**Institute of Technology Tralee**

**Computing Department**

**Introduction to Programming**

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

**Practical 5 – Logical Operators & Switch**

Last time we got started on decision-making within programs with the **if,if-else,nested if and nested else-if** statements. We also made plenty use of relational operators to compare values in our test expressions. This time we continue the theme of decision-making but, as well as the if statement, we will also examine another Java selection control structure called the **switch** statement. First off, we will examine the use of **logical operators**.

**The Java Logical Operators**

In Java, the decision-making process often makes use of **more than one condition** within an overall expression. The **logical operators** can be used for **combining various subexpressions to form an overall larger expression**. These operators are

&& *expression1* **&&** *expression2* is true only if **both** *expression1* **and** *expression2* are true

|| *expression1* **||** *expression2* is true if **either or both** *expression1* **or** *expression2* is true

! **!** *expression1* is true if *expression1* is currently false and vice versa.

**Some Quick Examples of Using Logical Operators**

5>2 && 4>7 is **false** because the subexpression 4>7 is false.

5>2 || 4>7 is **true** because at least one of the subexpressions, 5>2, is true

! (4>7) is **true** because the expression 4>7 is false

**N.B.** ! (4>7) is equivalent to **4<=7**

**Avoiding Nested if Statements by using &&**

Consider the following code:

if (age >= 13)

if (age <=19)

System.out.println("You must be a teenager - unless you were lying about your age :-)");

Essentially this segment of code tests to see whether or not a person’s age is between 13 and 19 inclusive (i.e. that they are a teenager). If the first if clause is true **and** the second if clause is also true then the message will be output as required.

Note that I have highlighted the word “**and”** above.

Now, rather than using 2 if statements as nested above, we could use the **&&** operator as follows instead:

if (age >= 13 && age <= 19)

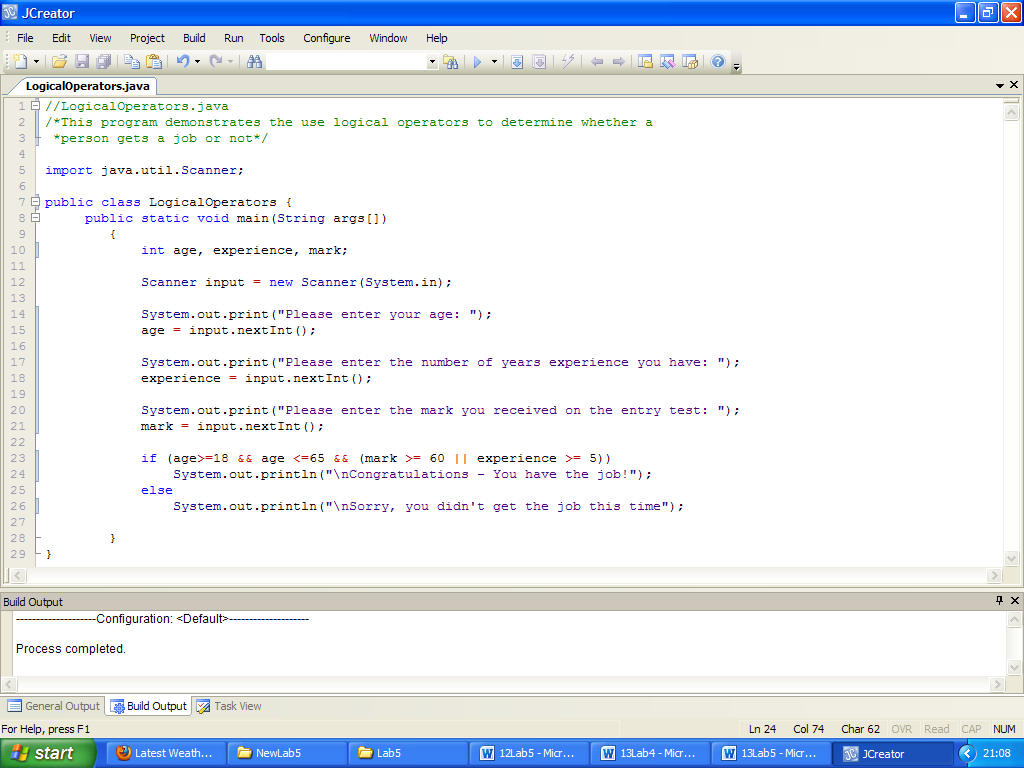
System.out.println("You must be a teenager - unless you were lying about your age :-)");

