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***.***

Property
-propertyName -city -rentalAmount -owner -plot +PropertyName(String: name, String: city, int: rentalAmount, int: owner,v: plot) +setPropName() +getPropName() +setCity() +getCity() +setRent() +getRent() +setOwner() +getOwner() +setPlot() +getPlot()

Plot
-x -y +Plot() +overlaps() +encompasses()

ManagementCompany
-properties: MgmtCom[] - MAX_PROPERTY - MGMT_WIDTH - MGMT_DEPTH Property ManagementCompany() +addProperty(v: Property) +addProperty(String propertyName, String city, double rent, String ownerName) +addProperty(String propertyName, String city, double rent, String ownerName, int x, int y, int width, int depth) +totalRent() +maxRentProp() -maxRentPropertyIndex() +toString()

Refer to the **Pseudocode Guideline** 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.

_addProperty()

Receives property object as a parameter

Adds property object to properties array

Loops through array checks for the conditions below

return –1 if the array is full,

return 2 if the property is null,

return -3 if the plot for the property is not encompassed by the management company plot

return 4 if the plot for the property overlaps any other property's plot.

returns index of array if above conditions aren't met

_addProperty()

Receives String propertyName, String city, double rent, String ownerName

Calls Property constructor

Loops through array checks for the conditions below

return –1 if the array is full,

return 2 if the property is null,

return -3 if the plot for the property is not encompassed by the management company plot

return 4 if the plot for the property overlaps any other property's plot.

returns index of array if above conditions aren't met

_addProperty()

Receives String propertyName, String city, double rent, String ownerName, int x, int y, int width, int depth

Calls Property constructor

Loops through array checks for the conditions below

return –1 if the array is full,

return 2 if the property is null,

return -3 if the plot for the property is not encompassed by the management company plot

return 4 if the plot for the property overlaps any other property's plot.

returns index of array if above conditions aren't met

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?

Finding the Fibonacci numbers till n:

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);`

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.

Design

Turn in a UML class diagram for all classes in a Word document (or .uml file if you use UmlScluptor).

Submit pseudo-code for the primary methods specified in ManagementCompany.java, and Plot.java in a Word document. Do not just list what gets read in a printed out, but explain the algorithm being used.

Implementation

Submit two compressed files containing the follow (see below):

Note: Only submit the files that are modified. DO NOT submit the files that are already provided for you.

Deliverable format: The deliverables will be packaged as follows. Two compressed files in the following formats:

1st zip file: FirstInitialLastName_Assignment4_Complete.zip, a compressed file containing the following:

Word document with a name FirstInitialLastName_Assignment4.docx should include:

UML Class Diagram for all classes

Pseudocode for each of the methods specified in ManagementCompany.java, Property.java, and Plot.java.

Screen snapshots of the GUI with several properties (similar to screenshots in Assignment 4 Descriptions

Screen snapshot of Junit (display test for each method)

Screen snapshot of GitHub submission

Lessons Learned

Check List

doc (a directory) containing your javadoc files for the following classes: Property, ManagementCompany, Plot

src (a directory) *contains your files:*

Property.java, ManagementCompany.java, Plot.java, ManagmentCompanyTestSTUDENT.java

2nd zip file: FirstInitialLastName_Assignment4_Moss.zip, a compressed file containing the following files:

Property.java, ManagementCompany.java, Plot.java, and ManagmentCompanyTestSTUDENT.java **This .zip will not have any folders in it – only .java files.**

Notes:

Learning Experience: highlight your lessons learned and learning experience from working on this project.

What have you learned?

I learned getting data to persist from an array can be very difficult. Ultimately, I was unable to generate a list of properties. I could add them and display the plot, but I couldn't display the data.

What did you struggle with?

I struggled mostly with the arrays.

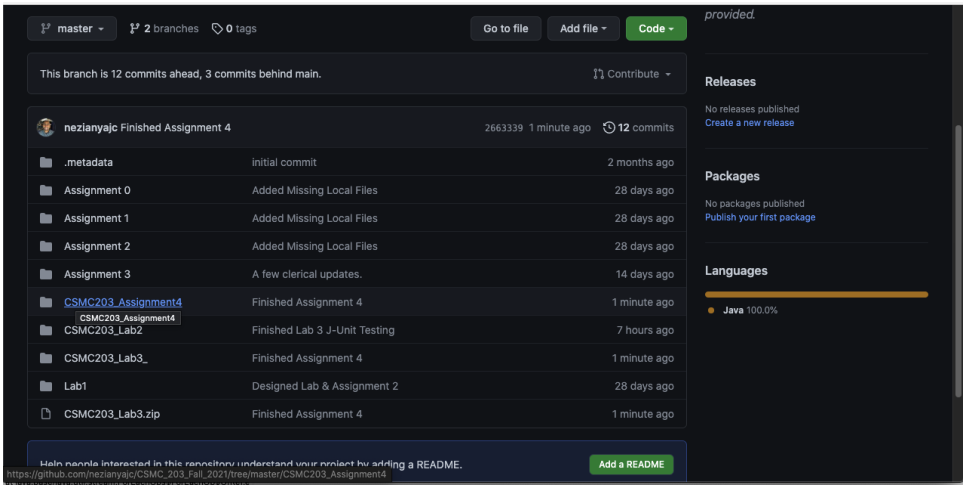
What will you do differently on your next project?

I'd work with arrays in a smaller project before trying to implement it with a GUI.

Include what parts of the project you were successful at, and what parts (if any) you were not successful at.

I was successful at getting the plot, gui, and add property functionality to work.

GitHub: In your repository (see Assignment0), upload your Word file and java file. You will want to upload these files as contents of a directory so that future uploads can be kept separate. Take and submit a screen shot of the GitHub repository.



Proper naming conventions: All constants, except 0 and 1, should be named. Constant names should be all upper-case, variable names should begin in lower case, but subsequent words should be in title case. Variable and method names should be descriptive of the role of the variable or method. Single letter names should be avoided.

Documentation: The documentation requirement for all programming projects is one block comment at the top of the program containing the course name, the project number, your name, the date and platform/compiler that you used to develop the project. If you use any code or specific algorithms that you did not create, a reference to its source should be made in the appropriate comment block. Additional comments should be provided as necessary to clarify the program.

Indentation: It must be consistent throughout the program and must reflect the control structure

Grading Rubric

See attachment: CMSC203 Assignment 4 Rubric.xlsx

Assignment 4 Check List (include Yes/No or N/A for each item)

#		Y/N or N/A	Comments
	Assignment files:		
	• FirstInitialLastName_ Assignment 4_Moss.zip	Yes	
	• FirstInitialLastName_Assignment4_Complete.zip	Yes	
	Program compiles	Yes	
	Program runs with desired outputs related to a Test Plan	No	
	Documentation file:		
	• Comprehensive Test Plan	Yes	
	• Screenshots for each Junit Test	Yes	
	• Screenshots for each Test case listed in the Test Plan	Yes	
	• Screenshots of your GitHub account with submitted Assignment# (if required)	Yes	
	• UML Diagram	Yes	
	• Algorithms/Pseudocode	Yes	
	• Flowchart (if required)	Yes	
	• Lessons Learned	Yes	
	• Checklist is completed and included in the Documentation	Yes	