**Institute of Technology Tralee**

**Computing Department**

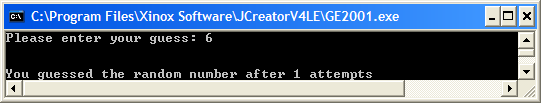
**Object Oriented Programming 1**

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

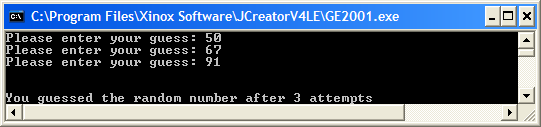
**Tutorial 7 – Java Loops and Input Validation**

1. Write the code for a Java program that firstly generates a random number between 1 and 100 inclusive. Once this is done, a while loop should be written to iterate up to 5 times and, on each iteration, the user should be asked to guess the value of the randomly generated number, effectively giving the user a maximum of 5 guesses at the number. If the user guesses the number within the first 4 attempts, then the loop should terminate early. Once the loop has finished, the program should then display whether or not the user guessed the random number correctly and, if they did, how many attempts it took. If the user fails to guess the random number within the 5 attempts, then the random number should be displayed. Your program would run as indicated in the following sample screenshots:

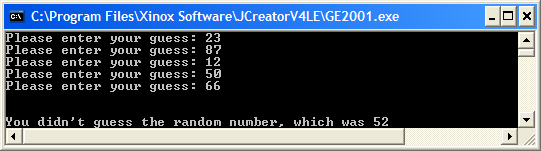
**Run 1 – guessed correctly after 1 attempt**



**Run 2 – guessed correctly after 3 attempts**

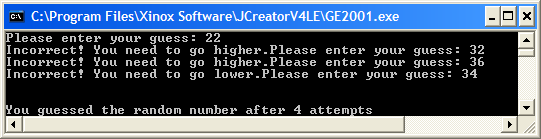


**Run 3 – used all 5 guesses but all incorrect**

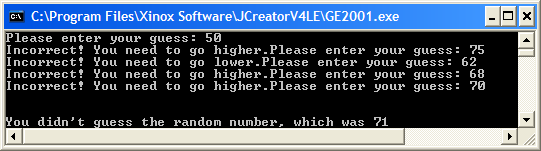


1. You should now make some modifications to the program from part(a) so that the user has a better chance at guessing the random number within the 5 loop iterations. This time the user will enter their guess and, if it does not match the random number generated, they will be told whether they should go “higher” or “lower” on their next guess for each of the first 4 guesses. In this way, the program will now resemble the game of “higher/lower”. The program would now run as indicated in the following sample screenshots:

**Run 1 – guessed correctly after 4 attempts**



**Run 2 – used all 5 guesses but all incorrect**

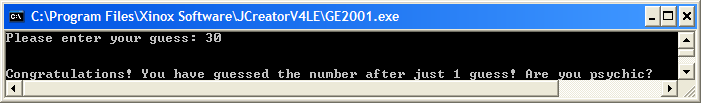


1. You should now make some modifications to the program from part (b) so that, when the loop has completed, the user will receive one of the following messages depending on the number of guesses used:

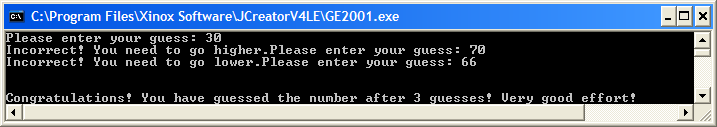
|  |  |
| --- | --- |
| **Guesses Used** | **Message Given** |
| 1 | Congratulations! You have guessed the number after just 1 guess! Are you psychic? |
| 2 | Congratulations! You have guessed the number after just 2 guesses! Excellent attempt! |
| 3 | Congratulations! You have guessed the number after 3 guesses! Very good effort! |
| 4 | Congratulations! You have guessed the number after 4 guesses! Good effort! |
| 5 and successful | Congratulations! You have guessed the number after 5 guesses! Just about got there! |
| 5 and unsuccessful | Commiserations! The actual target number was <target number>. You might win next time! |

The program would now run as indicated in the following sample screenshots:

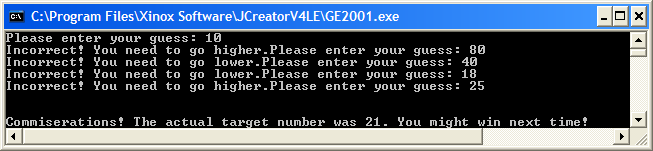
**Run 1** - **guessed correctly after 1 attempt**



**Run 2 – guessed correctly after 3 attempts**



**Run 3 – used all 5 guesses but all incorrect**



1. Currently, the user is able to enter invalid values for their guess, which could easily make the program crash at runtime. You should now add code to the program which will ensure that no invalid user input is possible. If the user does try to enter invalid input then the program will continually issue a suitable error message and ask for a valid input to be supplied. Note here that your validation routine can be simplified greatly, given the fact that the only valid inputs are whole numbers in the range 1-100 inclusive. The following sample screenshot shows how the program would run now:

