## Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical negation</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>logical conjunction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
<td>logical exclusion</td>
</tr>
</tbody>
</table>

\(^1\)See Table 3-2 in YDL, p. 102.
• Let $X$ and $Y$ be two Boolean variables.
• Then the truth table for logical operators is as follows:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$\neg X$</th>
<th>$X &amp; &amp; Y$</th>
<th>$X \parallel Y$</th>
<th>$X \land Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

• Note that the instructions of computers, such as arithmetic operations, are implemented by logic gates.\(^2\)

\(^2\)See any textbook for digital circuit design.
“Logic is the anatomy of thought.”
– John Locke (1632–1704)

“This sentence is false.”
– anonymous

“I know that I know nothing.”
– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)
### Arithmetic Compound Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>Increment</td>
</tr>
<tr>
<td>+=</td>
<td>Addition assignment</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assignment</td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assignment</td>
</tr>
<tr>
<td>/=</td>
<td>Division assignment</td>
</tr>
<tr>
<td>%=</td>
<td>Modulus assignment</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
</tr>
</tbody>
</table>

- Note that these shorthand operators are not available in languages such as Matlab, R, and Python.
Example

```java
int x = 1;
System.out.println(x); // output 1
x = x + 1;
System.out.println(x); // output 2
x += 2;
System.out.println(x); // output 4
x++; // equivalent to x += 1 and x = x + 1
System.out.println(x); // output 5
...
```
• The compound assignment operators are also useful for char values.³

• For example,

```java
...  
  char s = 'a';  
  System.out.println(s); // output a  
  s += 1;  
  System.out.println(s); // output b  
  s++;  
  System.out.println(s); // output c  
...  
```

³Contribution 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,

```java
... int x = 1;
int y = ++x;
System.out.println(y); // output 2; aka preincrement
System.out.println(x); // output 2

int w = 1;
int z = w++;
System.out.println(z); // output 1; aka postincrement
System.out.println(w); // output 2

...```

• We will use these notations very often.
Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td><code>var++</code> and <code>var--</code> (Postfix)</td>
</tr>
<tr>
<td>3</td>
<td><code>+</code>, <code>-</code> (Unary plus and minus), <code>++var</code> and <code>--var</code> (Prefix) (type) (Casting)</td>
</tr>
<tr>
<td>2</td>
<td><code>!</code> (Not)</td>
</tr>
<tr>
<td>1</td>
<td><code>*</code>, <code>/</code>, <code>%</code> (Multiplication, division, and remainder)</td>
</tr>
<tr>
<td>1</td>
<td><code>+</code>, <code>-</code> (Binary addition and subtraction)</td>
</tr>
<tr>
<td>1</td>
<td><code>&lt;</code>, <code>&lt;=</code>, <code>&gt;</code>, <code>&gt;=</code> (Comparison)</td>
</tr>
<tr>
<td>1</td>
<td><code>==</code>, <code>!=</code> (Equality)</td>
</tr>
<tr>
<td>1</td>
<td><code>^</code> (Exclusive OR)</td>
</tr>
<tr>
<td>1</td>
<td><code>&amp;&amp;</code> (AND)</td>
</tr>
<tr>
<td>1</td>
<td>`</td>
</tr>
<tr>
<td>0</td>
<td><code>=</code>, <code>+=</code>, <code>-=</code>, <code>*=``, </code>/=<code>, </code>%=` (Assignment operator)</td>
</tr>
</tbody>
</table>

See Table 3-10 in YDL, p. 116.
Using Parentheses

- Parentheses are used in expressions to change the natural order of precedence among the operators.
- One always evaluates the expression inside of parentheses first.
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 some input methods, say the input received from the keyboard or the files.
- 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.

```java
import java.util.Scanner;
...
Scanner input = new Scanner(System.in);
System.out.println("Enter r?");
// input
int r = input.nextInt();
// algorithm
double area = r * r * 3.14;
// output
System.out.println(area);
input.close();
...
• In the listing, Line 3 is to create a **Scanner** object by the `new` operator, as an agent between the keyboard and your program.
• Note that all objects are resided in the **heap** of the memory.
• To control this object, its **memory address** is then assigned to the variable `input` which is a variable in the **stack** of memory.
• So the variable `input` is a **reference**.
• We will discuss the objects and reference variables later.
## Methods Provided by Scanner Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nextByte()</code></td>
<td>reads an integer of the <code>byte</code> type.</td>
</tr>
<tr>
<td><code>nextShort()</code></td>
<td>reads an integer of the <code>short</code> type.</td>
</tr>
<tr>
<td><code>nextInt()</code></td>
<td>reads an integer of the <code>int</code> type.</td>
</tr>
<tr>
<td><code>nextLong()</code></td>
<td>reads an integer of the <code>long</code> type.</td>
</tr>
<tr>
<td><code>nextFloat()</code></td>
<td>reads a number of the <code>float</code> type.</td>
</tr>
<tr>
<td><code>nextDouble()</code></td>
<td>reads a number of the <code>double</code> type.</td>
</tr>
<tr>
<td><code>next()</code></td>
<td>reads a string that ends before a whitespace character.</td>
</tr>
<tr>
<td><code>nextLine()</code></td>
<td>reads a line of text (i.e., a string ending with the <code>Enter</code> key pressed).</td>
</tr>
</tbody>
</table>