This is much **more readable** and **more efficient** to use than the previous code and is what you should use in the future where appropriate.

Next we will look at a program that illustrates the use of some of the logical operators.

**Aim:** The purpose of the following program is to illustrate the use of some logical operators

**Java code:**



**Analysis of program:**

• The user is asked to enter their age, their number of years’ experience and the mark they received on an entry test.

• The next section of the program contains an if-else statement. As long as the user enters an age between 18 and 65 inclusive and they enter either a mark of at least 60 or a years’ experience value of at least 5, then the expression will evaluate to true and the “Congratulations” message will be displayed. Otherwise, the expression will evaluate to false and they will receive the “sorry” message. Note how the **logical operators are used to combine subexpressions to form a larger expression**.

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

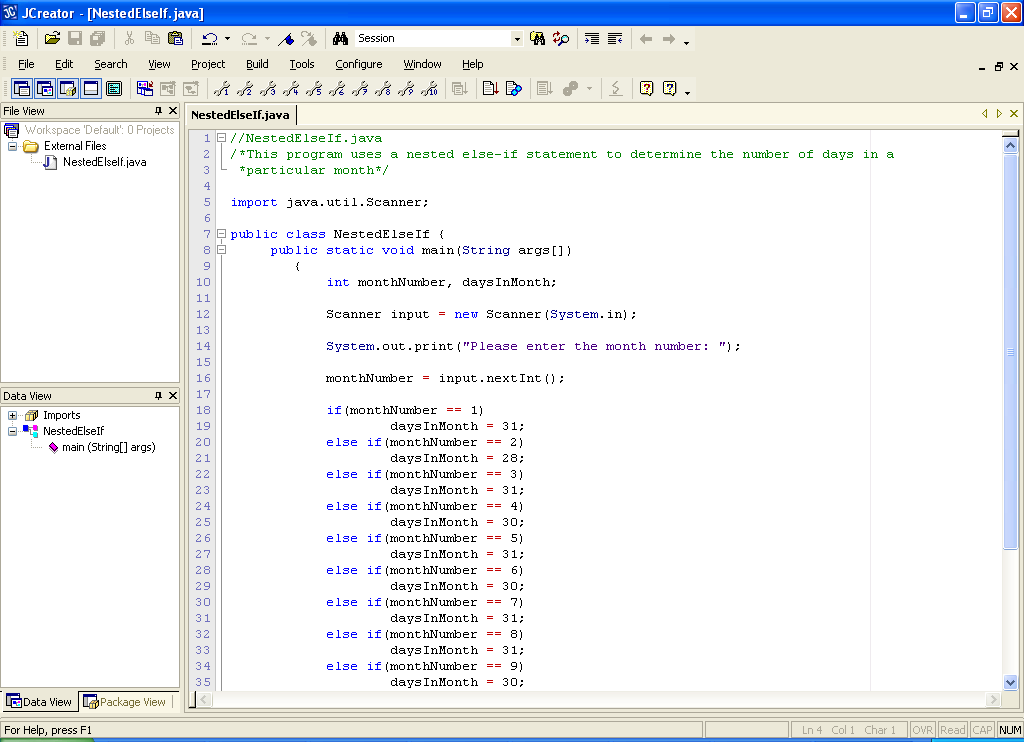
This week you should save all of your work to a new folder called **Lab5** within the **JavaStuff**  folder on your X: drive. Click the **New File** icon on the JCreator IDE , save the file as **LogicalOperators.java** and type in the code above.

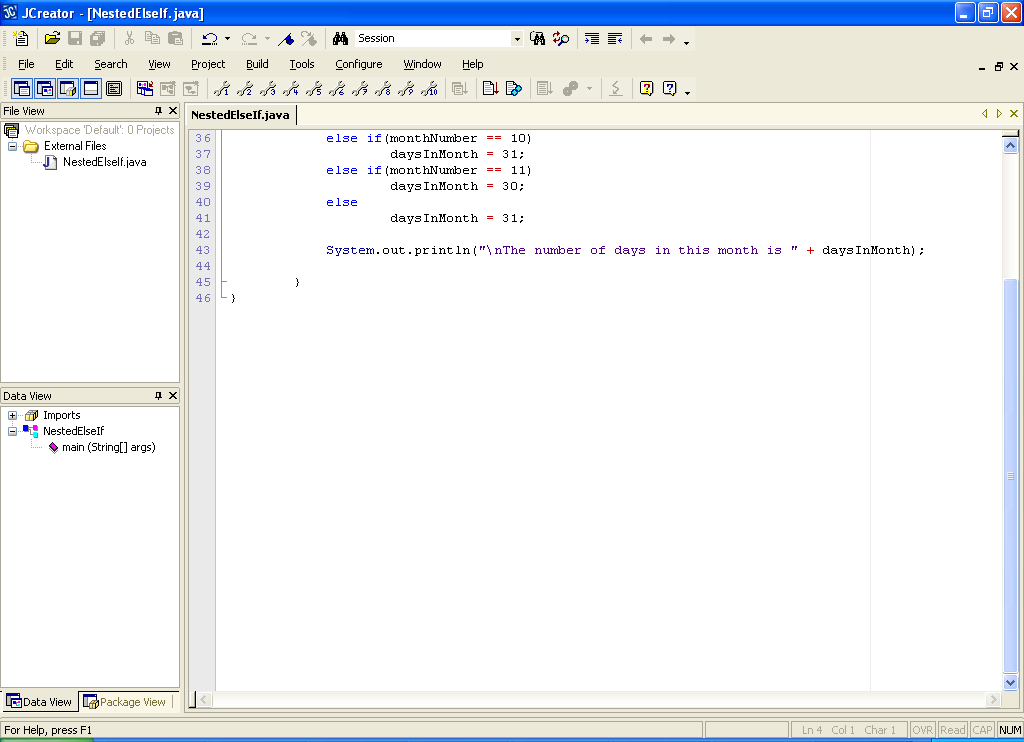
If your program has any errors or warnings, have a look at the edit window and check to ensure that the code is exactly as indicated earlier, including all **semicolons** (**;**). If you spot any differences correct them and compile again until the program is syntax error-free.

Once you are free from errors run the program with **several different sets of input** values. Try to ensure you **test your program fully**.

**Improving a Program From Last Week**

Recall from last week’s lab sheet, we had a program that asked the user to supply a month number and it would then display the number of days in that month. It looked as follows:





This program worked fine, but it can be improved upon in a number of ways. You see that in the program there are several month numbers that have 30 days and several that have 31 days. The message being displayed for all these month numbers is the same and so we can take advantage of **logical operators** to combine the expressions as appropriate.

Also, there is **no validation** for the month number. We would like to set up the program so that, if the user supplies an invalid month value, the program issues an appropriate warning message and terminates immediately. The most efficient way to do this here is to have the following – this is a **pseudocode** outline of the solution:

1. Prompt for the month number
2. Read in the month number
3. if (monthNumber < 1 or monthNumber >12)

3.1Display an “Invalid month” message

1. else //if we get here we know the month number is definitely valid

4.1 if (monthNumber is 1 or 3 or 5 or 7 or 8 or 10 or 12)

4.1.1 Set daysInMonth to 31

4.2 else if (monthNumber is 4 or 6 or 9 or 11)

4.2.1 Set daysInMonth to 30

4.3 else

4.3.1 Set daysInMonth to 28

4.4 Display the number of days in the month

You should now save the program as **NestedElseIfImproved.java** and make the necessary modifications indicated above. Your program should still work in the same way as before, except for the warning message associated with invalid input.

**Operator Precedence**

You may remember that all operators have a certain “**importance**” associated with them and expressions end up getting evaluated in a particular order based on this “importance” or precedence.

We’ve already looked at the operator precedence table for the arithmetic operators. Relational and logical operators are no different in that they also have a certain precedence level associated with them. In Java, the **relational operators have higher precedence than the logical operators (except for the ! operator)**.

Our precedence table now looks as follows:

**Operator Name Symbol**

parentheses ()

logical NOT !

multiplication,division,remainder \* / %

addition,subtraction + -

relational operators > < >= <=

relational operators = = !=

logical AND &&

logical OR ||

assignment =

**Overriding Precedence Rules**

Recall that the order in which an expression is evaluated can always be overridden through the use of parentheses. In the program **LogicalOperators.java** this was actually done. If the parentheses around the subexpressions

(mark >= 60 || experience >= 5)

had been omitted, a **logical error** would have occurred. In this case, the “congratulations” message would appear as long as the years’ experience was at least 5, irrespective of the age of the user. So, a person who entered an age of 8 would be offered the post as long as the years’ experience value was at least 5, which is logically incorrect in this case.

You should now **modify** the **LogicalOperators.java** program from earlier and remove the parentheses to see this effect. Recompile and run the program. It will hopefully impress on you the importance of parentheses when forming expressions like this. These types of logical errors can be toughest ones to detect and fix in practice.

**Exercise 1**

In order to join the Gardaí, a citizen must satisfy the following criteria (totally made up by myself!)

Be at least 18 years of age and under 35 years of age

Be at least 1.85 metres in height if male and 1.6 metres if female

Be less than 100 kilos in weight

Have no criminal record

and either

Have received at least a D grade in pass Irish in the Leaving Cert

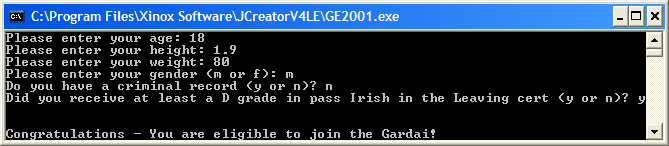
Or

Commit to taking a 10-week Irish course on application

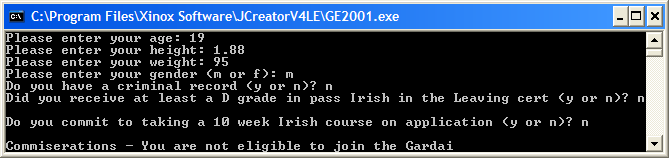
If the conditions above are satisfied, then the citizen is eligible to join the Gardaí, otherwise they are not.

You must write a program that works as indicated in the sample screenshots below:

Run 1:



Run 2:



Note that these screenshots are only a fraction of the total different execution paths this program could take – you should **test out your program fully** so that you are confident it will run under all conditions.

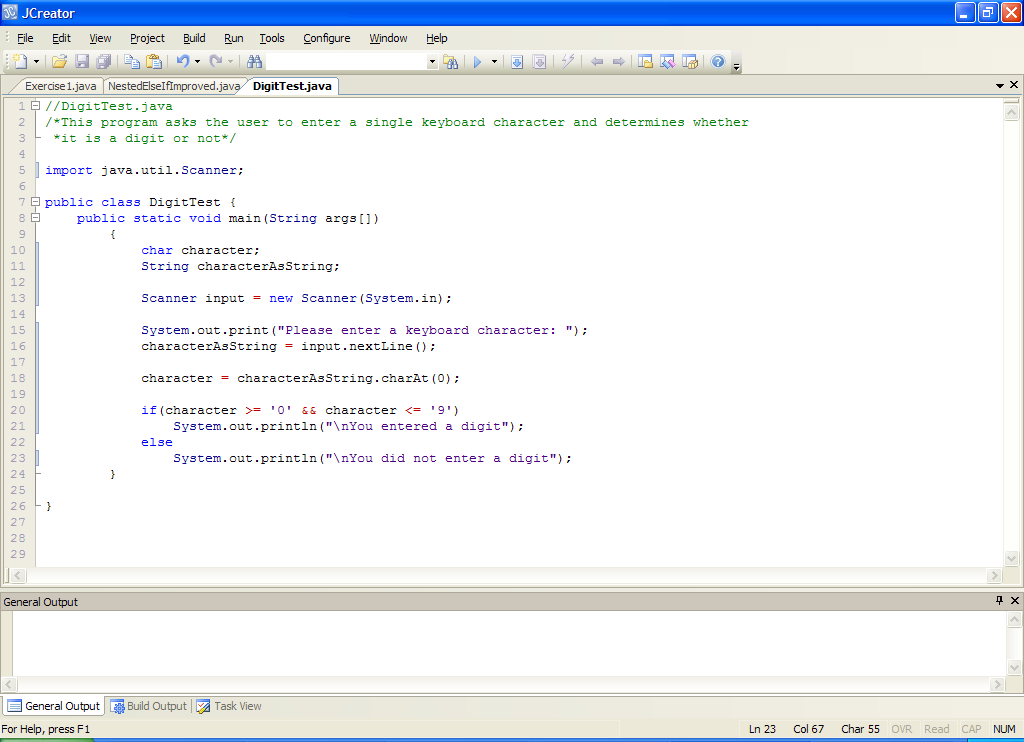
**Note also that the question relating to taking the Irish course is only asked if the user answers “n” to the leaving cert Irish question.**

**Comparing Characters in Java and the ASCII Table**

There is often a requirement to compare characters, especially for validation routines. We may need to know the nature of the character i.e. is it a letter, a digit, a punctuation symbol etc.

**Aim**: To write a program which will take a user supplied character and determine whether it is a digit or not. It will output an appropriate message in either case.

**Java Code**



**Analysis of Program**:

● When the user has entered their input, it is truncated to the first character using the charAt(0) method. This converts it to a character value also.

● Next, the if-else statement determines whether the character is a digit.

If it was a digit, then both of the conditions

**if(character >= '0' && character <= '9')**

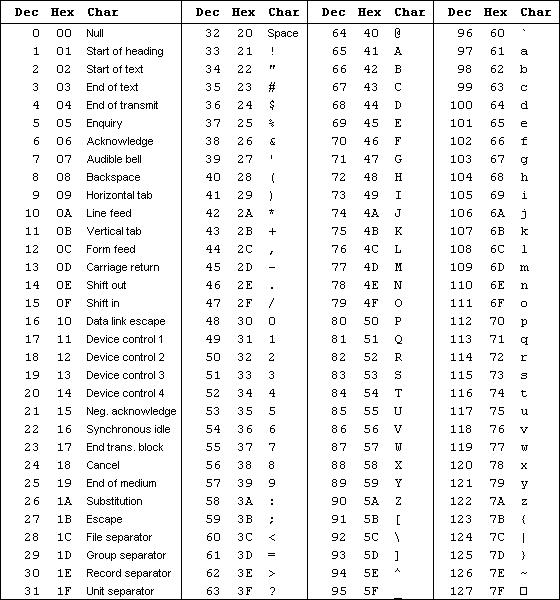
must be true. We are basically testing to see if the character input by the user lies within the range of **ASCII values** for the digits 0 to 9. If the ASCII number of the value inputted does lie within this range, the character entered must have been a digit and we give out that message.

The ASCII table itself is as follows and you can see that the digits 0 to 9 are **all together in the table** (our expression in the program totally depends on this). You can tell by looking at this table that the ASCII value of the character ‘0’ is 48 and that for ‘9’ is 57. Can you figure out how I am coming up with these values? Naturally, you are not expected to memorize these values – you don’t need to as a programmer since the codes are automatically available when performing comparisons as in the case above. However, if you did need to display the ASCII code of a character, it can be done easily as follows:

**System.out.println("ASCII code of 'A' is " + (int)'A');**

Note that this is done by simply **type-casting** the character constant ‘A’ as an integer. This will display the value 65 as its ASCII code which is correct – prove this by examining the ASCII table below.

Note in the table that the **lowercase letters are separated from the uppercase letters** by a few odd-ball characters.



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

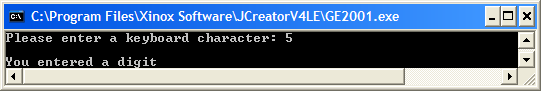
Click the **New File** icon on the JCreator IDE and save the file as **DigitTest.java** in your Lab5 folder. Now, for practice, type in the code for the program above.

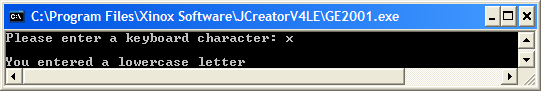
If your program has any errors or warnings, have a look at the edit window and check to ensure that the code is exactly as indicated earlier. If you spot any differences correct them and compile again until the program is syntax error-free.

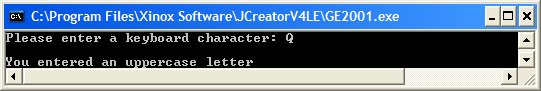
Once you are free from errors, run the program. Make sure that you **test** it with several different sets of input values.

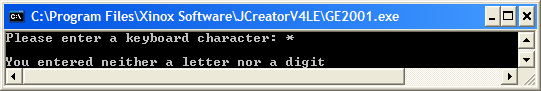
**Exercise 2**

Write a program which expands on the previous program to take a user-supplied character and determine whether the value is an uppercase letter, a lowercase letter, a digit or just “neither a letter or a digit”. Output appropriate messages in each of the four cases. You should save your program as **Exercise2.java** and you should use a **nested else-if** in the coding of the program. Your program should run as indicated in the following sample screenshots (these are all separate runs of the program):









**The switch Statement**

The **switch statement is an alternative to using a multi-way nested else-if** statement. It is never actually essential to use a switch statement when programming but it **executes more efficiently** than an equivalent nested else-if statement so it is useful to take advantage of it when we can.

In its general form the switch statement is as follows:

**switch (***expression***)**

{

**case** *value***:**

*statements*

**case** *value* **:**

*statements*

**:**

**:**

**case** *value* **:**

*statements*

**default :**

*statements*

}

Here everything in italics is only a placeholder for either a Java *expression*, a Java *statement* or a **unique** case *value*. Everything in bold such as the words **switch** and **case** will be as they appear above, all words in bold here are Java **keywords**. Notice that the *statements* part of the switch statement is normally **indented** (tabbed) in order to make the code more **readable** - this is a **style feature** in Java and is good programming practice. Lack of indentation **will not affect the compilation** or running of your program as mentioned before but will make it look better.

The next question is - how do we interpret the control structure above?

