ICS 102 Lab03 (Term 122) –More on Arithmetic Expressions

**Objectives:**

* Compound assignment operators
* Increment and Decrement Operators
* Automatic Type conversions
* Type casting
* The Math class
* Output Formatting using format specifiers

1. **Compound Assignment Operators**

In programs, it is quite common to get the current value of a variable, modify it, and then assign the resulting value back to the original variable.

Example:

**x = x + 10;**

This statement specifies that the value 10 is to be added to the current contents of the variable ***x*** and the resulting value becomes the new contents of ***x***. Effectively, this statement increases the value of the variable ***x*** by 10. Java provides a shorthand way to write this statement using the compound assignment operator: **+=** .Instead of writing the statement:

**x = x + 10;**

We can write:

**x += 10;**

In general, in Java an assignment of the form:

**variableX = variableX operator Expression ;**

Can be written in the form:

**variableX operator= Expression;**

For arithmetic operators, the compound assignment operators are \*= , /= , %= , +=, and -=. These operators have the same priority and associativity as the assignment operator =

Note: Sometimes it is not possible to make the previous two assignment forms equivalent. Example: **int x = 5, y = 2;  
 x = x \* 5 + y;  
 System.out.println(x); // output: 27  
 x = 5;  
 x \*= 5 + y;  
 System.out.println(x); // output: 35**

However the forms:

**variableX = variableX operator (Expression);**

**variableX operator= (Expression);**

are always equivalent.

1. **Increment and Decrement operators**

Each of the post-increment and pre-increment operators increment the value of a variable by one:

|  |  |
| --- | --- |
| variable++ | post-increment operator |
| ++variable | pre-increment operator |

Each of the post-decrement and pre-decrement operators decrement the value of a variable by one:

|  |  |
| --- | --- |
| variable-- | post-decrement operator |
| --variable | pre-decrement operator |

The increment operators have similar effect if they appear alone in an expression. Example:

**int x = 3;**

**x++; // x is incremented to 4**

**x = 3;**

**++x; // x is incremented to 4**

If these operators appear in an assignment statement or in an expression in which they are combined with other operands and operators, then for the pre-increment operator, the variable is first incremented and then this incremented value is used to evaluate the expression. For post-increment operator, the expression is first evaluated with the current value of the variable, then the variable is incremented. The decrement operators behave in a similar way, except that they decrement the variable:

Examples:

|  |  |
| --- | --- |
| Post-Increment | Pre-Increment |
| **int n = 8;**  **int result = n++;**  System.out.println("n = " + n + ", result = " + result);  Output: n = 9, result = 8 | **int n = 8;**  **int result = ++n;**  System.out.println("n = " + n + ", result = " + result);  Output: n = 9, result = 9 |
| double x = 2.0, y; y = x++ \* 5.0; System.out.println("x = " + x + ", y = " + y);  Output: x = 3.0 , y = 10.0 | double x = 2.0, y; y = ++x \* 5.0; System.out.println("x = " + x + ", y = " + y);  Output: x = 3.0 , y = 15.0 |

|  |  |
| --- | --- |
| Post-decrement | Pre-decrement |
| **int n = 8;**  **int result = n--;**  System.out.println("n = " + n + ", result = " + result);  Output: n = 7, result = 8 | **int n = 8;**  **int result = --n;**  System.out.println("n = " + n + ", result = " + result);  Output: n = 7, result = 7 |
| double x = 4.0, y; y = x-- \* 5.0; System.out.println("x = " + x + ", y = " + y);  Output: x = 3.0 , y = 20.0 | double x = 4.0, y; y = --x \* 5.0; System.out.println("x = " + x + ", y = " + y);  Output: x = 3.0 , y = 15.0 |

1. **Integer division**

The result of an expression in which all the operands are integers is an integer.

|  |  |
| --- | --- |
| Expression | Result |
| 10 / 2 | 5 |
| 7 / 3 | 2 |
| 4 / 5 | 0 |
| 6 + 2 \* 3 | 12 |

You have to be careful when using integer division as sometimes it may introduce logic errors in your program. Example:

Consider the following program fragment that computes the average of two integers:

**int num1 = 2, num2 = 5, average;**

**average = (num1 + num2 ) / 2;**

**System.out.println(“Average = ” + average);**

The output is:

Average = 3

which is not the average of 2 and 5.

