**Institute of Technology Tralee**

**Computing Department**

**Introduction to Programming**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical 8 – while & do-while loops**

We have now studied two application areas for **while** loops. This time we will look at the remaining application area for while – **task-controlled** iteration. We will also examine Java’s **do-while** loop, which is very closely related to the while loop.

**Task-Controlled while loops**

In this case the program needs to **keep the iteration process going until a certain task has been accomplished or a certain level or point has been reached**. This type of loop has many applications in control system software, for example, if we need the level of liquid in a tank to be at some desired level then we need to repeat a process such as allowing the liquid to escape until the desired level is reached and then suddenly stop it from escaping. It is also quite useful in mathematical type questions, such as those involving series and sums of numbers. The first example we shall examine will be of a mathematical nature.

Note that in the case of task-controlled while loops, **the user does not have any control over when the loop stops** and it **is not normally possible to predict in advance how many times the loop will iterate** in order to complete its task.

**The General Form of a Task Controlled while Loop**

In pseudocode terms, task-controlled while loops have the **general form**:

*while(task has not been completed)*

*{*

*statement1;*

*:*

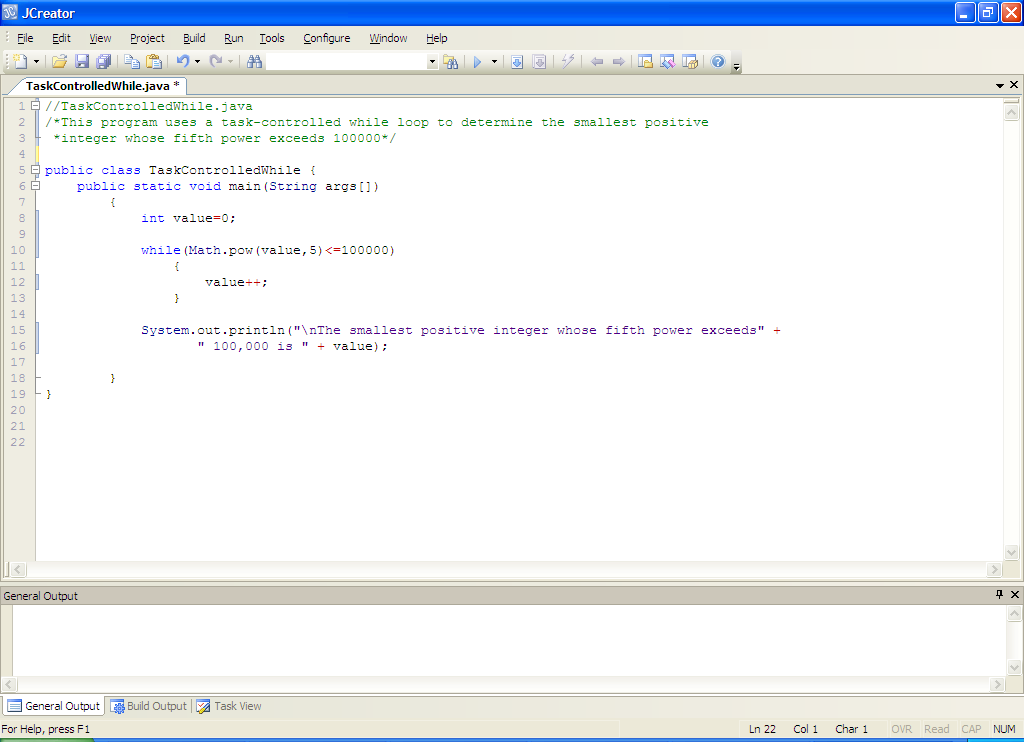
*statementn;*

*}*

The while loop expression tests to see if the task in question has been completed or if the required point has been reached. If it has not then the statements associated with the while loop will be executed. If however, the task has been completed or the desired point has been reached then the expression evaluates to false, the while loop exits and program execution continues at the statement immediately following the loop. As with all while loops, it is **possible that the statements associated with the loop may never be executed**. This will occur if the expression evaluates to false at the first time of testing the condition.

**Aim** **:** To write a program which will allow us to determine the smallest positive integer number whose fifth power is greater than 100,000 using a task-controlled while loop.

