**Montgomery College**

**CMSC 203**

**Assignment 4 Design**

Create UML diagrams for all the classes in this assignment along with pseudo-code for **addProperty**() method specified in ***ManagementCompany*** Class***.***

Refer to the [**Pseudocode Guideline**](#PSGdline)on how to write Pseudocode.

**Pseudocode Guideline**

Pseudocode is code written for human understanding­ n­ot a compiler. You can think of pseudocode as “English code,” code that can be understood by anyone (not just a computer scientist). Pseudocode is not language specific, which means that given a block of pseudocode, you could convert it to Java, Python, C++, or whatever language you so desire.

Pseudocode will be important to your future in Computer Science. Typically, pseudocode is used to write a high-level outline of an algorithm.

As you may already know, an algorithm is a series of steps that a program takes to complete a specific task. The algorithms can get very complicated without a detailed plan, so writing pseudocode before actually coding will be very beneficial.

**How to Write Pseudocode**

There are no concrete rules that dictate how to write pseudocode, however, there are commonly accepted standards. A reader should be able to follow the pseudocode and hand-simulate (run through the code using paper and pencil) what is going to happen at each step. After writing pseudocode, you should be able to easily convert your pseudocode into any programming language you like.

We use indentation to delineate blocks of code, so it is clear which lines are inside of which method (function), loop, etc. Indentation is crucial to writing pseudocode. Java may not care if you don't indent inside your **if** statements, but a human reader would be completely lost without indentation cues.

**Remember:** Human comprehension is the whole point of pseudocode. So, what does pseudocode look like?

|  |  |
| --- | --- |
| **Pseudocode** | **Real Code in Java** |
| Declare an integer variable called n  Declare an integer variable sum.  Declare an integer variable f1  Declare an integer variable f2  If n is less than 2  sum =n  else  set sum to 0  set f1 and f2 to 1  repeat n times  sum = f1 + f2  f2 = f1  f1 = sum  end loop  print sum | **int** n,k, f1, f2, sum;  **if** ( n < 2 )  sum =n;  **else**  {  sum=0;  f1 = f2 = 1;    **for**(k=2; k<n; k++)  {  sum = f1 + f2;  f2 = f1;  f1 = sum;  }  }  System.***out***.println("Fibonacci of number " + n + " is "+ sum); |

**Finding the Fibonacci numbers till n:**

**Remember that pseudocode is not language specific so we are not looking for “almost Java” code, but instead, we are looking for a strong understanding of the algorithm at hand.**

**Psuedocode:**

1. **Start.**
2. **Data Element Class – Plot.java**
3. **Create instance variables that will represent x & y coordinates.**
4. **Create a Constructor.**
5. **Create a method called ‘overlaps.’**
6. **Create a method called ‘encompasses.’**
7. **Create a toString method to represent a Plot instance, in the format [x], [y], [width], [depth].**
8. **Data Element Class – Property.java**
9. **Create instance variables for property name, city, rental amount, owner, and plot.**
10. **Create a Constructor.**
11. **Create Getter & Setter methods.**
12. **Create toString method to represent a property instance, in the format [property name], [city], [owner], [rental amount].**
13. **Data Manager Class – ManagementCompany.java**
14. **Create instance variables for ManagementCompany name, Tax Id, and management fee percentage.**
15. **Create constant variables:** 
    * 1. **MAX\_PROPERT = 5.**
      2. **MGMT\_WIDTH = 10.**
      3. **MAGMT\_DEPTH = 10.**
16. **Create instance Array variable called ‘properties’, that stores the properties of management company.**
17. **Create instance variable called ‘plot’, that defines the plot of management company.**
18. **Create instance variable names ‘numberOfProperties’, that stores the current number of properties of a management company.**
19. **Create a Constructor.**
20. **Create Getter & Setter methods.**
21. **Create a method called ‘addProperty.’ This is an overload method. If there is a problem adding the property, this method returns:**
    * 1. **If Array is full, return -1.**
      2. **If the property is null, return -2.**
      3. **If the plot of property is not encompassed by management company, return -3.**
      4. **If the plot of property overlaps any other property’s plot, return -4.**
      5. **Else, if the property is successfully added, return index of Array where the property was added.**
22. **Create a method called ‘getTotalRent’, this method returns the total amount.**
23. **Create a method called ‘getHighestRentProperty’, this method returns property with the highest rent amount.**
24. **Create a method called ‘removeLastProperty’, this method removes the last property in the properties Array.**
25. **Create a method called ‘isPropertiesFull’, this method checks if the properties array has reached its maximum capacity.**
26. **Create a method called ‘getPropertiesCount’, this method returns the number of existing properties in the Array.**
27. **Create a method called ‘isManagementFeeValid’, this method checks if the management company has a valid fee.**
28. **Create a toString method, this method returns information of ALL properties within this management company.**
29. **Data Structure, an Array of property objects to hold the properties that the management company handles.**
30. **GUI Driver Class.**
31. **JUnit Test.**
32. **End.**