---

\(^5\)See Table 2-1 in YDL, p. 38.
Example: Mean and Standard Deviation

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

• The mean of 3 numbers is given by \( \bar{x} = \left( \sum_{i=1}^{3} x_i \right) / 3 \).
• Also, the resulting standard deviation is given by

\[
S = \sqrt{\frac{\sum_{i=1}^{3} (x_i - \bar{x})^2}{3}}.
\]

• 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](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html).
Scanner input = new Scanner(System.in);
System.out.println("a = ?");
double a = input.nextDouble();
System.out.println("b = ?");
double b = input.nextDouble();
System.out.println("c = ?");
double c = input.nextDouble();

double mean = (a + b + c) / 3;
double std = Math.sqrt((Math.pow(a - mean, 2) +
 Math.pow(b - mean, 2) +
 Math.pow(c - mean, 2)) / 3);

System.out.println("mean = " + mean);
System.out.println("std = " + std);

...
class Lecture3 {
    "Selections"
}

// Keywords
if, else, else if, switch, case, default
Flow Controls

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

- **Sequential statements**: execute instructions in order.
- **Selection**: first check if the predetermined condition is satisfied, then execute the corresponding instruction.
- **Repetition**: repeat the execution of some instructions until the criterion fails.
Note that they are involved with each other generally.

For example, recall how to find the maximum in the input list?
Selections

- One-way `if` statements
- Two-way `if-else` statements
- Nested `if` statements
- Multiway `if-else if-else` statements
- `switch-case` statements
- Conditional operators
A one-way if statement executes an action if and only if the condition is true.
• The keyword `if` is followed by the parenthesized condition.
• The condition should be a boolean expression or a boolean value.
• If 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 and skip the whole selection body.
• Note that the braces can be omitted if the block contains only 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.

```java
... 
double area;
if (r > 0) {
    area = r * r * 3.14;
    System.out.println(area);
}
...
```

- 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.

```java
... if (condition) {
    // body for the true case
} else {
    // body for the false case
}
...```

Zheng-Liang Lu Java Programming 95 / 132
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.”

```java
... double area;
if (r > 0) {
    area = r * r * 3.14;
    System.out.println(area);
} else {
    System.out.println("Not a circle.");
}
input.close();
...
Nested if Statements

- For example,

```java
... if (score >= 90) 
    System.out.println("A");
else {
    if (score >= 80)
        System.out.println("B");
    else {
        if (score >= 70)
            System.out.println("C");
        else {
            if (score >= 60)
                System.out.println("D");
            else
                System.out.println("F");
        }
    }
...}
```
Multi-Way if-else

- Let's redo the previous problem.

```java
... if (score >= 90)
    System.out.println("A");
else if (score >= 80)
    System.out.println("B");
else if (score >= 70)
    System.out.println("C");
else if (score >= 60)
    System.out.println("D");
else
    System.out.println("F");
...
```

- An if-elseif-else statement is a preferred format for multiple alternatives, in order to avoid deep indentation and make the program easy to read.
• The order of conditions may be relevant. (Why?)

```java
...  
  if ((score >= 90) && (score <= 100))
  else if ((score >= 80) && (score < 90))
  ...  
  else
  ...
```

• The performance may degrade due to the order of conditions. (Why?)
Common Errors

```java
... double area;
if (r > 0);
    area = r * r * 3.14;
System.out.println(area);
...
```
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.
```java
int x = (int) (Math.random() * 10); // integers 0 ~ 9
int y = (int) (Math.random() * 10);
int answer = x + y;

System.out.println(x + " + " + y + " = ?");

Scanner input = new Scanner(System.in);
int z = input.nextInt();

if (z == answer)
    System.out.println("Correct.");
else
    System.out.println("Wrong. Answer: " + answer);
input.close();
```

- Can you extend this program for all arithmetic expressions (i.e., + − × ÷)?
Exercise

Find Max

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 arrays and loops.
A switch-case structure takes actions depending on the target variable.

```java
switch (target) {
    case v1:
        // statements
        break;
    case v2:
        .
        .
        case vk:
            // statements
            break;
    default:
        // statements
}
```
• 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 $v_1, \ldots, v_k$ must have the same data type as the variable target.

• In each case, a break statement is a must.  
  • 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.