**Java Code:**



**Analysis of program:**

• The variable **value** will store the number we are seeking here and is initialised to zero. Then the while loop begins. It tests to see whether value to the power of 5 is less than or equal to 100000. As long as it remains less than or equal to this number, the loop will continue.

The only statement inside the loop in this case is

**value++;**

which just increments value by 1 each time. When the loop has stopped iterating the variable value holds the smallest positive integer number whose fifth power exceeds 100000 and this is output to the screen.

• The program uses the **pow**() method from Java’s Math class to get the value of one number raised to the power of another. Some other examples of this method being used are:

Math.pow(8,3) => this gives 83 which is 512

Math.pow(5.5,7.2)=>this gives 5.57.2 which is 214097.3858

Math.pow(6,0.5)=>this gives 60.5 which is the square root of 6 i.e. 2.44948743

This method is very useful where we have formulas that involve odd-ball powers such as square root or cube roots. Technically, we could have got away without it in this program by just coding the test expression as:

(value\*value\*value\*value\*value <= 100000)

but it’s a lot neater to use Math.pow() in this case

• One unusual thing about this program is that there is **no user input** required, so we have **no need to import the Scanner class**. There is nothing wrong with that of course – some programs are designed to simply work away by themselves without requiring any user interaction whatsoever – for example, virus-scanning software is often setup so that it runs automatically at a certain time every day or once a week. This program performs its task in the absence of any user interaction whatsoever. You will see in your **Operating Systems** module after Christmas that many operating system programs start running at bootup time and continue running without requiring any user interaction, even while the user is busy performing other tasks on the computer.

**Typing in and Saving Code for the Program Just Analysed**

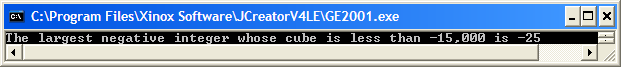
Click the **New File** icon on the JCreator IDE and save the file as **TaskControlledWhile.java** in your Lab8 folder. Now type in the code for the program above.

If your program has any errors or warnings, have a look at the edit window and check to ensure that the code is exactly as indicated earlier, including all **semicolons** (**;**) and concatenation operators (+) and ensuring that letters are written in lowercase where indicated. If you spot any differences correct them and compile again until the program is syntax error-free.

Once you are free from errors, run the program. There is only ever going to be one possible outcome from this type of program so it’s **very easy to test**.

**Exercise 1**

Write a program called **Exercise1.java** using a task-controlled while loop to determine the largest negative integer whose cube is less than -15000. The program should produce the following output:



**Exercise 2**

The **powers** of 2 are as follows:

20 = 1

21 = 2

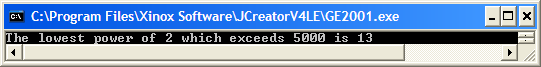
22 = 4

23 = 8

:

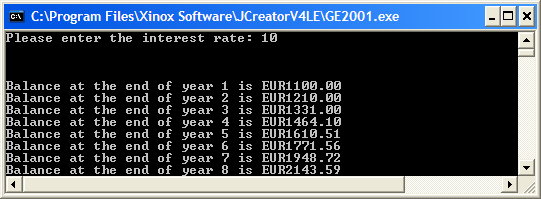
:

To get the next power of 2, you multiply the previous one by 2. Using a **while** loop, write a program that will find and display the **lowest** whole power of 2 that exceeds 5000. Your program should produce the following output:

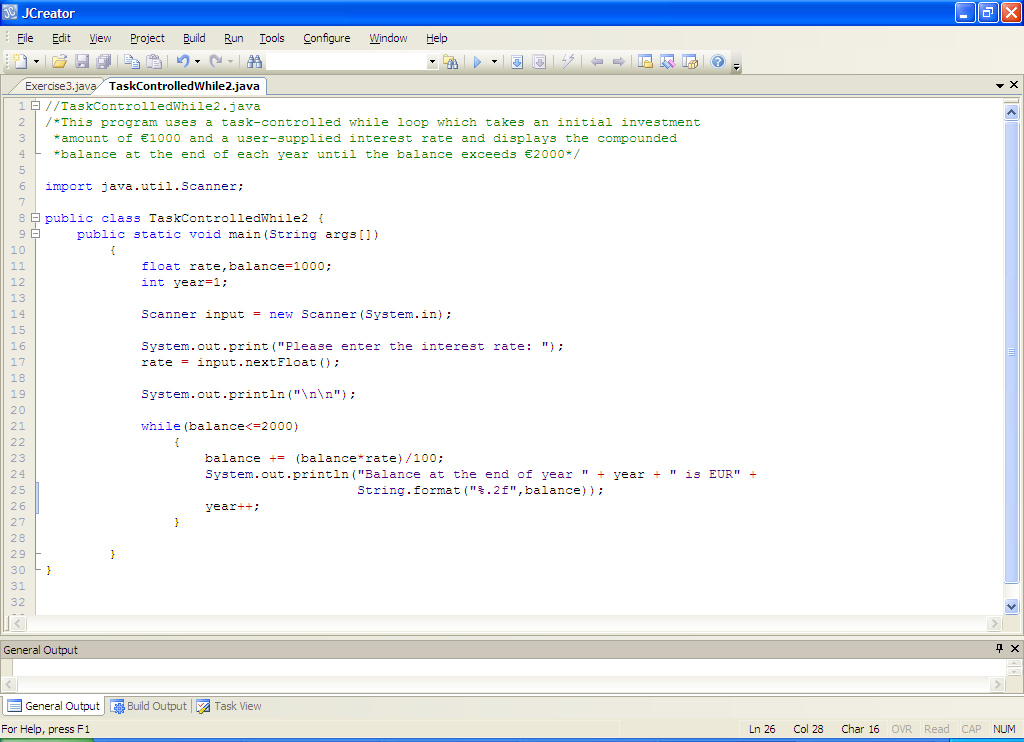


**Another Task-Controlled while Example**

**Aim** **:** To write a program which takes an initial investment amount of €1000 and a user-supplied rate of interest. The investment amount is to be compounded annually and the balance at the end of each year should be displayed until the balance first exceeds €2000. A sample run of the program would be as follows:



**Java Code:**



**Analysis of program:**

• balance gets initialized to 1000 to begin with.

• The user is asked for the interest rate that will apply and this is read in.

• The while loop test expression is

**(balance<=2000)**

As long as balance remains less than or equal to 2000 then the loop will keep going. The first time the loop is encountered, balance is 1000 and (1000<=2000) evaluates to true so the loop body will get executed.

• The first statement inside the while loop

**balance += (balance\*rate)/100;**

updates the balance, based on the interest rate supplied. The amount of interest earned would be given by the **(balance\*rate)/100** part. This interest is then added onto the current balance through the **+** in the **+=** operator and the result then stored in variable balance, to give us a brand new (compounded) balance, which is then displayed to 2 decimal places using **format**().

• The variable year then gets incremented

• The number of times the while loop iterates in this case is completely dependent on what the user supplies for the interest rate.

**Typing in and Saving Code for the Program Just Analysed**

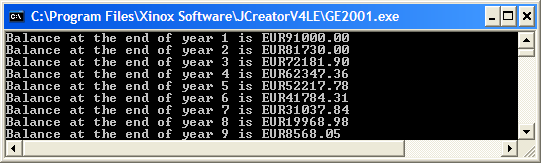
Click the **New File** icon on the JCreator IDE and save the file as **TaskControlledWhile2.java** in your Lab8 folder. Now type in the code for the program above.

If your program has any errors or warnings, have a look at the edit window and check to ensure that the code is exactly as indicated earlier, including all **semicolons** (**;**) and concatenation operators (+) and ensuring that letters are written in lowercase where indicated. If you spot any differences correct them and compile again until the program is syntax error-free.

Once you are free from errors, run the program. Use the **calculator** to make sure the figures are correct.

**Exercise 3**

A person starts off with a balance of €100000 and invests it at a rate of 3% compounded annually. At the end of the year, just after that years interest is added on, he will withdraw €12000 in a lump sum. What is the first year at the end of which his balance will not permit such a withdrawal? Save your program as **Exercise3.java**. Your program should display the balance at the end of each year to 2 decimal places (the balance should include the interest plus the withdrawal of the €12000) and behave as indicated in the following screenshot:



**The do-while Loop**

The do-while loop is very similar in its operation to the while loop which we have analysed in depth. There is one fundamental **difference** between the two looping structures however. It is **possible that the statements associated with a while loop may never be executed** (if the value of the test expression initially evaluates to false). This is not possible with the do-while structure, where the **statements associated with a do-while loop will always be** **guaranteed to execute at least once**. This is because the **test condition for a do-while is evaluated at the end of the loop body**.