**The modulus operator %**

In Java, the modulus operator may have floating point operands. In ICS 102 we will only consider this operator with positive integer operands. When applied to two integers the result is the remainder of dividing the integers. Examples:

|  |  |
| --- | --- |
| Expression | Result |
| 7 % 2 | 1 |
| 12 % 2 | 0 |
| 8 % 3 | 2 |
| 4 % 7 | 4 |
| 9 % 12 | 9 |

**Example:** Write a Java Program fragment that converts 34 days into weeks and days:

Solution: There are 7 days in one week, hence:

**int days, weeks, givenNumDays = 34;   
weeks = givenNumDays / 7;  
days = givenNumDays % 7;  
System.out.println("There are " + weeks + " weeks and " + days + " days in " + givenNumDays + " days");**

1. **Mixed-mode arithmetic operations and Mixed-mode assignments**
   1. **Automatic type conversions**

If an arithmetic expression contains operands of different primitive types, then an operand of a lower type is automatically converted to a higher type. A conversion from a lower type to a higher type is called a **widening conversion**:

|  |
| --- |
| Low type High type |
| byte 🡪 short 🡪 int 🡪 long 🡪 float 🡪 double |

The result of the expression is that of the higher type. When values of type byte and short are used in arithmetic expressions, they are temporarily converted

to type int.

An expression of a lower type may be assigned to a variable of a higher type. The following are all valid assignment statements:

**int x = 3;**

**long z = x \* 2;**

**float y = z + 4L;**

**double w = y \* 2;**

We can use mixed-mode arithmetic in the example of finding the average of two integers:

**int num1 = 2, num2 = 5;**

**double average;**

**average = (num1 + num2 ) / 2.0;**

**System.out.println(“Average = ” + average);**

The output is:

Average = 3.5

* 1. **Type casting**

In Java, you **cannot** assign the value of an expression of a higher type to a variable of a lower type. Such type of conversions are called narrowing conversions. The following are all invalid assignment statements:

**short num1 = 32 ; // by default 32 is of type int**

**int num2 = 45L;**

**int num3 = 3 \* 2.0;**

**float num4 = 45.7; // by default 45.7 is of type double**

Each one of the above will produce a compilation error: **“error: possible loss of precision”**

Value casting or type casting consists of converting a value of one type into a value of another type. For example, you may have an integer value and you may want that value in an expression that expects a short number. Value casting is also referred to as explicit conversion.

To cast a value or an expression, precede it with the desired data type in parentheses. Examples:

**short num1 = (short) 32 ;**

**double num2 = 24.32;**

**int num3 = (int) num2 \* 2;**

**float num4 = 45.7F / (float) num2;**

**char ch = (char) 65; // Note: integers from 0 to 65535 may be assigned to char without casting**

When performing explicit conversion, you must ensure that the value being cast does not overflow or underflow. For example, if you want an integer value to be assigned to a short variable, the value must fit in 16 bits, which means it must be between -32768 and 32767. Any value beyond this range would produce an unpredictable result. Consider the following program fragment:

**int iNumber = 680044;**

**short sNumber = (short)iNumber;**

**System.out.println("Number = " + iNumber);**

**System.out.println("Number = " + sNumber);**

This would produce:

Number = 680044

Number = 24684

Notice that the result is not reasonable.

Note:

* When you convert a floating-point type to an integer type, any fractional portion is truncated (discarded). **Note**: The value is not rounded.
* The operand of the cast operator is not changed by the operation. A copy of the value is made that is of the specified data type.

1. **The Math class**

To assist you with various types of calculations, the **java.lang** package contains a class named **Math**. In this class are the most commonly needed mathematical methods and constants:

|  |  |  |  |
| --- | --- | --- | --- |
| Constant or Method |  | Comment | Method return type or type of constant |
| E | *e* | e, the base of the natural logarithms. | double |
| PI | π |  | double |
| abs(x) | |x| | Absolute value of x. | double, int, float, or long depending on type of x |
| sqrt(x) |  | Square root of x. | double |
| [cbrt](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Math.html#cbrt(double))(x) |  | Cube root of x. | double |
| sin(x) |  | Sine of angle x, x is in radians. | double |
| cos(x) |  | Cosine of angle x, x is in radians. | double |
| tan(x) |  | Tangent of angle, x is in radians. | double |
| toDegrees(angrad) |  | Converts an angle in radians to the equivalent angle in degrees. | double |
| toRadians(angdeg) |  | Converts an angle in degrees to the equivalent angle in radians. | double |
| [asin](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Math.html#asin(double))(x) | sin -1 x | arc sine of a value x; the returned angle is in the range -*π* / 2 through *π* / 2. | double |
| [acos](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Math.html#acos(double))(x) | cos -1 x | arc cosine of a value x; the returned angle is in the range 0.0 through *π*. | double |
| [atan](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Math.html#atan(double))(x) | tan -1 x | Returns the arc tangent of a value; the returned angle is in the range -*π*/2 through *π*/2. | double |
| exp(x) |  | *e* is 2.7182... | double |
| log(x) | ln x | Returns the natural logarithm (base *e*) of x | double |
| [log10](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Math.html#log10(double))(x) | log 10 x | Returns the base 10 logarithm of x | double |
| round(x) |  | Rounds x.  Examples: round(3.4) = 3 , round(3.6) = 4 , round(-3.4) = -3, round(-3.9) = -4 | **long if x is double, int if x is float** |
| rint(x) |  | Rounds x if its decimal part ≥ 0.5; otherwise x is truncated. A negative number with decimal part ≥ 0.5 is rounded down.  Examples: rint(5.499999) = 5.0, rint(5.5) = 6.0, rint(5.8) = 6.0,  rint(-5.2) = -5.0, rint(-5.5) = -6.0 | **double** |
| [ceil](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Math.html#ceil(double))(x) | ⎡x⎤ | Returns the **smallest** double value that is **greater** than or equal to x, and is equal to a mathematical integer.  Examples: ceil(5.2) = 6.0, ceil(12.7) = 13.0, ceil(6.0) = 6.0,  ceil(-4.3) = -4.0 | double |
| [floor](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Math.html#floor(double))(x) | ⎣ x ⎦ | Returns the **largest** double value that is **less** than or equal to x and is equal to a mathematical integer.  Examples: floor(5.9) = 5.0, floor(12.3) = 12.0, floor(6.0) = 6.0, floor(-4.3) = -5.0 | double |
| **max**(x, y) |  | Maximum of x and y. | int, long, float, or double depending on the types of x and y |
| **min**(x, y) |  | Minimum of x and y. | int, long, float, or double depending on the types of x and y |
| **pow**(x, y) | xy |  | double |
| [hypot](file:///E:\122\ics102-122\Applications\jdk-7u11-apidocs\docs\api\java\lang\Math.html#hypot(double, double))(x, y) |  |  | double |
| **random**() |  | Returns a pseudo-random value in the interval [0.0, 1.0) | double |

To access any of these methods or constants, we need to prefix it with the name of the class **Math**. Examples:

// Assume all variables are initialized

double sqrtDiscriminant = Math.sqrt(b\*b – 4\*a\*c);

double root1 = (-b + sqrtDiscriminant)/ (2\*a);

double root2 = (-b - sqrtDiscriminant) / (2\*a);

double circleArea = 2 \* Math.PI \* Math.pow(radius, 2);

**Examples on the use of ceil and floor methods:**

A car park charges 3.50 Saudi Riyals for each hour or a fraction of an hour of parking. Write a Java program fragment that given the amount of parking hours (a double value), it calculates and displays the parking bill.

**final double CHARGE = 3.50;   
double hours, bill;  
System.out.println("Enter number of hours");  
//... Statement to input hours  
bill = CHARGE \* Math.ceil(hours);  
System.out.println("Bill = " + bill + " Saudi Riyals");**

An employee is paid a retirement benefit according to the actual number of complete years he has served (excluding any fractional part of the last year of service, if any). The benefit is calculated as:

**actual number of years served \* base salary at time of retirement**

Write a Java program fragment that given the number of years served (a double value) and the base salary, it calculates and displays the retirement benefit.

**double yearsServed, baseSalary, benefit;**

**System.out.println("Enter number of years and base salary");**

**//... Statement to input number of years and base salary**

**benefit = Math.floor(yearsServed) \* baseSalary;**

**System.out.println("Retirement Benefit = " + benefit + " Saudi Riyals");**

**Example on the random method:**

The random() method returns a pseudo-random number in the interval [0.0, 1.0). In other words: 0.0 <= Math.random() < 1.0. To get a number in a different range, you can perform arithmetic on the value returned by the random method. For example, to generate an integer between 0 and 9, you would write:

int number = (int)(Math.random() \* 10);

By multiplying the value by 10, the range of possible values becomes 0.0 <= number < 10.0.

1. **Output Formatting**

The print and println methods do not provide much control over the formatting of output. For example, the following code:

**int i = 2;  
 double r = Math.sqrt(i);  
 System.out.println("The square root of " + i + " is " + r);  
 double cost = 32.50;  
 System.out.println("The cost is " + cost + " Saudi Riyals");**

produces the output:

The square root of 2 is 1.4142135623730951  
 The cost is 32.5 Saudi Riyals

**The format and printf methods of System.out**

In addition to the **print** and **println** methods, the object **System.out**, has two other methods **format** and **printf** for performing output. You can use **format** or **printf** anywhere in your code where you have previously been using **print** or **println**.

The **format** and **printf** methods, are equivalent to one another. They format multiple arguments based on a ***format string***. The format string consists of a string constant embedded with ***format specifiers***. Except for the format specifiers, the format string is output unchanged. Calls to these methods have the form:

**System.out.format(FormatString , arg1, arg2, . . . , argN);**

**System.out.printf(FormatString , arg1, arg2, . . . , argN);**

where arg1, arg2, to argN are expressions.

All format specifiers begin with a **%** and end with a character ***converter*** that specifies the kind of formatted output being generated:

|  |  |  |
| --- | --- | --- |
| converter | Formatted value | Comment |
| d | integer |  |
| f | Fixed-point floating point |  |
| e or E | E-notation floating point | The case of the displayed e is the same as that of the converter |
| g or G | General floating point | Java decides to use E-notation or fixed-point floating point. You should not use width and/or precision with this converter. |
| s or S | String | If s is used the string is displayed in lowercase; otherwise if S is used the string is displayed in upper case |
| c or C | character | %c outputs a character as it is, %C outputs a lowercase character in uppercase. |
| n |  | %n outputs a new line character appropriate to the platform running the application. You should always use %n, rather than \n. |
| % |  | %% outputs % |

**Note:** **Except for %% and %n, all format specifiers must match an argument; otherwise a run-time error is generated.**

A format specifier has the following structure:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **%** | **Flags** | **Width** | **. Precision** | **Converter** |

where the shaded part is optional.

* The width is the minimum width of the formatted value; the value is padded if necessary. By default the value is left-padded with blanks.For the line separator conversion **n**, width is not applicable; if it is provided, a run-time error will be generated.
* For the floating-point conversions **e**, **E**, and **f** the precision is the number of digits after the decimal separator. If the conversion is **g** or **G**, then the precision is the total number of digits in the resulting magnitude after rounding. For the conversion **s** or **S**, precision is the number of characters displayed starting from the beginning of the displayed string.

**Flags**

The following table summarizes the supported flags, **yes** means the flag is supported for the indicated argument types and **no** means it is not supported.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Flag** | **character** | **integral** | **floating point** | **Description** |
| '-' | yes | yes | yes | The result will be left-justified. |
| '+' | no | yes | yes | The result will always include a sign + or - |
| ' ' | no | yes | yes | The result will include a leading space for positive values |
| '0' | no | yes | yes | The result will be zero-padded |
| **‘,’** | no | yes | yes | Groups of three digits in the result will be separated by commas |

**Format Examples:**

Example1: The code:

**int i = 2;  
 double r = Math.sqrt(i);  
 System.out.format("The square root of %d is %7.3f%n", i, r);  
 double cost = 32.50;  
 System.out.printf("The cost is %.2f Saudi Riyals%n", cost);**

produces the output:

The square root of 2 is 1.414  
 The cost is 32.50 Saudi Riyals

**Note:** Like the **print** method, both **format** and **printf** methods do not generate new line. To generate new line **%n** or **\n** is used at the end of the format string.

# The value 2 is output using the format specifier %d. The value of r is output using the specifier %7.3, it is output in a total width of 7 and with two leading blanks and 3 digits after the decimal point:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **1** | **.** | **4** | **1** | **4** |

The value of **cost** is output using the format specifier **%.2f** with two digits after the decimal point and in the minimum width required for 32.50:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 2 | . | 5 | 0 |

Example2:

**long n = 461012;**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | output | | | | | | | | | | | | | |
| System.out.format("%d%n", n); | 4 | 6 | 1 | 0 | 1 | 2 |  |  |  |  |  |  |  |  |
| System.out.format("%08d%n", n); | 0 | 0 | 4 | 6 | 1 | 0 | 1 | 2 |  |  |  |  |  |  |
| System.out.format("%+8d%n", n); |  | + | 4 | 6 | 1 | 0 | 1 | 2 |  |  |  |  |  |  |
| System.out.format("%,8d%n", n); |  | 4 | 6 | 1 | , | 0 | 1 | 2 |  |  |  |  |  |  |
| System.out.format("%+,8d%n%n", n); | + | 4 | 6 | 1 | , | 0 | 1 | 2 |  |  |  |  |  |  |

**double pi = Math.PI;**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | output | | | | | | | | | | | | | |
| System.out.format("%f%n", pi); | 3 | . | 1 | 4 | 1 | 5 | 9 |  |  |  |  |  |  |  |
| System.out.format("%.3f%n", pi); | 3 | . | 1 | 4 | 2 |  |  |  |  |  |  |  |  |  |
| System.out.format("%10.3f%n", pi); |  |  |  |  |  | 3 | . | 1 | 4 | 2 |  |  |  |  |
| System.out.format("%-10.3f%n", pi); | 3 | . | 1 | 4 | 2 |  |  |  |  |  |  |  |  |  |

double num = -75.9584;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | output | | | | | | | | | | | | | |
| System.out.printf(“num = %8.3f\n”, num); | n | u | m |  | = |  |  | - | 7 | 5 | . | 9 | 5 | 8 |
| System.out.printf(“num = %7.3f\n”, num); | n | u | m |  | = |  | - | 7 | 5 | . | 9 | 5 | 8 |  |
| System.out.printf(“num = %6.3f\n”, num); | n | u | m |  | = |  | - | 7 | 5 | . | 9 | 5 | 8 |  |
| System.out.printf(“num = %.3f\n”, num); | n | u | m |  | = |  | - | 7 | 5 | . | 9 | 5 | 8 |  |
| System.out.printf(“num = %.2f\n”, num); | n | u | m |  | = |  | - | 7 | 5 | . | 9 | 6 |  |  |
| System.out.printf(“num = %.1f\n”, num); | n | u | m |  | = |  | - | 7 | 6 | . | 0 |  |  |  |

**Note:** Java raises a run-time error if the width of a format specifier is 0, for example: %0.3f is not valid

String city = “Dhahran”;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | output | | | | | | | | | | | | | |
| System.out.printf("%s%n", city); | D | h | a | h | r | a | n |  |  |  |  |  |  |  |
| System.out.printf("%S%n", city); | D | H | A | H | R | A | N |  |  |  |  |  |  |  |
| System.out.printf("%4S%n", city); | D | H | A | H | R | A | N |  |  |  |  |  |  |  |
| System.out.printf("%10s%n", city); |  |  |  | D | h | a | h | r | a | n |  |  |  |  |
| System.out.printf("%.2s%n", city); | D | h |  |  |  |  |  |  |  |  |  |  |  |  |
| System.out.printf("%.4s%n", city); | D | h | a | h |  |  |  |  |  |  |  |  |  |  |

# 16. Exercises



Design a pseudo-code algorithm and then translate it into a Java program that defines and initializes the **volume** v of a sphere in cubic centimeters. It then displays the volume in cubic centimetres and the **surface area** **s** of the sphere in square centimeters.

|  |  |
| --- | --- |
|  | π r3  s = 4 π r2 |

Sample program outputs:

|  |  |  |
| --- | --- | --- |
|  |  |  |



Inflation is defined as the loss of purchasing power of a given currency over time. Let us assume that money loses 3% of its value every year. This means that an amount of money next year will equal to only 97% of its value this year.

Let us define the following equation to calculate inflation:

Design a pseudo-code algorithm and then translate it into a Java program that defines and initializes a variable currentAmount to an amount in Saudi Riyals. It then calculates and displays:

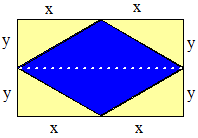
* the equivalent amount in two given years in the future.
* the equivalent amount in a given year in the past.

Sample Program outputs:

|  |  |
| --- | --- |
|  |  |



Consider the following figure:



Design a pseudo-code algorithm and then translate it into a Java program that defines and initializes the lengths x and y. It then displays the lengths x and y, and finally it calculates and displays the area and the perimeter of the blue figure. Assume that the lengths x and y are in centimeters.

**Hint:** Area of triangle is (base \* height)/2

Sample program outputs:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Design a pseudo-code algorithm and then translate it into a Java program that initializes the amount of change, in Saudi Riyals, to be returned to a customer at a shop. The program then calculates and displays the minimum number of Saudi Riyal bills for the change and the equivalent number of 50, 10, 5, and 1 Riyal bills. Assume that the bills available at the shop are 1, 5, 10, and 50 Riyals.

**Hint:** Use remainder and integer division.

Sample program outputs:

|  |  |  |
| --- | --- | --- |
|  |  |  |