---

6 If not, there will be a fall-through behavior.
Example

...  
// RED: 0
// YELLOW: 1
// GREEN: 2
int trafficLight = (int) (Math.random() * 3);
switch (trafficLight) {
    case 0:
        System.out.println("Stop!!!");
        break;
    case 1:
        System.out.println("Slow down!!");
        break;
    case 2:
        System.out.println("Go!");
}
...
Conditional Operators

A conditional expression evaluates an expression based on the specified condition and returns a value accordingly.

1
2
3

someVar = booleanExpr ? exprA : exprB;

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

```java
... if (num1 > num2) 
    max = num1;
else 
    max = num2;
...
```

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

```java
... max = (num1 > num2) ? num1 : num2;
...```
class Lecture4 {
    "Loops"
}

// keywords:
while, do, for, break, continue
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.

```java
... System.out.println("Hello, Java.");
System.out.println("Hello, Java.");
.
// copy and paste for 100 times
.
System.out.println("Hello, Java.");
...
```
This is a simple example to show the power of loops.

In practice, any routine which repeats couples of times\textsuperscript{7} can be done by folding them into a loop.

\textsuperscript{7}I prefer to call them “patterns.”
Loops provide substantial 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.

• The condition should be a boolean expression which determines whether or not the execution of the body occurs.
• If true, the loop body is executed and check the condition again.
• Otherwise, the entire loop terminates.
count = 0;

(count < 100)?

true

System.out.println("Welcome to Java!");
count++;

false

loop-continuation-condition?

false

true

Statement(s) (loop body)
Example

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

- In math, the question can be written as:
  \[ \text{sum} = 1 + 2 + \cdots + 100. \]

- But this form is not doable in the machine.\(^9\)

\(^9\)We need to develop computational thinking. Read
http://rsta.royalsocietypublishing.org/content/366/1881/3717.full
or
http://blog.orangeapple.tw/posts/what-is-computational-thinking/.
• Normally, the machine executes the instructions **sequentially**.
• So one needs to decompose the math equation into several steps, like:

```java
...  
    int sum = 0;
    sum = sum + 1;
    sum = sum + 2;
    ...
    sum = sum + 100;
...  
```

• It is obvious that many similar statements can be found.
Using a **while** loop, the program can be rearranged as follows:

```java
int sum = 0;
int i = 1;
while (i <= 100) {
    sum = sum + i;
    ++i;
}
```

- You should guarantee that the loop will terminate as expected.
- In practice, the number of loop steps (iterations) is **unknown** until the input data is given.
Malfunctioned Loops

- It is really easy to make an infinite loop.

```java
... 
while (true); 
... 
```

- The common errors of the loops are:
  - never start
  - never stop
  - not complete
  - exceed the expected number of iterations
Example

Write a program which asks the sum of two random integers and lets the user repeatedly enter a new answer until correct.

```java
... Scanner input = new Scanner(System.in);
int x = (int) (Math.random() * 10);
int y = (int) (Math.random() * 10);
int ans = x + y;

System.out.println(x + " + " + y + " = ? ");
int z = input.nextInt();

while (z != ans) {
    System.out.println("Try again? ");
    z = input.nextInt();
}
System.out.println("Correct.");
input.close();
...
```
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 by loops**: put these statements in the loop.
  - **Set the continuation condition**: translate the criterion from the real world problem into computational conditions.\(^\text{10}\)

\(^{10}\)Not unique.
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 over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```java
Scanner input = new Scanner(System.in);
int total = 0;
System.out.println("Enter a price?");
price = input.nextInt();
while (price > 0) {
    total += price;
    System.out.println("Enter a price?");
    price = input.nextInt();
}
System.out.println("Total = " + total);
input.close();
```

- Line 8 and 9 are the recurrence of Line 4 and 5?!
do-while Loops

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

```java
/*...*/
do {
    // loop body
} while (condition); // Do not miss the semicolon!
/*...*/
```

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

1. **Statement(s)** (loop body)
2. **loop-continuation-condition?**
   - **true**:
     - (Loops back to the beginning)
   - **false**:
     - (Ends the loop)
Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```java
... Scanner input = new Scanner(System.in);
    int total = 0;
    int price = 0;
    do {
        total += price;
        System.out.println("Enter a price?");
        price = input.nextInt();
    } while (price > 0);
    System.out.println("Total = " + total);
    input.close();
...
A for loop generally uses a variable to control how many times the loop body is executed and when the loop terminates.

```java
... for (init_action; condition; increment) {
    // loop body
}
...```

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

Sum from 1 to 100

Write a program which sums from 1 to 100.

```java
... int sum = 0;
for (int i = 1; i <= 100; ++i)
    sum = sum + i;
...
```

• Compared to the while version,

```java
... int sum = 0;
int i = 1;
while (i <= 100) {
    sum = sum + i;
    ++i;
}
... 
```
Initial-Action

loop-continuation-condition?

true

Statement(s) (loop body)

false

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 the modular operator (%).

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

• Also consider this alternative:

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

• How about odd numbers?