CMPT 225, Spring 2018, Assignment 2

Due date: February 6, 2018, 5:30 PM (before class)

1. Programming exercise. Modify the *CreditCard* class we saw in class to make the following changes (you can make them all at once, in the same class definition):

   1) Check that the price argument passed to function *chargeIt* and the payment argument passed to function *makePayment* are positive. If one of them is not positive, print the message "Error: the input argument must be positive!" and do not allow the charge or the payment through (i.e. leave the internal state unchanged). **(5 pts.)**

   2) Charge an interest of 5% on each payment (i.e. if the payment is 100, only 95 goes towards reducing the balance). Print the message "The current remaining balance is x dollars.", where x is the balance, after the payment goes through. **(5 pts.)**

   3) Add a private variable that keeps track of the fraction of the limit that has been used up (i.e. the ratio of the balance to the limit), and when a charge is made that makes this fraction exceed 50%, print out the message "Warning: the current balance on credit card N is y% of your limit.", where N is the card number and y is the percentage of the limit that has been taken up by the balance. **(5 pts.)**

   Details and suggestions:

   The source code for the *CreditCard* class is available in lines 400 to 515 of this file: *https://www.overleaf.com/read/pmjttghgmnrx*; be sure to remove any LaTeX code.

   You will need to add a print method << as shown in the textbook (pp. 49 and 51).

   The percentages (i.e. 5% and 50%) should both be constant floats in the header file.

   Test cases:

   CC = CreditCard(1234, "Bill Gates", 2500);
   creates a credit card *CC* for Bill Gates with number 1234 and a 2500-dollar limit.

   CC.chargeIt(1000);
   goes through fine and updates the balance to 1000.

   CC.chargeIt(-100);
   prints out *Error: the input argument must be positive!*

   CC.chargeIt(500);
   prints out *Warning: the current balance on credit card 1234 is 60% of your limit.*

   CC.makePayment(-100);
   prints out *Error: the input argument must be positive!*

   CC.makePayment(300); cout << CC << endl;
   prints out:
   Number = 1234
   Name = "Bill Gates"
   Balance = 1215
   Limit = 2500

2. Programming exercise. Generalize the *Person-Student* class hierarchy to include classes *Faculty, UndergraduateStudent, GraduateStudent, Professor, Instructor.*

   Clearly show the inheritance structure of these classes. Propose some appropriate member variables for each class. **(20 pts.)**

   Details and suggestions:

   The source code for the *Person* and *Student* classes is available in lines 545 to 636 of this file:

   https://www.overleaf.com/read/pmjttghgmnrx; be sure to remove any LaTeX code.

   Each of the classes should have at least one member variable that is unique to it.

   Include a virtual print function that ensures that you can use dynamic binding:

   For a *Person*, print out the name and the ID number (on separate lines).

   For a *Student*, also print out the major and the graduation year (on separate lines).

   For an *UndergraduateStudent*, also print out the residence hall (default: "None").

   For a *GraduateStudent*, also print out the thesis title (default: "None").

   For a *Faculty*, print out the department and the starting year (on separate lines).

   For a *Professor*, also print out the research topic (default: "None").

   For an *Instructor*, also print out the course number being taught (default: "None").

   Demonstrate your code with a test function that shows an instance of each class.

   Bonus: include a class diagram for your design in the Unified Modeling Language.

3. Programming exercise. Write a C++ class *HarmProgression* derived from the abstract *Progression* class to produce a harmonic progression, which is to say, a progression where the reciprocals of the values form an arithmetic progression. Just like with an arithmetic progression, it is defined by its first value $\frac{1}{a}$ and its increment d, and its n-th element is given by the formula $\frac{1}{a+(n-1)d}$. **(20 pts.)**

   Details and suggestions:

   The source code for the *Progression* class is available in lines 645 to 661 of this file: *https://www.overleaf.com/read/pmjttghgmnrx*; be sure to remove any LaTeX code.

   Modify the abstract class to allow floating-point numbers.

   Print the numbers one per line, rounded to 5 decimals; do not change them internally.

   The definition of nextValue() should remain very short (2-3 lines at most).

   Include a default constructor that starts with 1 as the first value and the increment.

   Also include a parametric constructor that takes any first value and increment.

   Bonus: get the class to throw a *ZeroDivide* exception with the message "Divide by zero in Module HarmProgression." if a division by 0 occurs instead of failing.

   Test cases (you must make sure your program works on these):

   prog = new HarmProgression(); prog –> printProgression(5); prints out

   1
   0.5
   0.33333
   0.25
   0.2

   prog = new HarmProgression(1,0); prog –> printProgression(5); prints out five 1's.

   prog = new HarmProgression(0.5,1); prog –> printProgression(5); prints out

   0.5
   0.33333
   0.25
   0.2
   0.16666

   Test case to check **only if you are attempting the bonus**:

   prog = new HarmProgression(1,-0.25); prog –> printProgression(5); prints out

   1
   1.33333
   2
   4
   Divide by zero in Module HarmProgression.

   since the fifth element is undefined, so an exception needs to be thrown at this stage.

4. What number would you replace the 16 by in the stopping condition in the declaration of the first **for** loop in the *testCard* method so that the charges cause exactly one of the three cards to go over its credit limit? Which card is it? **(10 pts.)**

5. Prove that if x and y are positive real numbers and $x < y$, then $n^x \in O(n^y)$ but $n^y \notin O(n^x)$. **(10 pts.)**

6. Give a recursive definition of a singly linked list. **(5 pts.)**

7. Write the pseudocode of a non-recursive function for finding, using only link hopping, the middle node of a doubly linked list with header and trailer sentinels (no counters allowed!). **(10 pts.)**

8. Write the pseudocode for a function to count the number of nodes in a circularly linked list. **(10 pts.)**