This structure suits certain applications, where it is known that a certain group of statements should execute at least once, however **an equivalent while can always be used** for the coding. We will look at some examples of using do-while shortly - one area where they are especially popular are for **menu type programs** where the user can select from a list of options and perform some action.

**The general form of a do-while loop**

**do {**

*statement1*

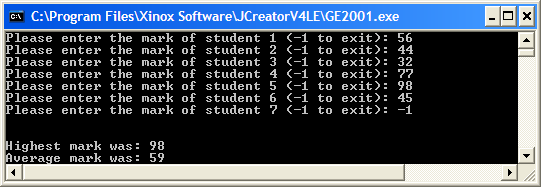
:

*statementn*

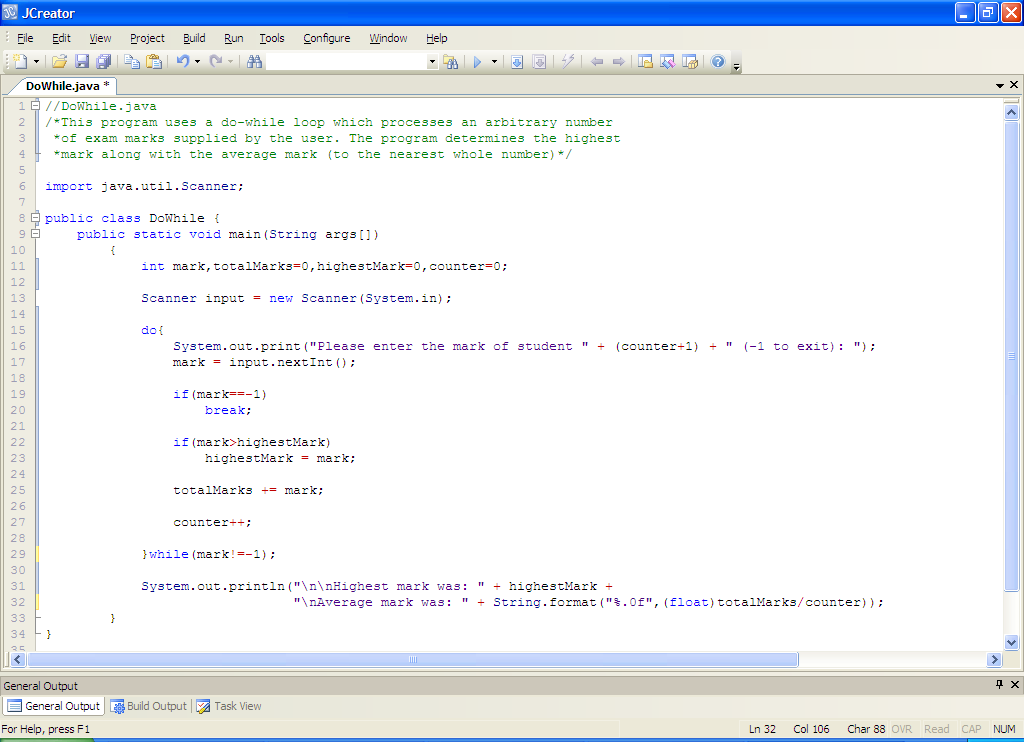
**} while** **(***expression***);**

it is clear from the form above that *statement1* to *statementn* will be executed at least once, with *expression* being evaluated at the end of each iteration of the loop. If expression evaluates to true then the loop iterates again, if it evaluates to false then the loop stops iterating. As normal, the words **do** and **while** are examples of Java **keywords**. As usual also, the **curly braces** and **parentheses** are essential. Note also the **semi-colon** after the test-expression – this is **unexpected** but is necessary and its omission would cause a **syntax error**.

**Aim**: To write a program that uses a **do-while** loop to process an arbitrary number of exam marks, supplied by the user. The program should display the highest mark and the average mark (to the nearest whole number). It would run as follows:



**Java Code**:



**Analysis of program:**

• several variables are initialized to zero to begin with. 2 of these are counter variables (totalMarks and counter) and the other (highestMark) is to keep track of the highest mark entered.

