# Calculations, Variables and Assignments

## Doing Calculations in Java

Java, like all high-level programming languages, provides support for the following mathematical operators:

| | |
|---|---|
| Addition | + |
| Subtraction | - |
| Multiplication | * |
| Division | / |
| Modulus | % |

The modulus operator calculates the remainder after division. It evaluates as follows:

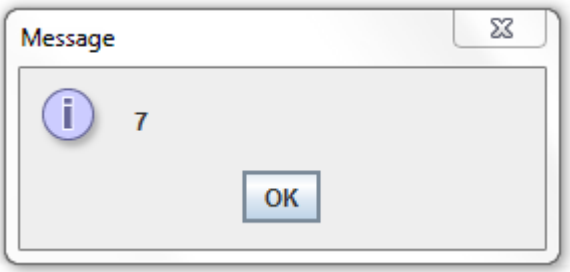
- 4 % 2 evaluates to 0 because 4 divided by 2 is 2, with the remainder being 0
- 5 % 2 evaluates to 1 because 5 divided by 2 is 2, with the remainder being 1
- 14 % 3 evaluates to 2 because 14 divided by 3 is 4, with the remainder being 2

**Doing Calculations in your Program**

Doing calculations in Java is pretty straight forward. Consider the following code:

```
JOptionPane.showMessageDialog(null,3 + 4);
```

This will display the following dialog box:




Can you determine what took place? What Java did was it evaluated the addition on the fly, and outputted the answer. This is typical behaviour for Java as well as other high-level languages.

**Operator Precedence**

Operator precedence determines which operator is evaluated first of there are more than one. Consider the following example:

2 + 3 * 4

If the addition is evaluated first, the result will be 20; if the multiplication is evaluated first, the result will be 14. The question is, which will be evaluated first by Java?

Java evaluates operators in the following order:

() * / % + -

Hence:

2 + 3 * 4 will evaluate to 14

(2 + 3) * 4 will evaluate to 20, because the expression within the brackets is evaluated first

**Activity 2.4**
How will Java evaluate the following expressions?

1. `1 + 2 - 3 * 4 / 5`
2. 4 + 5 * 6 / 3
3. (1/2) * (4 + 7) / 6 - 4
4. ((7 – 3) * 5) – (3 * 4) / 2)

Write a Java program to test your answers.

## Variables and Assignments

In the examples above, Java evaluates the expressions on the fly, and outputs the results onto the screen directly. What if a situation arises where you need to evaluate an expression, but rather than output the answer, you need to store it in memory for later use? To do this, you can use a variable.

Variables provide a convenient way to store information in the computer's memory temporarily. When you store information in a variable, you can use that information whenever you need to in your program, by calling its name. You may even change the value at any stage.

Typically, a variable is made up of three parts:

1. The **type** of the variable: before you create a variable, you will need to decide exactly what is going to be stored in it. Are you going to store and integer number, a text message (known as a String in programming terminology), a floating point number or a single character?
2. The **name** or **identifier** of the variable: variables need to have names.
3. The **value** which is stored in the variable: the value has to be of the same type with which the variable is declared; otherwise Java will return an error. In other words, if a variable is declared as an integer, it can only store an integer, not a floating point number or a String.

**Declaring a Variable**

An integer variable is declared as follows:

```
int num;
```

A String is declared as follows (note the uppercase "S"):

```
String firstname;
```

A floating point variable is declared as follows:

```
double price;
```

Java also provides a variable type which stores a single character:

```
char letter;
```

In both the examples above, the first part denotes the type and the second part the name of the variable.

**Variable Names**

Your variables should always have meaningful names which will indicate what value is stored in the variable. For example, if you have a *String* type variable which stores a person's first name, and an *int* type which stores his age, you could name them as follows:

```
String f;

int a;
```

This will be perfectly fine as far as Java is concerned, but it will become difficult to remember what is stored in the variables, especially if your program has dozens of variables, which is quite typical. Compare the names above with the following names:

```
String firstname;

int age;
```

You will agree that these names are far more user-friendly.

There are also a few Java-imposed rules for naming variables:

1. A variable name must start with a letter of the alphabet, an underscore or a dollar sign.
2. From the second letter onwards, you may use numeric digits as well as letters of the alphabet, underscores and the dollar sign.
3. You may not use Java keywords, such as *public* and *static* as variable names.

The following are examples of legal variable names:

```
surname
_abc_
$total
totalPurchaseAmountForTheYear
```

The following are illegal variable names:

```
1surname
void
2012Totals
```

## The Assignment Statement

After declaring a variable, the next step is to place a value into it. This is done by using the assignment operator, which is the equals ( = ) sign. For example:

```
age = 25;
```

The statement above instructs Java to take whatever is on the right-hand-side of the ( = ) sign, and to place it into the variable “age”.

We can combine the declaration as well as the assignment, into one line, as shown below:

```
int age = 25;
```

Below are some examples of variable assignments:

```
firstname = "Sam"
letter = 'z'
distance = 2.5
```

**More Complex Assignments**

The right-hand-side of an assignment statement may be a single value as shown above, or it may be an expression. Look at the table below:

| | |
|---|---|
| `total = 5 + 7` | This will first evaluate the expression on the right-hand-side, and then it will assign the result to the variable *total*. The outcome will be that the variable *total* will have the value 12 stored in it. |
| `num1 = num2` | This will take the value which is stored in *num2*, and assign it to *num1*. |
| `num1 = num1 + 1` | This will add 1 to the existing value of *num1* |
| `num1++` | Same as above. This is a convenient shorthand method. |
| `num2 = num2 + 8` | This will add 8 to the existing value of *num2* |
| `num1 = num2 + 3` | This will add the value of *num2* and 3 to *num1*. The value of *num2* is not affected |

Consider the following code snippet:

```
int num1 = 5;
int num2 = 3;
num1 = num2;
```

Can you figure out what the resulting value in num1 will be? The correct answer is 6.

Look at the following code snippet:

```
int num1 = 5;
int num2 = 9;
num1 = num2 + 10;
```

Can you figure out what the resulting value of *num1* and *num2* will be? The value of *num1* will be 19, and *num2* will still be 9, because it is unaffected.

## Using Variables in your Programs

In this section we will write a program which makes use of the concepts covered in this chapter. The program will take the unit price and the quantity of an item, multiply them to get the total, and then output the total.

```
 1 import javax.swing.JOptionPane;
 2
 3 public class PriceCalculator
 4 {
 5    public static void main(String[] args)
 6    {
 7       int quantity;
 8       double unitPrice;
 9       double totalPrice;
10       quantity = 3;
11       unitPrice = 4.5;
12       totalPrice = unitPrice * quantity;
13       JOptionPane.showMessageDialog(null, "The total price is : R " + totalPrice);
14       System.exit(0);
15    }
16 }
```

You should be able to figure out what each line of the program is doing, but just to recap:

- Lines 7, 8 and 9 are declaring the three variables we will need.
- Lines 10 and 11 are assigning values to *quantity* and *unitPrice* respectively
- Line 12 calculates the value of *totalPrice* by multiplying *quantity* and *unitPrice*.
- Line 13 handles the output. The text message which will be sent to the dialog box is:

  ```
  "The total price is : R " + totalPrice
  ```

  This instructs Java to display the text message “The total price is : R” and to add to it the value of *totalPrice*.

  The + sign in the above code serves a different function to a normal plus sign; in this situation it is joining the value of *totalPrice* to the text message “The total price is : R”.

**Activity 2.5**
Modify the above program to calculate the VAT on the total price. The dialog box should display the following text:
*The total price including vat is: R xxx.xx*
The VAT rate is 14%
Hint: to calculate 14% of an amount, you can use the formula:

*amount / 100 * 14*

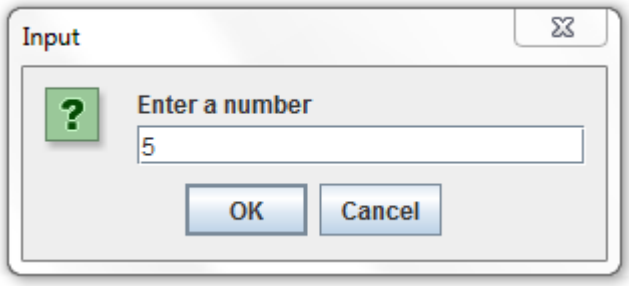
## Making your Programs Interactive

The programs you've written thus far did not interact with the user in any way. All they did was to take in values which were input directly in the code, and did some calculations on it. A program will be far more useful if it took inputs from the user instead.

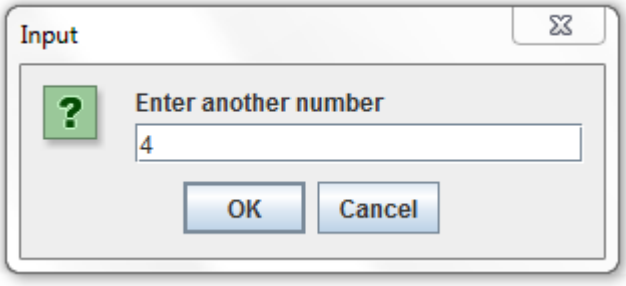
In this section we are going to look at writing programs which accept user input. The following program prompts the user to input 2 numbers, and then adds them up, and displays the result:

```
1   import javax.swing.JOptionPane;
2
3   public class InputsOutput
4   {
5      public static void main(String[] args)
6      {
7         int num1;
8         int num2;
9         int total;
10        String numString1;
11        String numString2;
12
13        numString1 = JOptionPane.showInputDialog("Enter a number");
14        numString2 = JOptionPane.showInputDialog("Enter another number");
15
16        num1 = Integer.parseInt(numString1);
17        num2 = Integer.parseInt(numString2);
18
19        total = num1 + num2;
20
21        JOptionPane.showMessageDialog(null,"The total is: " + total);
22        System.exit(0);
23     }
24  }
```

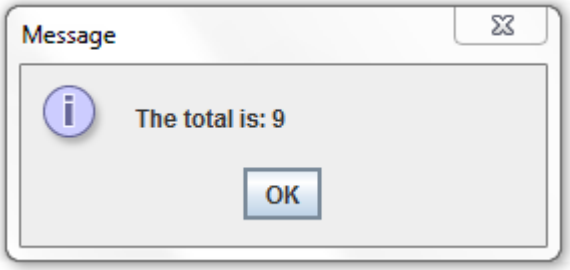
When you run the above program, it displays a dialog box with a text input field, and asks the user to type in a number:




After the user enters a number, the program displays another dialog box prompting for another number:




When the user presses *OK*, a third dialog box is displayed with the sum of the two numbers:




**The code explained:**

- Lines 7 to 11 declare all the variables we will need. You may be wondering why we are declaring two Strings, *numString1* and *numString2*, when all we are doing is adding two integers. This will become clear when we discuss lines 13 and 14 below.
- Line 13: In this line you are introduced to a new method, the showInputDialog() method. This method lives in the JOptionPane class, just like the showMessageDialog() method. The function of the showInputDialog() method is to display a dialog box with a text input field.

  When the user enters text into the text input field and presses *OK*, Java retrieves the text which the user typed, and stores in a variable, *numString1*, hence you see the assignment operator ( = ):

```
numString1 = JOptionPane.showInputDialog("Enter a number");
```

- Line 14: This line displays the second dialog box, and prompts the user to type in the second number, which is then stored in the variable *numString2*.

**If the user is entering numbers into the text boxes, then why are these being stored into *String* variables and not variables of type *int*?**

**Answer:** Java does not recognize the user inputs as numbers. As far as Java is concerned, everything that comes from a text input field is a String. As a result, if we attempt to store the user input into variables of type *int*, Java will return an error.

It is up to the programmer to convert the input Strings into integers, in order to be able to do calculations on them. This will happen in the next 2 lines of the program.

- Line 16: In this line, the String in numString1 is being converted into an integer. The method *parseInt()*, which lives in a class called *Integer*, is responsible for doing this. So line 16 instructs Java as follows:
  Take the String in the variable *numString1* and convert it into an integer value. Then, assign that integer value to the variable *num1*.
- Line 17: The String in the *numString2* is converted into an integer, and then assigned to *num2*.
- Line 19 until the end of the program: you should be able to figure out what these lines are doing.

**Activity 2.2**
Rewrite the **PriceCalculator** program from section SECTION to take in the price and quantity of the item from the user. The rest of the program should remain the same.