

SS-1201 Lab02 (Part 1/3) –Arithmetic Expressions, Computations and Output Formatting

Objectives:

1. Introduction to:

Completed?	Sub-topics
Yes / No	Compound assignment operators
Yes / No	Increment and Decrement Operators
Yes / No	Automatic Type conversions
Yes / No	Type casting
Yes / No	The Math class
Yes / No	Output Formatting using format specifiers
Yes/ No	Arithmetic Computations

2. Complete practice and increase your understanding using examples

3. Complete all Lab tasks/exercises

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.

2. 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
--------------------------	---------------

<pre>int n = 8; int result = n++; System.out.println("n = " + n + ", result = " + result);</pre> <p>Output: n = 9, result = 8</p>	<pre>int n = 8; int result = ++n; System.out.println("n = " + n + ", result = " + result);</pre> <p>Output: n = 9, result = 9</p>
<pre>double x = 2.0, y; y = x++ * 5.0; System.out.println("x = " + x + ", y = " + y);</pre> <p>Output: x = 3.0, y = 10.0</p>	<pre>double x = 2.0, y; y = ++x * 5.0; System.out.println("x = " + x + ", y = " + y);</pre> <p>Output: x = 3.0, y = 15.0</p>

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

3. 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;
int final NUM_DAYS_IN_WEEK = 7;
weeks = givenNumDays / NUM_DAYS_IN_WEEK;
days = givenNumDays % NUM_DAYS_IN_WEEK;
System.out.println("There are " + weeks + " weeks and " + days + " days in " + givenNumDays + " days");
```

4. Mixed-mode arithmetic operations and Mixed-mode assignments

4.1 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

4.2 Type casting

Java primitive data types arranged from low to high:

(low) byte → short → char → int → long → float → double (high)

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 exceptions to this rule are:

- A **byte** variable can be assigned a constant integer in the range -128 to 127 inclusive without type casting
- A **short** variable can be assigned a constant integer in the range -32768 to 32767 inclusive without type casting
- A **char** variable can be assigned a constant integer in the range 0 to 65535 inclusive without type casting

Note: A type cast is required if the assigned value is in a variable:

```
int num1 = 32767;  
short num = (short) num1;
```

The following are all invalid assignment statements:

```
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:

```
double num2 = 24.32;
int num3 = (int) num2 * 2;
float num4 = 45.7F / (float) num2;
```

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.

5. 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	PI
abs(x)	$ x $	Absolute value of x.	double, int, float, or long depending on type of x
sqrt(x)	\sqrt{x}	Square root of x.	double
cbrt(x)	$\sqrt[3]{x}$	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(x)	$\sin^{-1} x$	arc sine of a value x; the returned angle is in the range $-\pi / 2$ through $\pi / 2$.	double
acos(x)	$\cos^{-1} x$	arc cosine of a value x; the returned angle is in the range 0.0 through π .	double
atan(x)	$\tan^{-1} x$	Returns the arc tangent of a value; the returned angle is in the range $-\pi/2$ through $\pi/2$.	double
exp(x)	e^x	e is 2.7182...	double
log(x)	$\ln x$	Returns the natural logarithm (base e) of x	double
log10(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(x)	$\lceil x \rceil$	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(x)	$\lfloor x \rfloor$	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)	x^y		double
hypot(x, y)	$\sqrt{x^2+y^2}$		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 BND 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 + " BND");
```

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 + " BND");
```

Example on the random method:

The random() method returns a pseudo-random number in the interval [0.0, 1.0). In other words: $0.0 \leq \text{Math.random()} < 1.0$. 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 $0.0 \leq \text{number} < 10.0$.

6. 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 + " BND");
```

produces the output:

The square root of 2 is 1.4142135623730951 The cost is 32.5 BND

The printf methods of System.out

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

The **printf** method formats multiple arguments based on a *format string*. The format string consists of a string constant embedded with *format specifiers*. Except for the format specifiers, other characters in the format string are output unchanged. Calls to this method have the form:

```
System.out.printf(FormatString , expression1, expression2, . . . , expressionN);
```

For each expression there must be a corresponding format specifier (fs):

```
System.out.printf(" fs1 fs2 . . . fsN " , expression1, expression2, . . . , expressionN);
```

Example:

```
double y = 5.6;
```

```
System.out.printf("x = %d, y = %f cm" , 2 * 3, y);
```

outputs:

```
x = 6, y = 5.600000 cm
```

A format specifier has the following structure:

%	Flags	Width	. Precision	Converter
---	-------	-------	-------------	-----------

where the shaded part is optional; it begins 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
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: **IllegalFormatConversionException** is generated.

Width: The minimum total width of the formatted output, including a decimal point, a + or a – sign, if any.

Precision: For floating point (**double** or **float**), precision is the number of digits after the decimal point.

Flag: A character that specifies additional formatting:

Flag	Description
'-'	The result will be left-justified.
'+'	A numeric output will always include a sign + or -

'0'	A numeric output will be zero-padded
' '	space will display a minus sign if numeric output is negative or a space if it is positive
','	Groups of three digits in a numeric output 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 BND%n", cost);
System.out.printf("Number of people = %,d",5673458);
```

produces the output:

```
The square root of 2 is 1.414
The cost is 32.50 BND
Number of people = 5,673,458
```

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

In the above example, 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
---	---	---	---	---

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

Example2: double num = -52.34578;

	output								
System.out.printf("%-9.2f%n", num);	-	5	2	.	3	5			
System.out.printf("%9.2f", num);				-	5	2	.	3	5

Example3: String city = "Gadong";

	output												
System.out.printf("%s%n", city);	G	a	d	o	n	g							
System.out.printf("%S%n", city);	G	A	D	O	N	G							
System.out.printf("%4S%n", city);	G	A	D	O	N	G							
System.out.printf("%10s%n", city);					G	a	d	o	n	g			
System.out.printf("%-10s%n", city);	G	a	d	o	n	g							

7. Arithmetic Computations: How to solve problems computationally?

For example, suppose you are asked to write a program that simulates a vending machine. A customer selects an item for purchase and inserts a bill into the vending machine. The vending machine dispenses the purchased item and gives change. We will assume that all item prices are multiples of 25 cents, and the machine gives all change in dollar coins and quarters. Your task is to compute how many coins of each type to return.

Step 1 Understand the problem: What are the inputs? What are the desired outputs? In this problem, there are two inputs:

- The denomination of the bill that the customer inserts
- The price of the purchased item

There are two desired outputs:

- The number of dollar coins that the machine returns
- The number of quarters that the machine returns

Step 2 Work out examples by hand. This is a very important step. If you can't compute a couple of solutions by hand, it's unlikely that you'll be able to write a program that automates the computation. *Let's assume that a customer purchased an item that cost \$2.25 and inserted a \$5 bill. The customer is due \$2.75, or two dollar coins and three quarters, in change. That is easy for you to see, but how can a Java program come to the same conclusion? The key is to work in pennies, not dollars. The change due the customer is 275 pennies. Dividing by 100 yields 2, the number of dollars. Dividing the remainder (75) by 25 yields 3, the number of quarters.*

Step 3 Write pseudocode for computing the answers. In the previous step, you worked out a specific instance of the problem. You now need to come up with a method that works in general. Given an arbitrary item price and payment, how can you compute the coins due? First, compute the change due in pennies:
change due = 100 x bill value - item price in pennies

To get the dollars, divide by 100 and discard the remainder:
dollar coins = change due / 100 (without remainder)

The remaining change due can be computed in two ways. If you are familiar with the modulus operator, you can simply compute
change due = change due % 100

Alternatively, subtract the penny value of the dollar coins from the change due:

$\text{change due} = \text{change due} - 100 \times \text{dollar coins}$

To get the quarters due, divide by 25:

$\text{quarters} = \text{change due} / 25$

Step 4 Declare the variables and constants that you need, and specify their types. Here, we have five variables:

- billValue
- itemPrice
- changeDue
- dollarCoins
- quarters

Should we introduce constants to explain 100 and 25 as PENNIES_PER_DOLLAR and PENNIES_PER_QUARTER? Doing so will make it easier to convert the program to international markets, so we will take this step. It is very important that changeDue and PENNIES_PER_DOLLAR are of type int because the computation of dollarCoins uses integer division. Similarly, the other variables are integers.

Step 5 Turn the pseudocode into Java statements. If you did a thorough job with the pseudocode, this step should be easy. Of course, you have to know how to express mathematical operations (such as powers or integer division) in Java.

```
changeDue = PENNIES_PER_DOLLAR * billValue - itemPrice;
```

```
dollarCoins = changeDue / PENNIES_PER_DOLLAR;
```

```
changeDue = changeDue % PENNIES_PER_DOLLAR;
```

```
quarters = changeDue / PENNIES_PER_QUARTER;
```

Step 6 Provide input and output. Before starting the computation, we prompt the user for the bill value and item price:

```
System.out.print("Enter bill value (1 = $1 bill, 5 = $5 bill, etc.): ");
```

```
billValue = in.nextInt();
```

```
System.out.print("Enter item price in pennies: ");
```

```
itemPrice = in.nextInt();
```

When the computation is finished, we display the result. For extra credit, we use the printf method to make sure that the output lines up neatly.

```
System.out.printf("Dollar coins: %6d", dollarCoins);
```

```
System.out.printf("Quarters: %6d", quarters);
```

Step 7 Provide a class with a main method. Your computation needs to be placed into a class. Find an appropriate name for the class that describes the purpose of the computation. In our example, we will choose the name Vending-Machine. Inside the class, supply a main method.

In the main method, you need to declare constants and variables (Step 4), carry out computations (Step 5), and provide input and output (Step 6).

Clearly, you will want to first get the input, then do the computations, and finally show the output.

Declare the constants at the beginning of the method, and declare each variable just before it is needed.

Here is the complete program, how_to_1/VendingMachine.java:

```
/**
 * This program simulates a vending machine that gives change.
 */
public class VendingMachine
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        final int PENNIES_PER_DOLLAR = 100;
        final int PENNIES_PER_QUARTER = 25;

        System.out.print("Enter bill value (1 = $1 bill, 5 = $5 bill, etc.): ");
        int billValue = in.nextInt();
        System.out.print("Enter item price in pennies: ");
        int itemPrice = in.nextInt();

        // Compute change due

        int changeDue = PENNIES_PER_DOLLAR * billValue - itemPrice;
        int dollarCoins = changeDue / PENNIES_PER_DOLLAR;
        changeDue = changeDue % PENNIES_PER_DOLLAR;
        int quarters = changeDue / PENNIES_PER_QUARTER;

        // Print change due

        System.out.printf("Dollar coins: %6d", dollarCoins);
        System.out.println();
        System.out.printf("Quarters:      %6d", quarters);
        System.out.println();
    }
}
```

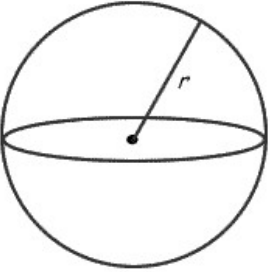
Program Output:

Enter bill value (1 = \$1 bill, 5 = \$5 bill, etc.): 5
Enter item price in pennies: 225
Dollar coins: 2
Quarters: 3

8. Lab Tasks /Exercises

Each task/exercise is worth 10 marks.

Task 01: Write the analysis, pseudo-code algorithm and a then a corresponding complete 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.

 $v = \frac{4}{3} \pi r^3$ $s = 4 \pi r^2$	Sample program output:		
	Volume = 15.75 Cubic cm Surface Area = 30.39 Square cm	Volume = 20.00 Cubic cm Surface Area = 35.63 Square cm	Volume = 6.50 Cubic cm Surface Area = 16.84 Square cm

Task 02: 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:

$$OtherAmount = CurrentAmount * 0.97^{OtherYear - CurrentYear}$$

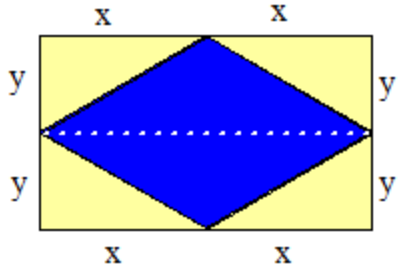
Write the analysis, pseudo-code algorithm and a corresponding complete Java program that defines and initializes a variable currentAmount to an amount in BND. It then calculates and displays:

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

Sample Program outputs [assuming the current year is 2013 and 2017]:

The current amount is: 1000.0 The current year is: 2013 The amount for year 2009 is 1129.5697747731626 The amount for year 2025 is 693.8423609954377 The amount for year 2040 is 439.3765001738383	The current amount is: 1000.0 The current year is: 2017 The amount for year 2015 is 1062.8122010840686 The amount for year 2020 is 912.673 The amount for year 2025 is 783.7433594376959
--	--

Task 03: 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 $(\text{base} * \text{height})/2$

Length x = 4.00 cm Length y = 3.00 cm Perimeter of Blue area = 20.00 cm Blue area = 24.00 square cm	Length x = 12.00 cm Length y = 5.00 cm Perimeter of Blue area = 52.00 cm Blue area = 120.00 square cm	Length x = 3.67 cm Length y = 2.45 cm Perimeter of Blue area = 17.65 cm Blue area = 17.98 square cm
--	--	--

Task 04: Design a pseudo-code algorithm and then translate it into a Java program that initializes the amount of change, in BND, to be returned to a customer at a shop. The program then calculates and displays the minimum number of BND bills for the change and the equivalent number of 50, 10, 5, and 1 BND bills. Assume that the bills available at the shop are 1, 5, 10, and 50 BND. **Hint:** Use remainder and integer division.

Sample program outputs:

Change Amount: 532 BND Minimum number of bills: 15 Number of fifty dollar bills: 10 Number of ten dollar bills: 3 Number of five dollar bills: 0 Number of one dollar bills: 2	Change Amount: 268 BND Minimum number of bills: 10 Number of fifty dollar bills: 5 Number of ten dollar bills: 1 Number of five dollar bills: 1 Number of one dollar bills: 3	Change Amount: 28 BND Minimum number of bills: 6 Number of fifty dollar bills: 0 Number of ten dollar bills: 2 Number of five dollar bills: 1 Number of one dollar bills: 3
---	--	--

Task 05: Write a program that accepts the unit weight of a bag of Sugar in Kilograms and the number of bags sold and displays the total price of the sale, computed as
`totalPrice = unitWeight * numberOfUnits * 5.99;`
`totalPriceWithTax = totalPrice + totalPrice * 0.0725;`
 where 5.99 is the cost per Kg and 0.0725 is the sales tax. Display the result in the following manner:

```
Please enter the number of bags: 32
Please enter the weight per bag (in Kg): 5
Price per Kg: $5.99
Sales tax: 7.25%
Total price: $ 1027.884
```

7. Naming Conventions

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it's a constant, package, or class-which can be helpful in understanding the code.

Identifier Type	Rules for Naming	Examples
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	class Raster; class ImageSprite;
Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate; interface Storing;
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	run(); runFast(); getBackground();
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign \$ characters, even though both are allowed.	int i; char c; float myWidth;

	Variable names should be short yet meaningful. The choice of a variable name should be mnemonic—that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are <code>i</code> , <code>j</code> , <code>k</code> , <code>m</code> , and <code>n</code> for integers; <code>c</code> , <code>d</code> , and <code>e</code> for characters.	
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4; static final int MAX_WIDTH = 999; static final int GET_THE_CPU = 1;