**Institute of Technology Tralee**

**Computing Department**

**Introduction to Programming**

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

**Practical 4 – Decision Making with if**

In this lab sheet we move on to another essential aspect of computer programming – the ability for your programs to make decisions. This is achieved through the **selection control structures** and the **relational** and **logical** **operators**.

**The Java Relational Operators**

In order to make full use of decision-making structures we must be familiar with the **relational operators**.

In Java the decision-making process generally makes use of one or more relational operators for **comparing different variables and values**. These operators are

= = equal

< less than

> greater than

<= less than or equal to

>= greater than or equal to

!= not equal

So using these operators we can perform various tests in our programs e.g. is the age of the person over 66? Is the height of the person below 1.8 metres? Is the price of the computer at least €500 etc.

However in order to make full use of the relational operators we must be able to write **selection control structures**.

**Sequence and Selection Control Structures**

All of the programs you have seen so far have involved simple **sequences of Java statements**, with **each** **statement being executed after the one immediately preceding it**. There is absolutely nothing wrong with this, of course, and as you have seen, a multitude of programs can be written using only **sequence control structures**. That said, however, there are even more programs which **cannot be written** if we were to rely on this type of control structure alone.

Hence, we shall now investigate **selection control structures** which **allow Java programs to make decisions** by **executing different portions of code depending on the outcome of some test condition.**

**The Simple if Statement**

The **primary selection control structure** in Java is the if statement.

The **general form** of the simple if statement is as follows:

**if** (*expression***)**

{

*statements*

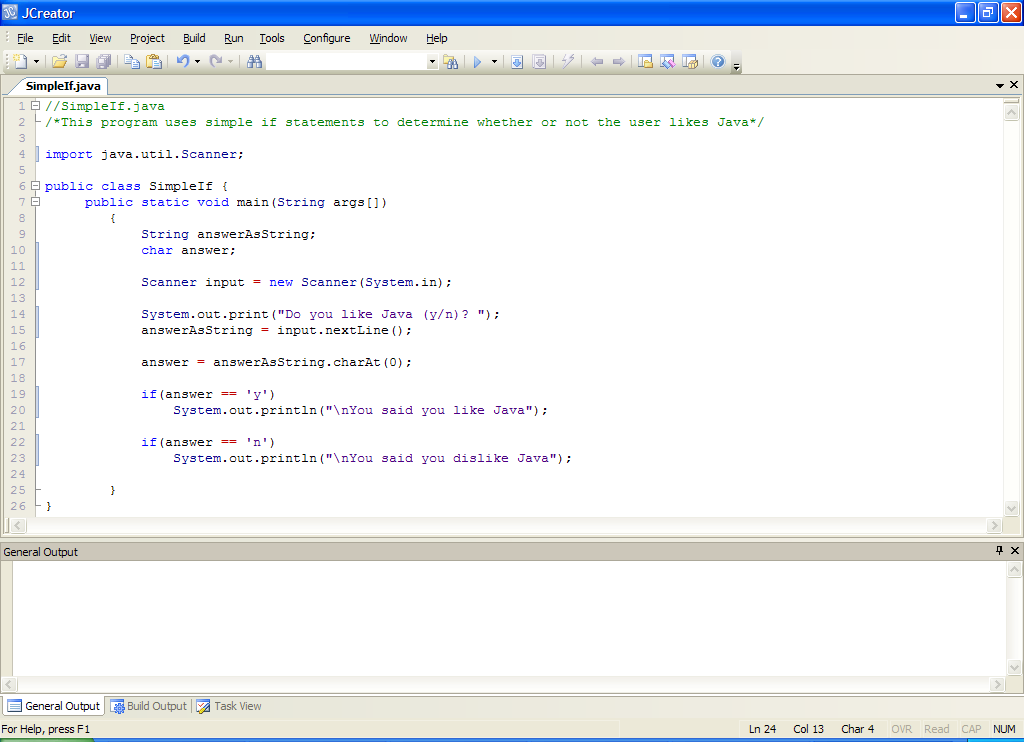
}

What happens here is that the test expression is evaluated. If the **test expression** happens to evaluate to **true**, then the **statements** associated with the if section **execute**. However, if the **test expression** evaluates to **false**, then the **statements** are **bypassed**.

if is a **keyword** as you might have expected. The **test** **expression must be enclosed within parentheses**. Also, if there is more than one statement to be associated with the if, then **curly braces** must surround the entire block of statements. Violation of either rule will generate a **syntax error**. Also note that the statements are usually indented as indicated above – this is a **style feature**, aimed at making the code easier to read. It is highly recommended that you follow this style, but the compiler will not complain if you don’t!

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

**Java code:**



**Analysis of program**

• The user is prompted with a question and is expected to provide a y or n in reply to it.

• The users response is read in using the **nextLine**() method and stored in answerAsString. Then the code

answer = answerAsString.charAt(0);

picks out the very first character in the user’s response and stores it in the char variable answer. We have not really examined the char data type until now but it is useful to know about. A char variable is **capable of holding only a single keyboard character**.

The **charAt**() method of the String class is being used here to extract the very first character from the users input. The first character is accessed through the number 0. In general the method returns any character you like from a String e.g.

String name = “Joe Bloggs”;

name.charAt(4) would be ‘B’ and name.charAt(7) would be ‘g’. Note that the index of the very first character is 0 which is **unexpected**.

The advantage of the charAt() method here is that, if the user types in something like “yes” or “yep” or anything beginning with ‘y’ then it will be effectively converted to ‘y’ anyhow by charAt(0).

We will use the **charAt**() method very often in this module, as there is a general need to be able to extract characters from a String value.

• The simple if statements come next. The first one compares the value stored in the variable answer with the character ‘y’ and, if they match, the statement associated with this if will execute giving the user the “You said you like Java” message. If they typed something that did not begin with a lowercase ‘y’ then it will just bypass the statement.

• The second simple if operates in an identical manner to the first one, except this one compares the value stored in answer with ‘n’. If they match then the user gets the “dislike” message. If they typed something that did not begin with lowercase ‘n’ then it will bypass this statement.

Notice that in Java, **character constants** such as ‘y’ and ‘n’ must be enclosed in **single-quotes**. To omit these would constitute a syntax error.

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

Create a new folder for this week’s work called **Lab4**. At this stage everything is set up for you to write your first Java program which uses selection control structures. Click the **New File** icon on the JCreator IDE and save the file as **SimpleIf.java** in your Lab4 folder. Now type in the program 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** (**;**) 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**. In particular, try out some values that don’t begin with either ‘y’ or ‘n’. Hopefully the output is what you expect.

**The if-else Statement**

With the simple if statement either something happens or nothing happens for a given test expression. When there is a **2-way decision** to be made, an if-else statement can be used.

Its general form is:

**if** (*expression*)

{

*statements*

}

**else**

{

*statements*

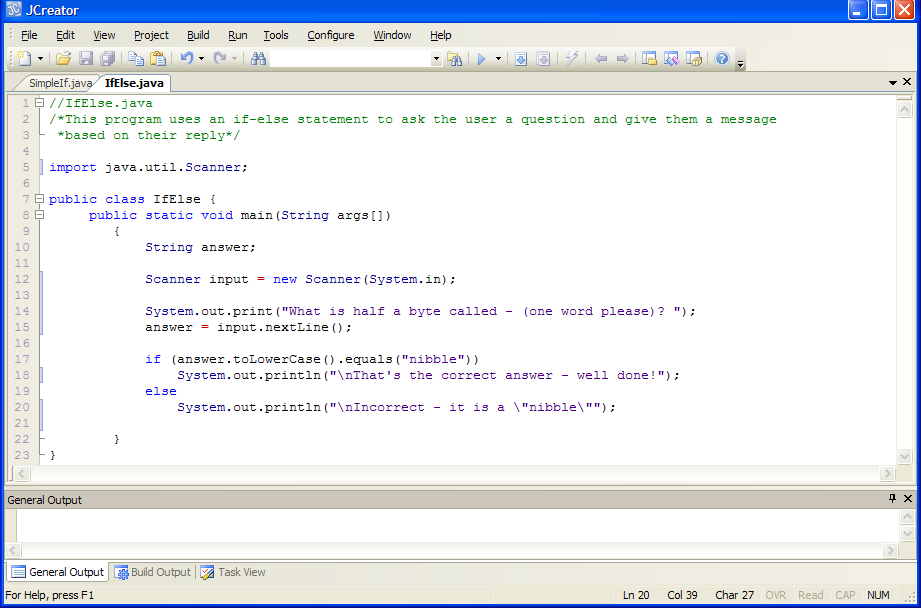
}

What happens here is that a Java test expression is evaluated and **if** the **expression** evaluates to **true**, the **statements associated with the if section will be executed** and the **statements associated with the else section will be bypassed**. However, if the **expression** evaluates to **false** then the **statements** **associated with the if section will be bypassed** and instead the **statements associated with the else section will be executed**.

As you might have guessed, **else** is a **keyword** of the Java language.

**Aim:** The purpose of the following program is to illustrate the use of an if-else statement.

**Java code:**



**Analysis of program:**

• The user is asked a question and their reply is stored in the String variable **answer**.