• The first iteration of the do-while loop begins by asking the user to enter a mark for a particular student. As this program involves an arbitrary number of marks, we must choose a suitable sentinel value for the situation. I have chosen -1 here so that there is no conflict with potential valid marks (which should all be positive). The user is told about the sentinel value as part of the prompt so that, when the program runs, they’ll know what value to enter in order to bring the loop to a halt. So what we have here really is a **data-sentinel controlled do-while loop**, which operates in a very similar fashion to the while loop version.

• Because the test takes place at the end of the do-while loop rather than at the start for the while loop, we must compensate for this in our code. I do this as follows:

**if(mark==-1)**

**break;**

here we check whether the user has just entered -1 for the mark. If they have then we immediately exit the loop with the **break** as we don’t want to process the value -1 as if it were a proper mark value.

If the user does not enter -1 the break is bypassed and the mark is processed.

There are several other ways of avoiding the processing of the -1 value here but this technique works fine. Can you spot another way?

• The code

**if(mark>highestMark)**

**highestMark = mark;**

will keep track of the highest mark entered. If the mark just entered is greater than the current highest mark, then we reset the variable highestMark to store this newly entered mark. Take your time to figure this mini-algorithm out

• Next the value of totalMarks is updated. This is necessary so that we can eventually calculate the average of the marks.

• Finally within the loop, the value of **counter** is incremented. This variable is necessary on two fronts. We need it so that we can calculate the average mark and also it is used within the prompt.

• Once the loop finished the println() call displays the pertinent information. Notice how **type-casting** is being used when calculating the average mark. A **runtime error** would actually occur in this case if the type-casting were missing. Can you see why?

**Typing in and Saving Code for the Program Just Analysed**

Click the **New File** icon on the JCreator IDE and save the file as **DoWhile.java** in your Lab8 folder. Now type in the code for the program above.

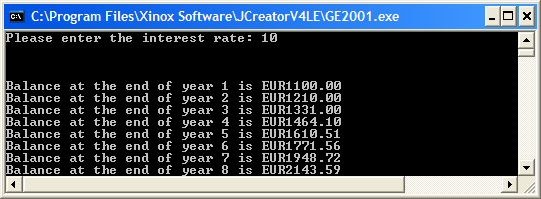
If your program has any errors or warnings, have a look at the edit window and check to ensure that the code is exactly as indicated earlier, including all **semicolons** (**;**) and concatenation operators (+) and ensuring that letters are written in lowercase where indicated. If you spot any differences correct them and compile again until the program is syntax error-free.

Once you are free from errors, run the program. **Test** it out with several different sets of input values.

The example we just looked at shows how a do-while can be used in data-sentinel controlled scenarios. In a similar way, they can be used for **counter-controlled** and **task-controlled** situations. The use of do-while over while is a matter of personal preference really. Most programmers tend to use one of these structures predominantly in their coding (usually while), but it is **vital that you can recognize and use both**, especially if you are in a situation where you are part of a team where there are several developers involved in a given project. while is considered the **primary** repetition control structure in Java however.

**Exercise 4**

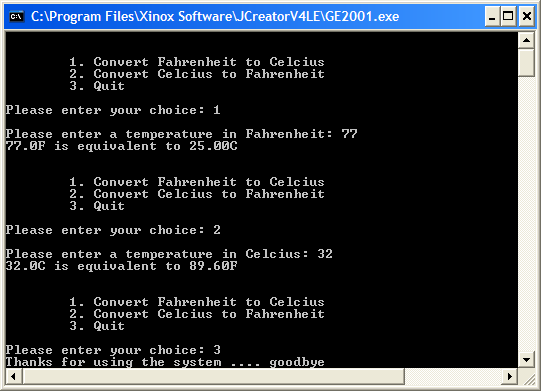
Take the **TaskControlledWhile2.java** program from earlier and save it as **Exercise4.java**. Now modify the original code so that it will now use a **do-while** loop rather than a while loop. The program should produce exactly the same output as before as follows:



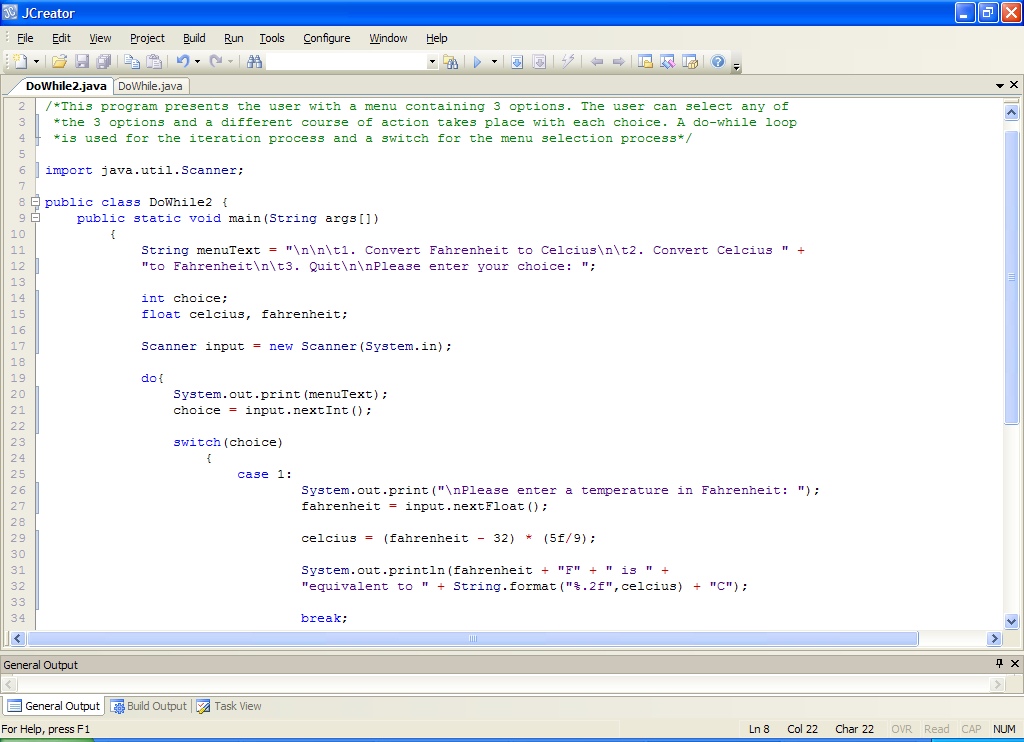
**Another Program That Uses a do-while Loop**

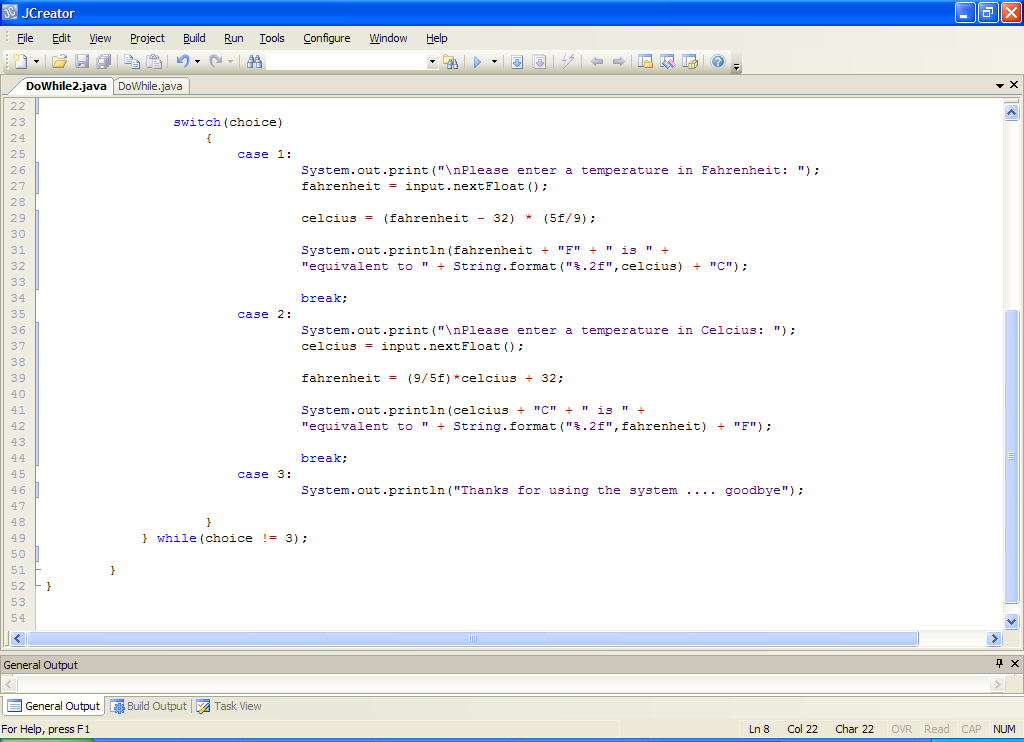
One of the most common uses for a do-while loop is with **menu-type programs** so we look at an example now.

**Aim**: To write a menu-type program which will give the user 3 choices as follows: **convert** **degrees Fahrenheit to Celsius, convert degrees Celsius to Fahrenheit** and to **Quit the system**. A do-while loop and switch statement will be used to achieve this. The program would run as indicated in the following sample screenshot:



**Java Code**:





**Program Analysis**:

• The program starts off by placing the text for the menu options and the prompt for the users choice into the String variable menuText. It can **aid the readability** of the program by splitting it up in this manner, rather than lumping everything in together within the print() method call.

• The do-while loop begins by displaying the menu options to the user and prompting for their choice. Once this choice has been entered it is stored in choice.

• Next up is the switch statement. Recall that switch can only work with variables/expressions that evaluate to an integer type (including char) so note that it would have been impossible here for us to have the choice variable declared as a String as the switch would throw up a **syntax error**. The switch has 3 cases, the first of these deals with conversions from Fahrenheit to Celcius. If the user selects choice 1, then this case executes and the user is prompted for a temperature in Fahrenheit. The temperature is then converted to a Celcius equivalent and this is then displayed to 2 decimal places. Note the **break** statement at the end of the first 2 cases. Hopefully you can recall their purpose.

• case 2 works in an almost identical manner to case 1, except the conversion is from Celcius to Fahrenheit and the associated formula is different.

• case 3 is the quit case and just gives the user a “goodbye” message.

• At the very end of the do-while loop is its test condition. In this case, the do-while loop will terminate as soon as the user enters the value 3 as their choice. Otherwise, it keeps going.

**Typing in Code for the Program Just Analysed**

Click the **New File** icon on the JCreator IDE and save the file as **DoWhile2.java** in your Lab8 folder. Now, for practice, type in the code for the program above.

If your program has any errors or warnings, have a look at the edit window and check to ensure that the code is exactly as indicated earlier, including all **semicolons** (**;**) and concatenation operators (+) and ensuring that letters are written in lowercase where indicated. If you spot any differences correct them and compile again until the program is syntax error-free.

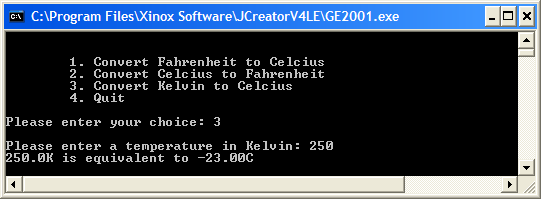
Once you are free from errors, run the program and **test it fully**. You should see that certain Celcius and Fahrenheit values cause problems but that choice is okay. What happens when you enter an **invalid choice** value such as 5 or -8?

**Exercise 5**

Another temperature scale exists - the Kelvin scale. Save the program above as **Exercise5.java** and add an extra option to the program which allows **conversion from degrees Kelvin to degrees Celsius**.

**NB**: **degrees Celcius = degrees Kelvin – 273**

The program should operate as indicated in the sample screenshot below:



**Exercise 6**

Now save Exercise5.java as **Exercise6.java** and **modify** it so that the program is capable of performing some **basic input validation** on some of the values inputted. The rules for input validation here are as follows:

* The choice value must be between 1 and 3 inclusive. If the user enters a value outside this range, they should get a warning message and be continually asked to re-enter until they supply a valid choice
* The Celcius value must not be less than -273C. If the user enters a value below this, they should get a warning message and be continually asked to re-enter until they supply a valid Celcius temperature.

You should use **data-sentinel controlled while loops** for each of the validation loops here.

The program would now run as indicated in the sample screenshot below:

