### 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 and R.
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.\textsuperscript{1}

• 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
...
```

\textsuperscript{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,

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

---

²See 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>

---

3See 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.
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
One-Way if Statements

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.
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
}
... 
```
true

boolean-expression

false

Statement(s) for the true case

Statement(s) for the false case
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,

```
...  
  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.\(^4\)

\(^4\)You may see PRNG in https://en.wikipedia.org/wiki/Pseudorandom_number_generator
// (1) generate random integers
int x = (int) (Math.random() * 10);
int y = (int) (Math.random() * 10);
int answer = x + y;

// (2) display the math expression
System.out.println(x + " + " + y + " = ?");

// (3) user input
Scanner input = new Scanner(System.in);
int z = input.nextInt();

// (4) judgement
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., + − × ÷)?
“Exploring the unknown requires tolerating uncertainty.”

– Brian Greene

“I can live with doubt, and uncertainty, and not knowing. I think it is much more interesting to live not knowing than have answers which might be wrong.”

– Richard Feynman
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
}
...```

Zheng-Liang Lu

Java Programming
• 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.\(^5\)
  • 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

...  
  // 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.

```java
... 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
Loops

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

- For example, output “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 toy 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 these routines “patterns.”