and for Loops

Compounding problem

Consider that one saves $10,000$ NTD in a bank. Write a program which determines the number of years n such that the compounding amount of saving exceeds $15,000$ NTD as r is specified.

- We may need the following variables:
  - *principal*: the starting amount
  - *currAmount*: the current balance of saving
  - *goalAmount*: the amount we set
  - r: interest rate, herein, assuming constant
- Recall that the compounding formula.
- Stopping criterion? Continuation criterion?

```
1     ...
2     public static void main(String[] args) {
3     Scanner inpunt = new Scanner(System.in);
4
5     System.out.println("Interest rate (%) = ?");
6     double r = input.nextDouble();
7     System.out.println("Principal = ?");
8     int principal = input.nextInt();
9     System.out.println("Goal = ?");
10    int goalAmount = input.nextInt();
11    int currAmount = principal;
12    in.close();
13
14    int years = 0;
15    while (currAmount < goalAmount) { // continuation criterion
16    currAmount *= (1 + r / 100);
17    years++;
18    }
19
20    System.out.println("Years = " + years);
21    System.out.println("Amount = " + currAmount);
22    }
23    ...
```

```
1    ...
2    int years; // should be declared; scope issue
3    for (years = 1; true; n++) {
4    currAmount *= (1 + r / 100);
5    if (currAmount >= goalAmount) // stopping criterion
6    break;
7    }
8    ...
```

- A for loop can be an infinite loop by setting true or simply leaving empty in the condition statement.
- An infinite for loop with an if-break statement is equivalent to a normal while loop.

Equivalence: while and for Loops (Concluded)

You can use a for loop, a while loop, or a do-while loop, whichever is convenient.

- In general, a for loop may be used if the number of repetitions is known in advance.
- If not, a while loop is preferred.