

How to **THINK** like a Programmer

Problem Solving for the Bewildered

paul vickers



Problem Solving 8

The do-while loop

Aim



- ▶ In this lesson we will continue our study of looping control structures
 - ▶ At-least-once iterations
 - ▶ The `do-while` loop
 - ▶ Solving a problem involving at-least-once iteration using a `do-while` loop and the problem-solving strategy.

At-least-once iterations



- ▶ In some programming situations, it is required that a loop executes at least once. This is the case for **menu type applications**, also for example in an ATM machine where the request for PIN input displays after the user has entered their card.
- ▶ A while loop could be used in this situation but it is **not the most elegant solution** since the test condition must be tested before the user is actually asked for their PIN

Set PIN to -1

while (PIN \neq storedPIN)

 Display 'Enter your PIN'

 Read in PIN

endwhile

The do-while Loop



- ▶ Because of this issue, most programming languages support an alternative, more efficient, looping structure for these situations, called the **do-while** loop
- ▶ The general form of this structure is as follows:

```
do  
    Action block  
while (condition expression)
```

- ▶ Here the action block will execute at least once.
After the first iteration, the conditional expression is evaluated. If it evaluates to true then the loop executes again, otherwise the loop stops.

Example of do-while



- ▶ We can now use the `do-while` with our PIN example:

```
do
    Display 'Please enter your PIN'
    Read in PIN
while (PIN ≠ storedPIN)
```

- ▶ This loop will **execute at least once**, give the user the message and continue reiterating as long as an incorrect PIN is entered.
- ▶ The `do-while` can be also used for **counter-controlled iteration** as you have seen in the lab but the for loop is more efficient here.

do-while Looping - understanding the problem



- ▶ Imagine that we need to write an algorithm that involves displaying a set of menu options to the user and asking them to select from the options provided. The selected option should then execute and display its results and then the options should re-appear. This should continue until the user selects the exit option. The menu should look as follows:
 - A. Calculate the natural log of a number
 - B. Calculate the square root of a number
 - C. Exit
- ▶ The problem statement tells us a number of things:
 - ▶ We know that we have to display some menu options to the screen
 - ▶ We know that we must ask the user for their choice
 - ▶ We know that once the users choice is known, a certain set of actions should happen

do-while Looping - understanding the problem



- ▶ We know that once the appropriate set of actions have executed the menu options should reappear and that this should keep happening until the user selects 'exit'.
- ▶ We know what the menu options are and that the first two involve mathematical operations. There are math functions available in all programming languages already so this is not a big problem.

do-while Looping – devising the plan



- ▶ A good way to approach this type of problem is to give it some thought. Think about the sequence of events in this algorithm.
- ▶ The first thing to happen will be that the menu options appear on screen. On the same screen we should prompt the user for their choice and read that in.
- ▶ Once the choice is read in, the program will execute a certain set of actions associated with that choice. Note **since choice is involved here, we should use an if-type structure**. In this case there are 3 possible choices so the most appropriate structure is a **3-way nested else-if**.
- ▶ The set of actions associated with the choices are mathematical for choices A and B and exit for choice C

do-while Looping – devising the plan



- ▶ We have little to do for any of the 3 choices here. Math predefined functions already exist for determining the natural log and the square root of a number and the exit option is there to allow us to exit from the loop.
- ▶ Is the loop determinate or indeterminate? Here, once the user has supplied their choice, it will execute a set of actions and then the menu options will reappear. The loop will keep executing until the user decides to select “exit”. This makes it an **indeterminate loop**.
- ▶ We have a choice as to whether to use a while loop or a do-while loop but because **the menu options need to appear at least once anyhow**, we should go with a do-while loop here.

do-while Looping – devising the plan



- ▶ We have figured out pretty much everything at this stage. One minor, but important, consideration involves the display. In a case like this, it is important that once the user has chosen option A or B and this selection has executed that the **output screen halts giving the user a chance to see results**. A “please hit return to continue” message and a **halt action** will do this.
- ▶ The halt action is simply a “reading user input” action since this always halts the programs execution.

do-while Looping – executing the plan



- ▶ Our final pseudocode draft here might be:

1. do

1.1 Display the 3 menu options

1.2 Prompt for the user's choice

1.3 Read in the user's choice

1.4 if (choice = 'A')

1.4.1 Prompt the user to enter a number

1.4.2 Read in the number

1.4.3 Calculate and display the natural log of the number

1.5 else if (choice = 'B')

1.5.1 Prompt the user to enter a number

1.5.2 Read in the number

1.5.3 Calculate and display the square root of the number

1.6 else if (choice = 'C')

1.6.1 Display "Thanks for using the system" message

endif

1.7 Display "Please hit return to continue" message

1.8 Read in return keypress from the user

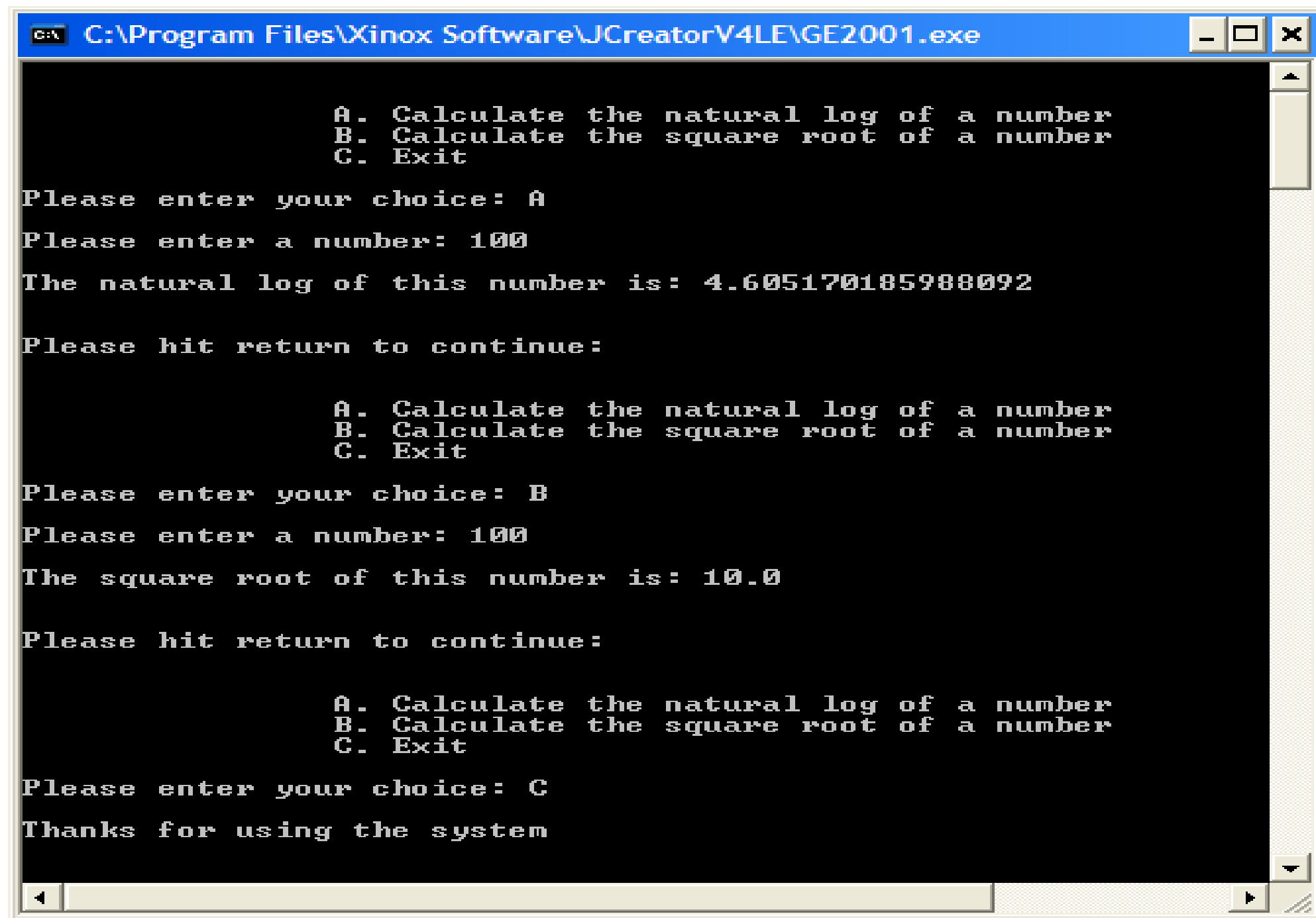
while (choice ≠ 'C')

do-while Looping – assessing the result and documenting the solution



- ▶ As usual, once the algorithm has been written, it should be tested out fully. Trace through how things will happen for each of the 3 possibilities here. Note that we are assuming that the user will only put in either A, B or C as their choice – there is **no validation** in our solution but that is okay here, **it was not asked for** or mentioned in the problem specification.
- ▶ If the assessment of the solution finds a flaw in the logic, then the algorithm must be reconsidered and fixed.
- ▶ If you are confident the solution is valid, then it should be **documented**. For us, this means adding a **brief summary at the start** about what the algorithm is doing and, if necessary, adding a few extra **comments** within the algorithm to explain specific sections of it.

You should now take the pseudocode solution you have written and translate it into a Java program. It should run as indicated below – you can determine the natural log and square root of a number by using the **Math.log()** and **Math.sqrt()** methods respectively.



```
C:\Program Files\Xinox Software\JCreatorV4LE\GE2001.exe

      A. Calculate the natural log of a number
      B. Calculate the square root of a number
      C. Exit

Please enter your choice: A
Please enter a number: 100
The natural log of this number is: 4.605170185988092

Please hit return to continue:

      A. Calculate the natural log of a number
      B. Calculate the square root of a number
      C. Exit

Please enter your choice: B
Please enter a number: 100
The square root of this number is: 10.0

Please hit return to continue:

      A. Calculate the natural log of a number
      B. Calculate the square root of a number
      C. Exit

Please enter your choice: C
Thanks for using the system
```

ACTIVITY

Write a pseudocode solution which reads in the name, course and GPA of a group of students (whose size is at least 1) and displays the name, course and GPA of the student who had the highest GPA. Use a do-while in your pseudocode here. Before the loop ends, you ask the user if they wish to enter more details – the loop continues until they answer “no”.

Solution



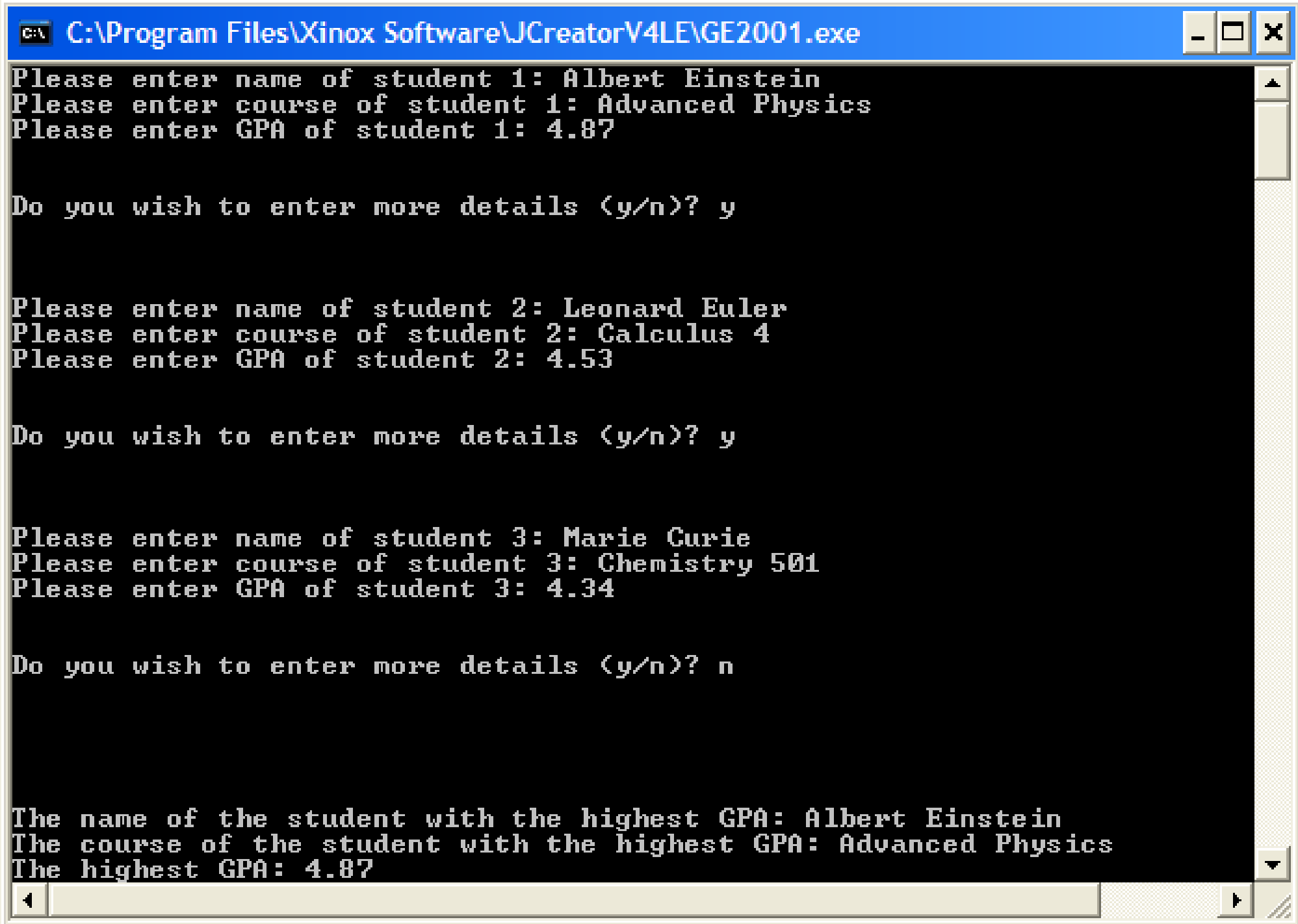
1. Initialise i to 1
2. do
 - 2.1 Prompt for the students name
 - 2.2 Read in the students name
 - 2.3 Prompt for the students course
 - 2.4 Read in the students course
 - 2.5 Prompt for the students GPA
 - 2.6 Read in the students GPA
 - 2.7 if (i = 1) 'it must be the first loop iteration'
 - 2.7.1 Set nameHighestGPA to studentName
 - 2.7.2 Set courseHighestGPA to studentCourse
 - 2.7.3 Set highestGPA to studentGPA
 - 2.8 else if (studentGPA > highestGPA)
 - 2.8.1 Set nameHighestGPA to studentName
 - 2.8.2 Set courseHighestGPA to studentCourse
 - 2.8.3 Set highestGPA to studentGPA
 - endif
 - 2.9 Display a message asking user if they wish to enter more

Solution



- 2.10 Read in the users choice
- 2.11 Set i to $i + 1$
- while (choice \neq 'no')
- 3. Display nameHighestGPA, courseHighestGPA and highestGPA

You should now take the pseudocode solution above and translate it into a Java program. It should run as indicated below:



```
C:\Program Files\Xinox Software\JCreatorV4LE\GE2001.exe
Please enter name of student 1: Albert Einstein
Please enter course of student 1: Advanced Physics
Please enter GPA of student 1: 4.87

Do you wish to enter more details (y/n)? y

Please enter name of student 2: Leonard Euler
Please enter course of student 2: Calculus 4
Please enter GPA of student 2: 4.53

Do you wish to enter more details (y/n)? y

Please enter name of student 3: Marie Curie
Please enter course of student 3: Chemistry 501
Please enter GPA of student 3: 4.34

Do you wish to enter more details (y/n)? n

The name of the student with the highest GPA: Albert Einstein
The course of the student with the highest GPA: Advanced Physics
The highest GPA: 4.87
```