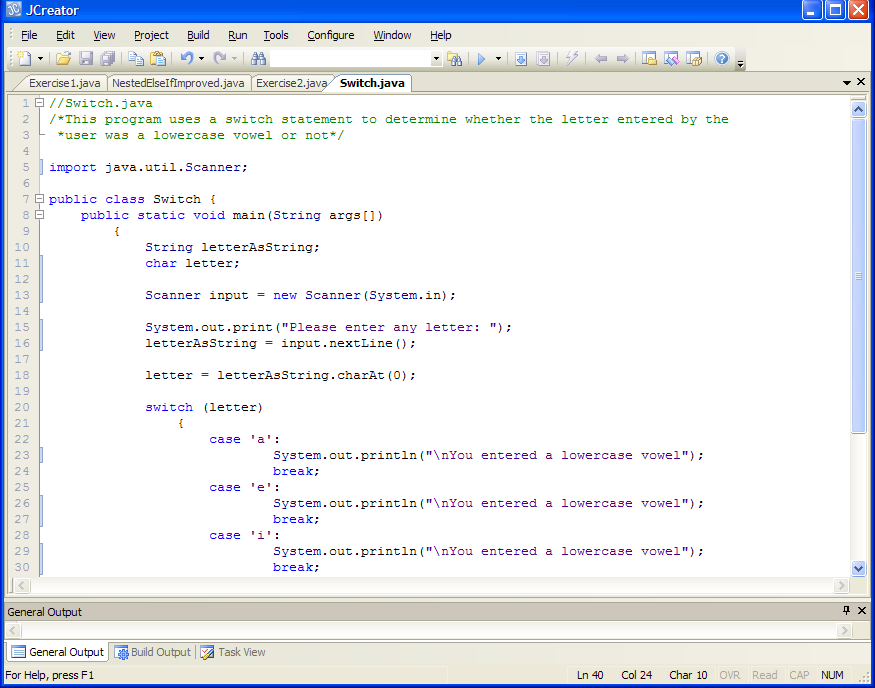
What happens here is that a Java **expression is evaluated and then compared to the very first case *value* in the list**. If a **match occurs then the statements associated with that case *value* are executed**. If a match does not occur **comparison continues with the next case *value*.** If none of the case values match the test expression value then the statements associated with the **default** clause are executed. However, the inclusion of this clause is completely **optional**.

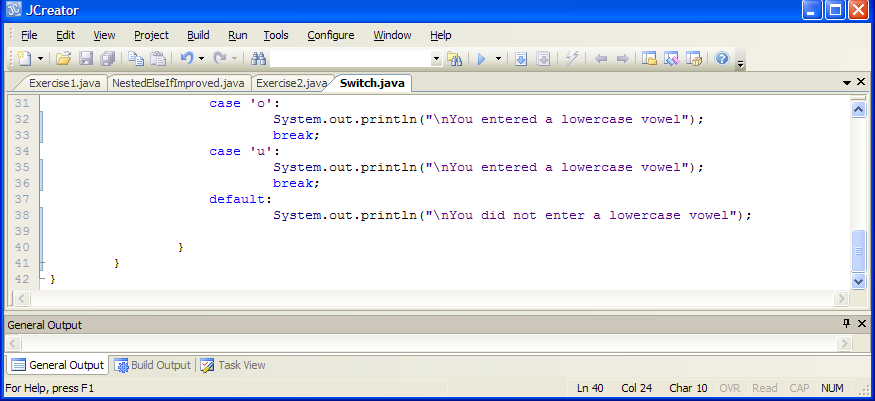
Of **vital importance** for the operation of switch is that the expression evaluates to an integer or single character (char) – **switch can only operate with individual integer, single character or String values** and so is **very limited** in its usefulness. If the test expression does not evaluate to an integer, character or String the compiler generates a **syntax error**.

Note also the **colons** following the case values – these are essential and their omission generates a **syntax error**.

**Aim:** The purpose of the following program is to illustrate the use of a switch statement.

**Java code:**





**Analysis of program:**

• The user is asked to enter a letter. This is then read in and converted to a char using the charAt(0) method call. Note that the effect of this is to “chop off” anything else that the user entered and just return the first character. Strictly speaking we could have gotten away without doing this conversion to char here as the switch would work with the String value directly, but we use char just for practice.

• The switch expression here is simply the char variable letter. This is compared with the first case value , which is the character constant ‘a’. If they match, then the user gets the “You entered a lowercase vowel” message.

• The **break** statement is **very important** to the correct functioning of this program. The break ensures that once a particular case value executes its statements, then all the other case values will be bypassed. Without it, **logical errors** would creep in here. For example, if we omitted the break at the end of **case ‘a’:** then, if the user were to enter an ‘a’, they would see that the println() method associated with case ‘a’ would execute but execution would then “roll on down” to case ‘e’ also and execute its println() method, before eventually breaking.

• If the user enters a character which does not match any of the case values then the **default** case will execute, giving the user the “did not enter” message.

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

Click the **New File** icon on the JCreator IDE and save the file as **Switch.java** in your Lab5 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. Make sure that you test it with several different input values such as A, 3, %, u, O, apple

**Improving the Last Program**

The last program can be improved upon. You see that the message being displayed for all the lowercase vowel values is the same. In this situation we can **combine a number of case values** together as follows:

switch (letter)

{

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

System.out.println("\nYou entered a lowercase vowel"); break;

default:

System.out.println("\nYou did not enter a lowercase vowel");

}

There is no break at the end of the first 4 cases and so, if the user enters any of these values, it will just “roll on down” until **case ‘u’:** is met. This then displays the message as required.

You should now save the program as **SwitchImproved.java** and make the necessary modifications indicated above. Your program should still work in exactly the same way as before.

**Exercise 3**

Write a Java program that presents the user with a menu of options as follows:

1. Calculate the area of a circle
2. Calculate the volume of a sphere
3. Quit

When the menu appears the user will be asked to enter their preferred choice as a number (1-3) and the program will proceed to carry out the appropriate action. In executing choices 1 and 2 the user will be prompted for the radius of the circle or sphere and this will then be used to determine the area or volume as required. All results from the program should be displayed to **3 decimal places**. If the user selects choice 3, then they should receive a farewell message. In the event that the user enters a value outside the 1-3 range then an appropriate warning message should be issued. You should use a **switch statement** for coding the solution here.

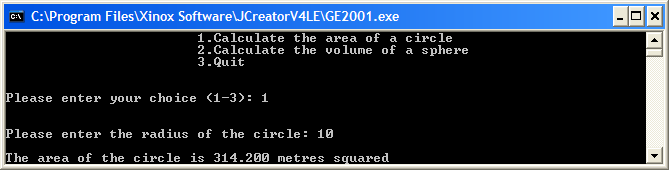
**N.B.** The formulae to be used here are as follows:

area of circle = πr2

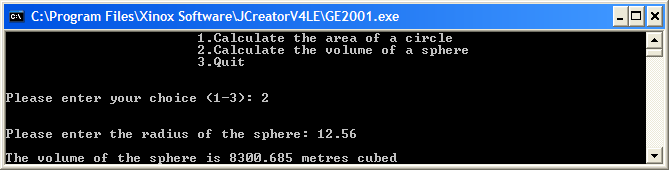
volume of sphere = 

π=3.142 and should be declared as a **constant** in your program

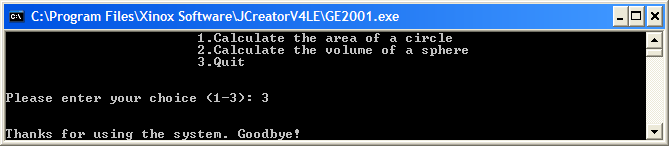
Your program should run as indicated in the following sample screenshots:

Run 1: 

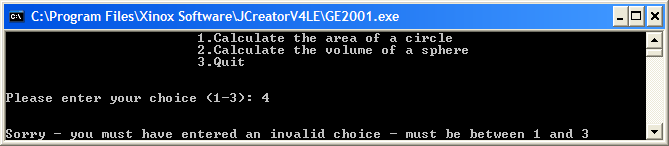
Run 2:



Run 3:



Run 4:



**Exercise 4**

Take the **NestedElseIfImproved.java** program you wrote earlier and save it as **Exercise4.java**. Now modify the code so that it uses a **switch** statement for its decision-making rather than a nested else-if. The new program should run exactly as the original one did. One sample run of the program is as follows:

