|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LEARNING PROFILE FOR Interest2WithScanner | | | | | |
| *Name* | *:* | *Tyler Lucas* | *Due Date* | *:* | *N/A* |
| *Student ID* | *:* | *3305203* | *Submission Date* | *:* | *N/A* |

# 1. Problem Statement:

Textbook example program.

# 2. Description of the Code:

Computes interest earned on an investment over time. Initial amount and interest rate input by the user.

# 3. Errors and Warnings:

*Note: No errors.*

# 4. Sample Input and Output:

# Starting investment value/principal: $100

# The starting principal has been set to $100.00

# Annual interest rate [%]: 7.5

# The annual interest rate has been set to 7.5%

# Accrual time, in years: 3

# The accrual time has been set to 3.0 years

# Number of compound periods per year: 12

# The number of compound periods per year has been set to 12

# The interest earned on $100.00 initial principal with 7.5% annual interest over 3.0 years with 12 compounding periods per year is $25.14, for a final investment value of $125.14.

# 5. Discussion:

Below is a comparison of the code of my own efforts to use Scanner in the first iteration of this interest computing algorithm (*Interest.java*), and of the code from the current example (*Interest2WithScanner.java*), both with my more complex interest determining algorithm. Minor changes were made to the below code (not the sources) to better compare them and format it in the table. The differences in output are the same as between *Interest.java* and *Interest2.java* (see *COMP268\_Interest2\_MyProgramProfile.docx*).

|  |  |
| --- | --- |
| Interest.java | Interest2WithScanner.java |
| Scanner sc = new Scanner( System.in );  double ppl = 0; // Value of investment.  double rate = 0; // Annual interest rate.  double t; // Accrual time, in years.  double n; // Number of compound periods per year.  double pint; // Interest earned.    System.out.print("\nStarting investment value/ ppl: $");  while( !sc.hasNextDouble() ){ sc.next(); }  ppl = sc.nextDouble();  sc.nextLine();  System.out.println("The starting principal has been set to $” + ppl);  System.out.println();    System.out.print("Annual interest rate [%]: ");  while( !sc.hasNextDouble() ){ sc.next(); }  rate = sc.nextDouble()/100;  sc.nextLine();  System.out.println("The annual interest rate has been set to "  + rate\*100 + "%");  System.out.println();    System.out.print("Accrual time, in years: ");  while( !sc.hasNextDouble() ){ sc.next(); }  t = sc.nextDouble();  sc.nextLine();  System.out.println("The accrual time has been set to " + t + " years");  System.out.println();    System.out.print("Number of compound periods per year: ");  while( !sc.hasNextDouble() ){ sc.next(); }  n = sc.nextDouble();  sc.nextLine();  System.out.println("The number of compound periods per year has been “  + “set to " + n);  System.out.println();    pint = ppl \* ( Math.pow(1 + rate/n, n\*t) - 1 );    System.out.println("\nThe interest earned on $" + ppl + “ initial “  + "principal with " + rate\*100 + "% annual interest over " + t  + " years with " + n + " compounding periods per year is”  + “ $" + pint + ", for a final investment value of “  + “$" + ( ppl + pint ) + "."); | Scanner stdin = new Scanner( System.in );    double ppl; // initial investment value  double rate; // annual interest rate, decimal  double t; // Accrual time, in years.  double n; // Number of compound periods per year.  double pint; // interest earned during the year    System.out.print("Starting investment value/principal: $");  ppl = stdin.nextDouble();  System.out.printf("The starting principal has been set to "$%1.2f%n%n",  ppl);    System.out.print("Annual interest rate [%]: ");  rate = stdin.nextDouble()/100;  System.out.printf("The annual interest rate has been set to "  + "%1.1f%%%n%n", rate\*100);    System.out.print("Accrual time, in years: ");  t = stdin.nextDouble();  System.out.printf("The accrual time has been set to %1.1f “  + “years%n%n", t);    System.out.print("Number of compound periods per year: ");  n = stdin.nextDouble();  System.out.printf("The number of compound periods per year “  + “has been set to %1.0f%n%n", n);    pint = ppl \* ( Math.pow(1 + rate/n, n\*t) - 1 );    System.out.printf("%nThe interest earned on $%1.2f initial “  + “principal with %1.1f%% annual interest over %1.1f”  + “ years with %1.0f compounding periods per year is”  + “ $%1.2f, for a final investment value of “  + “$%1.2f.%n", ppl, rate\*100, t, n, pint, ppl + pint); |

*Interest.java* is a bit more complex than *Interest2WithScanner.java*, including some input error detection and handling. Using while( !sc.hasNextDouble() ){ sc.next(); } before the input ppl = sc.nextDouble() allows the user to input characters that aren’t type double, skipping over everything until a number is found. While this seems to be a quick fix for missing keys and related errors, it may also result in unintended behaviour. For example, it sets the value to “1000000” given the input “about 1000000 bucks”, but also sets the value to “1” given the input “about 1 million bucks”. It may be better to simply display an error when the user inputs the wrong type instead of attempting to parse the intended input.

I had to do some research to discover the purpose of including the line sc.nextLine() after each Scanner input numerical fetching operation. A user’s input is in the form of a character string, usually terminated with an end-of-line (EOL) character, normally the newline character ‘\n’. The Scanner’s numerical fetching operations, like sc.nextDouble(), only look so far as the number itself, no further, which means that it doesn’t advance past the EOL. sc.nextLine() does this. The error handling in *Interent.java* has the same effect, skipping over all non-numerical characters, including EOL characters, so these sc.nextDouble() statements are not technically required, but I’ve kept them in there for what I think is good practice. Error handling code is supposed to handle situations when the program or the user does something they’re not supposed to do, and the EOL character is supposed to be there. It’s not an error, so I shouldn’t force the error handling code to deal with it. It should be noted that the EOL character depends on the system, and may even be two characters; this is more of an issue when working with text files on various systems.

A difference carried from *Interest2.java* is the use of the printf method instead of println.