• The next section of the program contains the if-else statement. The code

answer.toLowerCase().equals("nibble")

first converts the answer entered by the user to lowercase form using the **toLowerCase**() method. Then this lowercase version of the answer is compared with the word “nibble”. As long as they match, the statement associated with the if executes and the user gets the “correct answer” message as required. The use of toLowerCase() here means that the user can enter the answer in any form e.g. “Nibble”, “NIBBLE” etc. and it will be converted to the lowercase equivalent for comparison with “nibble”. Essentially is makes the answer **case-insensitive**.

However, if the user enters anything other than the word “nibble” then the statement associated with the if gets bypassed and instead the else section executes and the user gets the “Incorrect” message.

One important point to note here is that **strings cannot be compared with each other by using the relational operators**, as variables of primitive data types can (remember **String is a class** and not a primitive data type). Here the equals() method is used to compare two strings for equality. If they are equal then the expression evaluates to true, false otherwise.

**equals()** is another method that we will use very often in this module.

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

Click the **New File** icon on the JCreator IDE and save the file as **IfElse.java** in your Lab4 folder. Now type in the code for the program above.

If your program has any errors or warnings, keep an eagle-eye out for any discrepancies and try to fix them.

Once you are free from errors, run the program. Make sure that you test it with several different input values. In particular in this case, try out the word “Nibble” with a capital “N” to see how it runs. Is it what you expect?

**Improving Program Efficiency through if-else statements**

Recall in the SimpleIf.java program we had the following code:

if(answer == 'y')

System.out.println("\nYou said you like Java");

if(answer == 'n')

System.out.println("\nYou said you dislike Java");

Assuming the user enters only a ‘y’ or an ‘n’ when this program runs, there are **two mutually exclusive outcomes** here – the user will either get the “like” message or the “dislike” message.

It is inefficient in the sense that 2 tests must be performed when the program runs, even if the first test evaluates to true (which would implicitly mean the second test would evaluate to false).

We can take advantage of the **if-else** statement here to improve the efficiency of the code. So instead we could write the code as:

if(answer == 'y')

System.out.println("\nYou said you like Java");

else

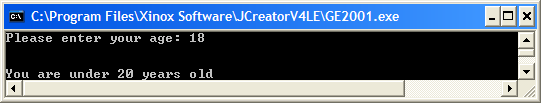
System.out.println("\nYou said you dislike Java");

Now, should the user enter ‘y’, it will call the first println() above but completely bypass the else section, saving us the extra test in the original version.

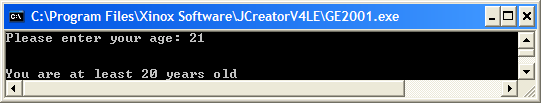
**Exercise 1**

Write a Java program that asks the user to enter their age (a whole number). It should then use an if-else statement to determine whether or not the user is at least 20 years old, and display the outcome. The program would run as indicated in the following sample runs:

Run 1:



Run 2:



**Exercise 2**

Under the new taxation system employed by the Irish Government, an employee’s owed tax (PAYE) for a given months work can be calculated using the following formula:

PAYE = [20% of Cut-off Point] + [42% of (Gross Income – Cut-Off Point – Pension Payments)] – [Tax Credits]

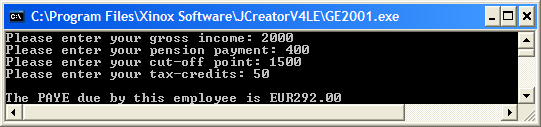
The formula above applies if the employee’s gross income exceeds the cut-off point value. If, however, this is not the case then the following formula is used instead:

PAYE = [20% of (Gross Income – Pension Payments)] – [Tax Credits]

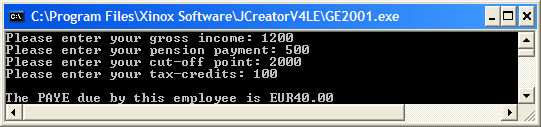
Write a Java program that asks the user to enter their *gross income* for the month, their *cut-off point* (this is just a certain amount of Euro), their *pension payments* and their *tax-credits* (again just a certain amount of Euro) and uses the formulae above to determine the correct PAYE owed by that employee for their month’s work. Make sure also that your program creates the rates 20% and 42% as **constants** in the program.

Your program should run as indicated in the following sample screenshots and display the PAYE due by the employee to **2 decimal places**.

Run 1:



Run 2:

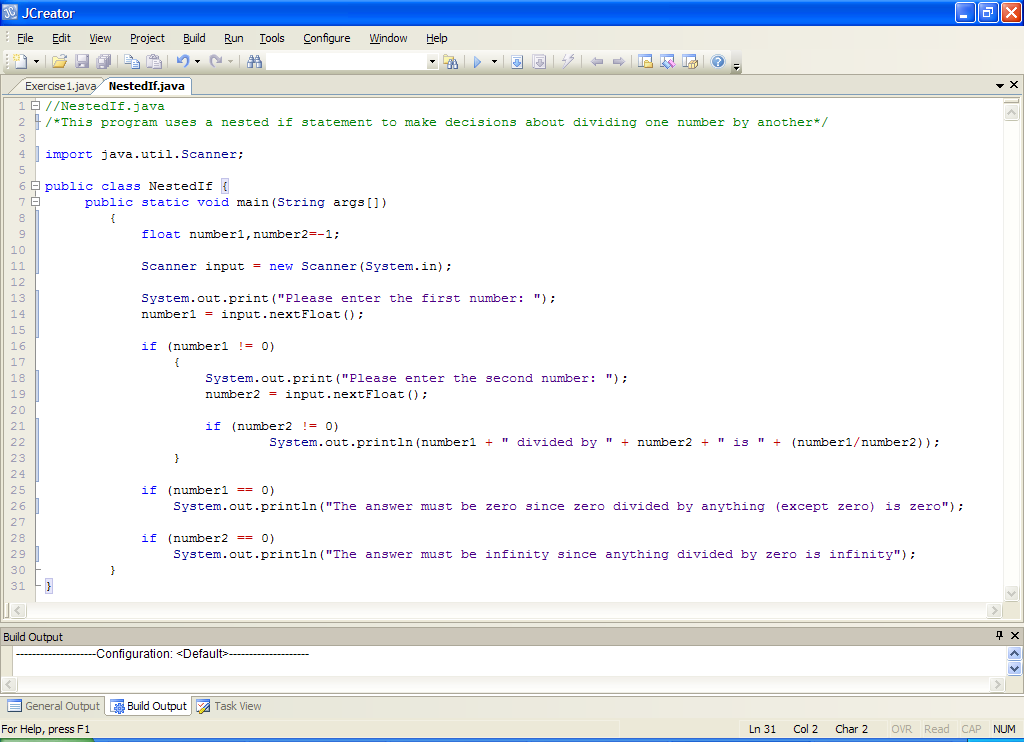


**Nested if Statements**

Sometimes there is a need to perform a certain test and then, based on the results of that test, perform another test and so on. This is possible through the use of **nested if statements**.

**Aim**: The purpose of this program is to read two numbers from the user and divide the first by the second, but only under certain circumstances – it uses a **nested if** structure.

**Java Code**:



**Analysis of program**

• The user is prompted for a first number and this is read in using nextFloat()

• The next part contains the **nested if** structure. The first number entered is compared with zero and, as long as the number is not zero, the statements associated with the if execute. This means that the user is prompted for a second number, which again is read in and converted. Another if statement occurs next to compare the second number with zero. Again, as long as the value of the second number is not zero, the statement associated with this if will execute. In this case the statement is simply to display to the output window the result of dividing number1 by number2.

• Following the nested if there are 2 simple if statements. The first of these checks to see whether the value of number1 is zero. If it is then it displays a message indicating that zero divided by anything is zero. The second simple if checks to see whether the value of number2 is zero. If it is then it displays a message indicating that anything divided by zero is infinity.

• Note that the simple if statements have **no curly braces** here but the nested if structure does use them. This is because, in Java, if there is only one statement associated with an if section (or else section), the curly braces can be omitted – but it is also possible to use them in this case also. Therefore it is sometimes **better that you always put in the curly braces so that you don’t forget them when they are needed** (which is most of the time). Omission of curly braces when they are needed **normally leads to a syntax error** but **sometimes a logical error**.

• Another important point from the above program is the code:

**float number1,number2=-1;**

notice that number1 is not initialised but number2 is. The reason for this is not obvious at first sight. In fact, you need to look at what is happening later on in the program to figure out why one is initialised and the other is not. The way it works is as follows:

The Java compiler looks at the whole program and sees that a variable called number1 is declared. It also sees that this variable is being used in expressions and also in a display message but it is quite happy (if compilers can be happy!) as it sees in line 14 that number1 will definitely have the chance to receive a value during the programs execution.

Things are not so rosy with number2 though. The compiler again sees that number2 is declared and that it is also used in expressions and in a display message but it sees that number2 only gets a chance to receive a value provided that a certain condition (number1 != 0) is met. The compiler will not allow this kind of behaviour in a program – in short **it is a rule that all variables must have a chance to receive a value before they can be used for processing** anywhere else in the program and the compiler sees that the variable number2 may never receive this chance. This generates a “**variable may not have been initialised**” syntax error, which is one of the more common error messages you are likely to see. To avoid this, number2 needs to be initialised with a particular value. I chose to initialise it to -1 here, but it could have been initialised to any number other than zero (can you figure out why it shouldn’t be initialized to zero?)

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

Click the **New File** icon on the JCreator IDE and save the file as **NestedIf.java** in your Lab4 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 sets** **of input** values. In this case you should test the program as follows:

Run 1: Enter 10 for number 1 and 20 for number 2 => output should be 0.5

Run 2: Enter 0 for number 1 => output should be “zero” message

Run 3: Enter 10 for number 1 and zero for number 2 => output should be “infinity” message

Now you should also carry out the following test - **see what happens when you enter a word such as “one” rather than a number for one of the variables.** Can you explain this?

**Improving the Last Program**

The nested if program functions properly but it could be improved on. In this program there **are conditions that are exact opposites of each other, which can be combined more efficiently by using an if-else** structure.

In the program there are **2 sets of exact opposite conditions**. You should now save the program above as **NestedIfImproved.java** and modify it so that it combines conditions more efficiently using if-else statements (note that there will still be a nested if statement in the modified version). Your program should still run exactly the same as before.

**Exercise 3**

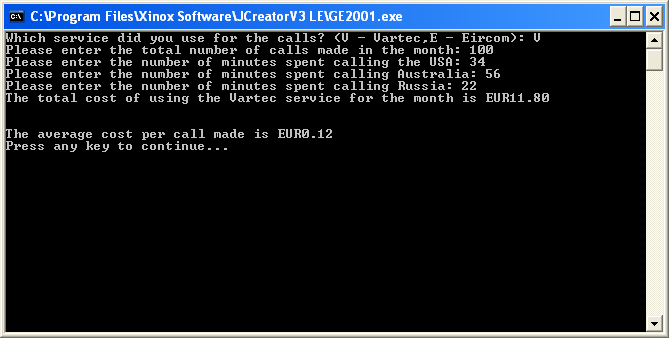
A new telecommunications company called Vartec have recently issued their call rates (per minute) to the general public and in their leaflet they also compare their prices to those of Eircom. Part of that table is as follows:

|  |  |  |
| --- | --- | --- |
| **Call To** | **Vartec Rate** | **Eircom Rate** |
| USA | 6c | 19c |
| Australia | 8c | 85c |
| Russia | 24c | 92c |

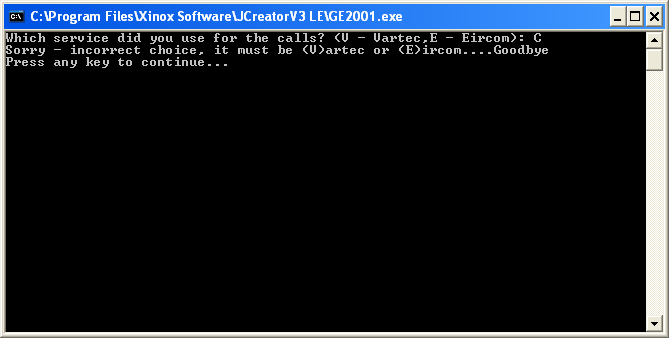
Your task here is to write a program that will first of all read in the **actual service** used (say ‘V’ for Vartec and ‘E’ for Eircom). Next the program should take in the **total number of minutes** the user has spent making calls to **each** of the locations above in the month. Also the **total number of calls** made in the month should be read in so that the *average* *cost* *per call* can be calculated. Both the Vartec and Eircom **call rates** should be coded as **constants** in your program (so you’ll have 6 constants in total). Once the total minutes has been read in for each location, the **total cost** of using the service for the month should be calculated and displayed by the program. The program should then determine and display the **average cost per call** using the chosen service.

Your program should also contain a few “**foolproofing**” bits of code e.g. if the user enters neither ‘E’ nor ‘V’ for the service (use a **char variable** for this) , the program should give an appropriate warning message to the screen and **then immediately terminate**. Also, should the user enter zero for the number of calls made, the program should not then ask for the number of minutes spent calling the various locations and just terminate at this point, thanking the user for their time. Some typical runs of the program are indicated in the screen captures below and your code should aim to emulate these. All costs should be displayed to **2 decimal places**.

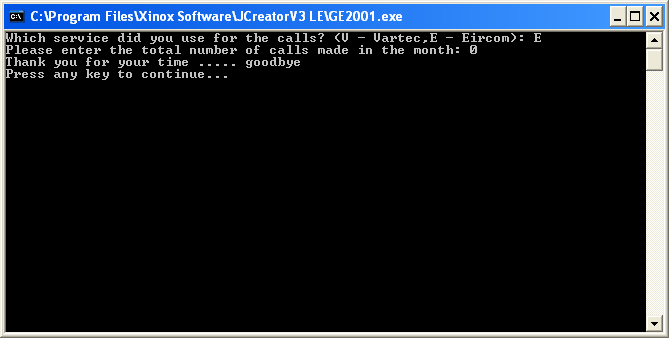
Run 1:



Run 2:



Run 3:

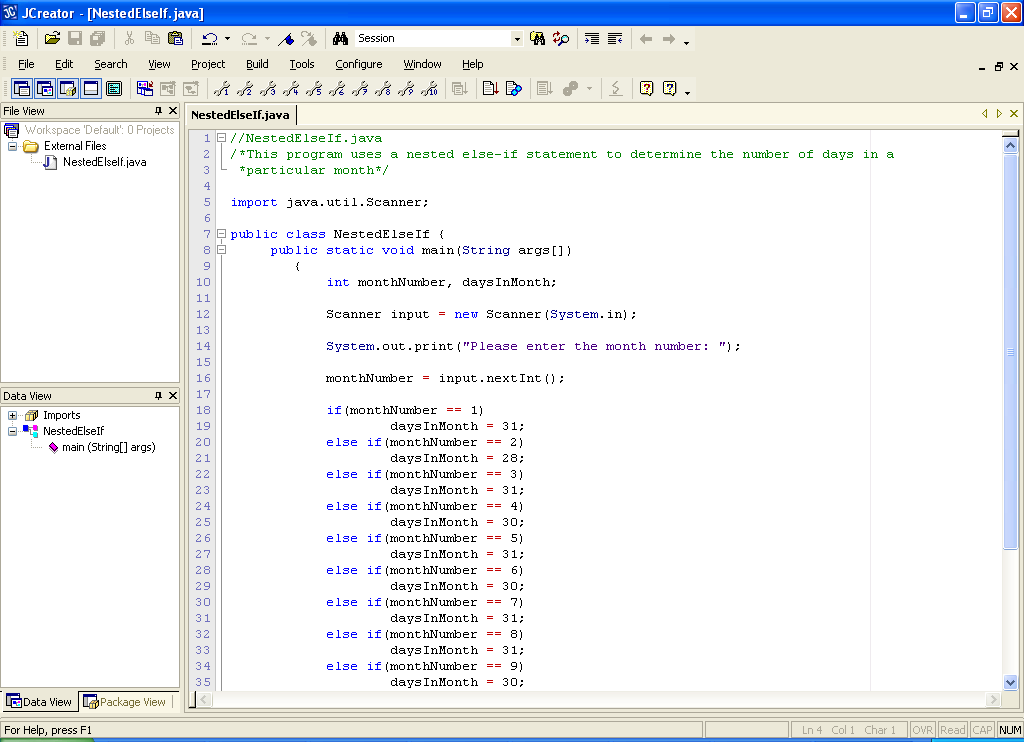


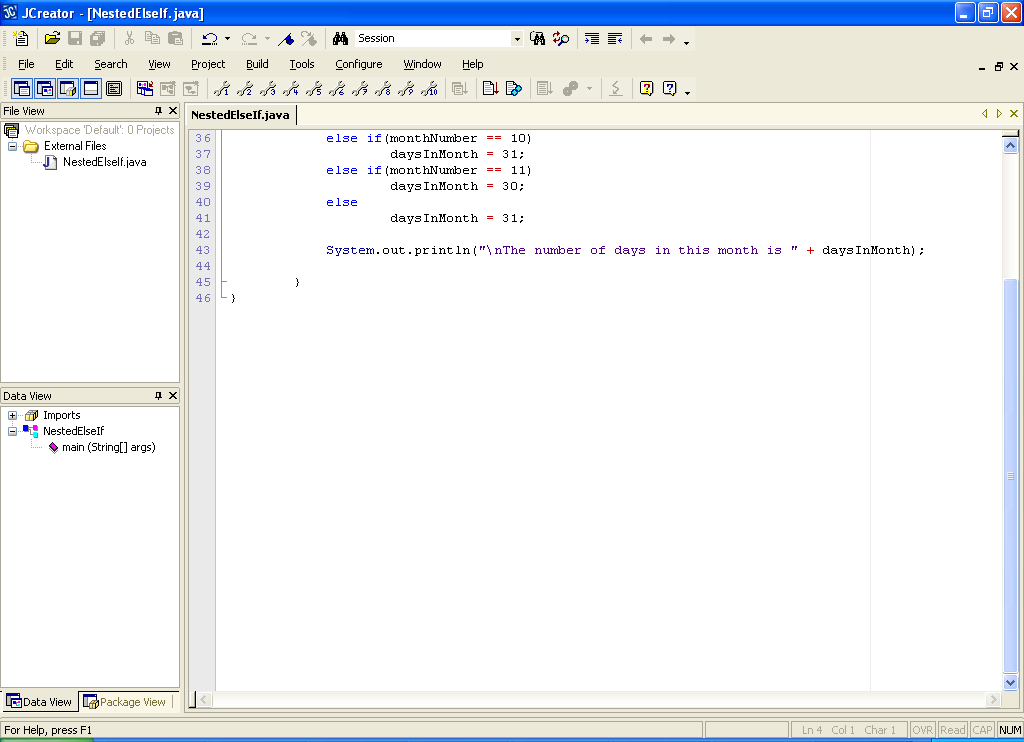
**Multi-way decisions using a nested else-if statement**

Sometimes we need to make decisions based on several different possibilities and in these circumstances, a nested **else-if statement** is recommended.

**Aim**: The purpose of this program is to read in a month number and then use a nested else-if statement to display the number of days in that month.

**Java Code**:





**Analysis of program**

• After declaring the variables, the user is prompted for a month number and this is read in using the nextInt() method of the Scanner class.

• Next, the decision making process occurs. Here there are 12 possibilities, one for each month of the year. The first thing that happens here is that the users input is compared with the value 1. If they match then the value of daysInMonth is set to 31. If they don’t match the else section is entered and the user’s input is compared to 2. This testing process continues until a match is found. In this case, if no match is found (if the user has entered an invalid month value), then it will actually execute the final else, which means that the daysInMonth would be set to 31 (really this final else is there to deal with month 12).

Note that with this type of program, the **number of tests performed varies according to the value inputted**. So, if the user enters a month number of 1, only one test is performed since the first else clause never gets executed and that bypasses most of the program’s code. The worst-case scenario is that the user enters a month number that is either less than 1 or greater than 11 because then a whole 11 tests will be performed at run time.

Note also the **layout** of the nested else-if structure here. It could equally well have been written as follows:

if(monthNumber == 1)

daysInMonth = 31;

else

if(monthNumber == 2)

daysInMonth = 28;

else

if(monthNumber == 3)

daysInMonth = 31;

else

if(monthNumber == 4)

daysInMonth = 30;

else

if(monthNumber == 5)

daysInMonth = 31;

else

if(monthNumber == 6)

daysInMonth = 30;

else

if(monthNumber == 7)

daysInMonth = 31;

else

if(monthNumber == 8)

daysInMonth = 31;

else

if(monthNumber == 9)

daysInMonth = 30;

else

if(monthNumber == 10)

:

:

//ran out of space!

But this coding, although perfectly valid, is **not as easy to read** (or write) as the format used in our program by **convention**.

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

Click the **New File** icon on the JCreator IDE and save the file as **NestedElseIf.java** in your Lab4 folder. Now type in the code for the program above. Make sure to use the **copy and paste facility** of the IDE as much as possible.

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 input values. How many tests should you perform in order to test the program properly here? Also, **see what happens when you enter an invalid month number such as -1 or 15.** Is this what you expect?

**Another Program involving Nested else-if statements**

**Aim**: We wish to write a program which asks the user to enter the electrical kWh they have used in the past 2 months and output the total charge according to the following list of rates:

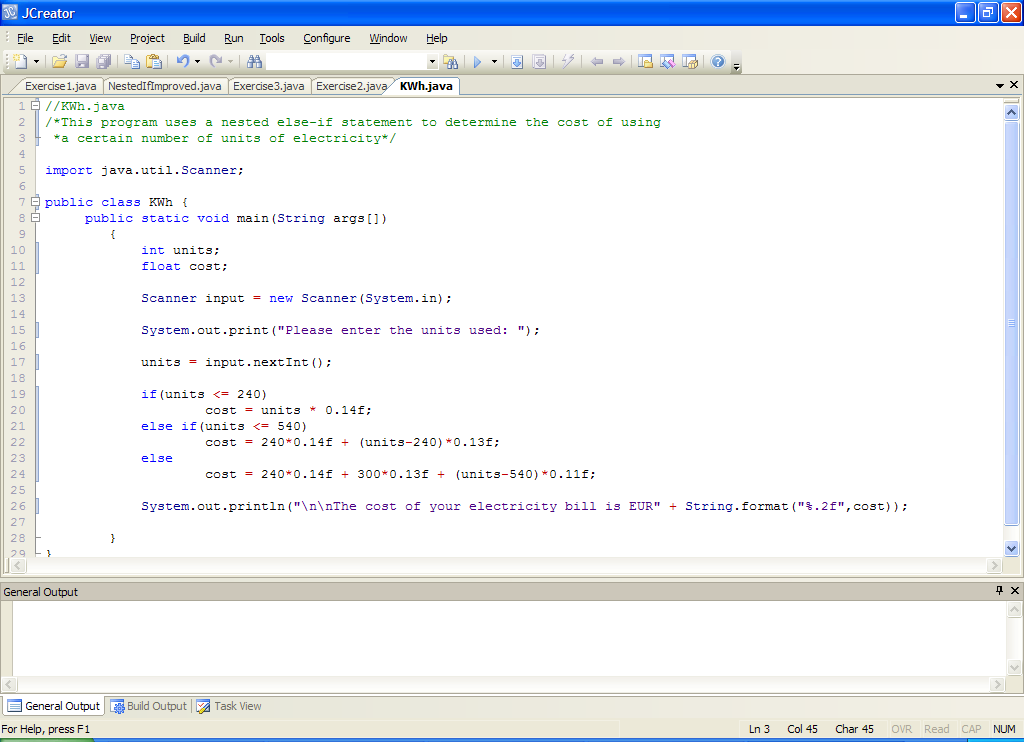
first 240kWh: 14c per kWh

next 300kWh: 13c per kWh

over 540kWh: 11c per kWh

This means that if the user consumes 280kWh units, they will be charged 14c per kWh for the first 240 units and 13c per kWh for anything over this i.e. the remaining 40kWh.

**Java Code**



**Analysis of Code**:

• The user is prompted for the amount of kWh they use and this is stored in the variable units.

• In the next section of the program a nested else-if is used to determine the cost, which will **vary depending on the number of units used** according to the rates supplied in the question.

• In this case, if the user inputs some value for kWh which is less than or equal to 240, then each unit will be charged at the rate of 14c per kWh. Note here that the first if section will be executed and the else section is completely bypassed.

Now, if on the other hand, the user inputs a value for kWh which is more than 240 **but less than or equal to** (240+300) i.e. **540**, then we know that the charge for the first 240 will be at 14c per kWh and at 13c for anything remaining. The “remaining” part is coded as **kWh-240** in this case. Here the first if has been bypassed and the first else section is executed. The second if condition is satisfied in this case.

• Finally, if the user inputs a value for kWh which is more than 540 then we know that the charge for the first 240 will be at 14c, the next 300 will be at 13c and anything remaining will be charged at 11c per kWh. In this case the first if has been bypassed, the first else section is executed, the second if is bypassed and the second else is executed.

• The println() just displays the cost of the bill to 2 decimal places using format()

Take your time to figure out exactly how this program is working as its tricky enough.

**Typing in, Saving, Compiling, Running and Testing the Program just Analysed**

You should now take the program analysed above and type it in to your editor window. Save it as KWh.java, compile it and then run it to see it in operation.

In order to **test this program fully**, use the following test data and fill in the details for the actual results. I have calculated the expected results for you already.

**Value Entered for kWh Expected Output(cost) Actual Output(cost)**

0 (<240) 0.00

240 (=240) 33.60

350 (<540) 47.90

540 (=540) 72.60

600 (>540) 79.20

If your actual results correspond to those expected you can be quite confident at this stage that this program will produce the correct results in **every case**. You know this because you have accounted for each and every possibility in your test data. You have tested the case where kWh<240, =240 (called a **borderline case** which is very important to check for), <540, =540 (another **borderline case**) and >540. This covers all the possible numbers which can be entered.

The point which is really being made here is that it is **vital that you test your programs fully** and account for every case which may arise. As you can see this **does not mean** that you have to test the program with every number which exists (because you’d be here ‘til Christmas 2099 doing that!!). It does mean that you should **test specific ranges** of values and you can see that this program covers all the **meaningful values** i.e. 0 upwards (since you cannot consume a negative amount of units!). However, this **program will happily accept negative values** since there is **no validation code** in the program. This is a topic we will cover later.

**Exercise 4**

A mobile phone company charges its customers at a number of different rates depending on how many units they consume per day as follows:

**number of units rate/unit**

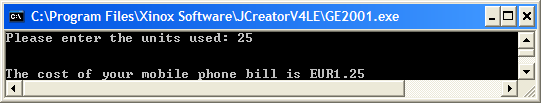
first 99 5c

next 200 3c next 100 2c

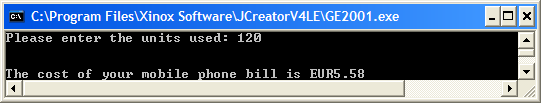
anything more 1c

Write a program which asks a customer to input the number of units they have consumed and determines the associated total cost using the table above. Your program should run as indicated in the following sample screenshots, with the total cost corrected to 2 **decimal places**.

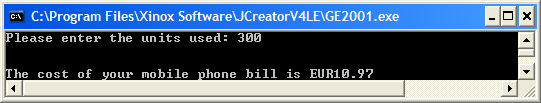
Run 1:



Run 2:



Run 3:



Run 4:

