**CMSC203**

**Assignment #4**



**Assignment Description**

## A property management company manages individual properties they will build to rent, and charges them a management fee as the percentages of the monthly rental amount. The properties cannot overlap each other, and each property must be within the limits of the management company’s plot. Write an application that lets the user create a management company and add the properties managed by the company to its list. Assume the maximum number of properties handled by the company is 5.

**Concepts tested by this assignment**

* Aggregation
* Passing object to method
* Array Structure
* Objects as elements of the Array
* Processing array elements
* Copy Constructor
* Junit testing

**Classes**

**Data Element class – Property.java**

The class *Property* will contain:

1. Instance variables for property name, city, rental amount, owner, and plot. Refer to JavaDoc for the data types of each instance variable.
2. toString method to represent a Property object.
3. Constructors and getter and setter methods. Refer to Javadoc of the Property class. One parameterized constructor will have parameters for property name, city, rent amount, owner, x and y location of the upper left corner of the property’s plot, the plot’s width and its depth. A second constructor will only have parameters for name, city, rental amount, and owner, and will generate a default x, y, width, and depth.

**Data Element class – Plot.java**

The class *Plot* will contain:

1. Instance variables to represent the x and y coordinates of the upper left corner of the location, and depth and width to represent the vertical and horizontal extents of the plot.
2. A toString method to represent a Plot object
3. Constructors, Refer to Javadoc for Plot class.
4. A method named *overlaps* that takes a Plot instance and determines if it is overlapped by the current plot.
5. A method named *encompasses* that takes a Plot instance and determines if the current plot contains it. Note that the determination should be inclusive, in other words, if an edge lies on the edge of the current plot, this is acceptable.

**Data Structure** – An Array of Property objects to hold the properties that the management company handles. This array will be declared as an attribute of the **ManagementCompany** class.

**Data Manager class – ManagementCompany.java**

This class should not have any output functionality (e.g., no GUI-related or printing related functionality), but should take input, operate on the data structure, and return values or set variables that may be accessed with getters.

It will contain instance variables of name, tax Id, management fee, MAX\_PROPERTY (a constant set to 5) and an array named properties of Property objects of size MAX\_PROPERTY, as well as two constants MGMT\_WIDTH and MGMT\_DEPTH, both set to 10; an attribute plot of type Plot that defines the plot of the ManagementCompany Class. Refer to Javadoc for more details.

The class *ManagementCompany* will contain the following methods in addition to get and set methods:

1. Constructors (refer to Javadoc for more details)
2. Method **addProperty (3 versions):** 
   1. Method addProperty version 1:
      1. Pass in a parameter of type *Property* object (calls *Property* copy constructor). It will add the copy of the *Property* object to the *properties* array.
   2. Method addProperty version 2:
      1. Pass in four parameters of types:
         * String propertyName,
         * String city,
         * double rent,
         * String ownerName.
      2. Calls *Property* 4-arg constructor.
   3. Method addProperty version 3:
      1. Pass in eight parameters of types:
         * String propertyName,
         * String city,
         * double rent,
         * String ownerName,
         * int x,
         * int y,
         * int width
         * int depth.
      2. Calls *Property* 8-arg constructor.
   4. addProperty methods will return the index of the array where the property is added. If there is a problem adding the property, this method will return -1 if the array is full, -2 if the property is null, -3 if the plot for the property is not encompassed by the management company plot, or -4 if the plot for the property overlaps any other property’s plot.
3. Method **totalRent**– Returns the total rent of the properties in the *properties* array.
4. Method **maxRentPropertyIndex**- returns the index of the property within the *properties* array that has the highest rent amount. This method will be **private**.
5. Method **maxRentProp**- Returns the highest rent amount of the property within the *properties* array. For simplicity assume that each "Property" object's rent amount is different. This method should call the **maxRentPropertyIndex** method**.**
6. Method **toString**- returns information of ALL the properties within this management company by accessing the "Properties" array. The format is as following example:

List of the properties for Alliance, taxID: 1235

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Property Name : Belmar

Located in Silver Spring

Belonging to: John Smith

Rent Amount: 1200.0

Property Name: Camden Lakeway

Located in Rockville

Belonging to: Ann Taylor

Rent Amount: 2450.0

Property Name: Hamptons

Located in Rockville

Belonging to: Rick Steves

Rent Amount: 1250.0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

total management Fee: 294.0

**You may need additional methods to include in this class. Follow the Javadoc files provided.**

**Driver class – (provided)**

The provided PropertyMgmDriverNoGui.java is a class that allows you to test the methods of ManagementCompany.java

**GUI Driver class – (provided)**

A Graphical User Interface (GUI) is provided. Be sure that the GUI will compile and run with your methods. The GUI will not compile if your methods in ManagementCompany.java are not exactly in the format specified.

Do not modify the GUI.

**JUnit Test**

Run the JUnit test file (provided). Ensure that the JUnit tests all succeed.

Do not modify the JUnit tests.

Implement your tests in ManagementCompanyTestSTUDENT. These tests should be similar to the Junit tests.

**Assignment Details**

Write a Data Element Class named Property that has fields to hold the property name, the city where the property is located, the rent amount, the owner's name, and the Plot to be occupied by the property, along with getters and setters to access and set these fields. Write a parameterized constructor (i.e., takes values for the fields as parameters) and a copy constructor (takes a Property object as the parameter). Follow the Javadoc file provided.

Write a Data Element Class named Plot that has fields specifying the X and Y location of the upper left corner of each Plot and a depth and width of each Plot. Notice that the X, Y location is at the upper left, not as in normal Cartesian coordinates, due to the grid system adopted by computer monitors.

A driver class is provided that creates rental properties to test the property manager. A Graphical User Interface is provided using JavaFX which duplicates this driver’s functionality. You are not required to read in any data, but the GUI will allow you to enter the property management company and each property by hand. A directory of images is provided. **Be sure to place the “images” directory (provided) inside the “src” directory in Eclipse.** The images do not need to display in order for the GUI to continue running.

Upload the initial files from Blackboard and your final java files to GitHub in your repo from Lab 1, in a directory named CMSC203\_Assignment4.

## Operation

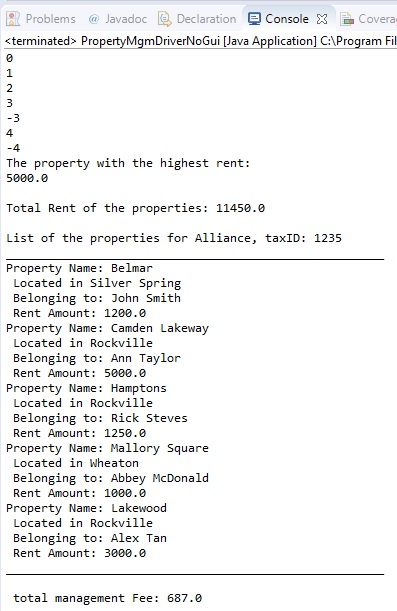
When driver-driven application starts, a driver class (provided) creates a management company, creates rental properties, adds them to the property manager, and prints information about the properties using the property manager’s methods.

When the GUI-driven application starts (provided), a window is created as in the following screen shots which allows the user to enter applicable data and display the resulting property. The driver and the GUI will both use the same classes and methods for their operation.

The JUnit test class also tests the same classes as the driver and the GUI.

**Examples**

***Expected output from running PropertyMgmDriverNoGui.java***



***Expected output from running with GUI:***

***PropertyMgmGui.java at startup***

A screenshot of a cell phone

Description automatically generated

***Add Management Co Info (Note Mgmt. Co Plot)***

A screenshot of a social media post

Description automatically generated

***Add property information - the Plot outline***

A screenshot of a social media post

Description automatically generated

***Add property information - successful addition***

A screenshot of a cell phone

Description automatically generated

***Add property information - unsuccessful: overlaps***

***A screenshot of a social media post

Description automatically generated***

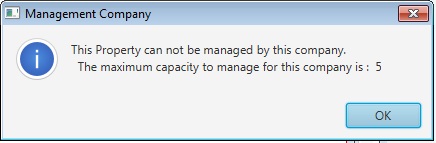
***Add property information - unsuccessful: Mgmt Co Plot does not encompass Property Plot***

***Note: red rectangle’s width extends to right of window.***

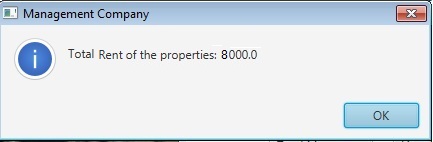
***A screenshot of a social media post

Description automatically generated***

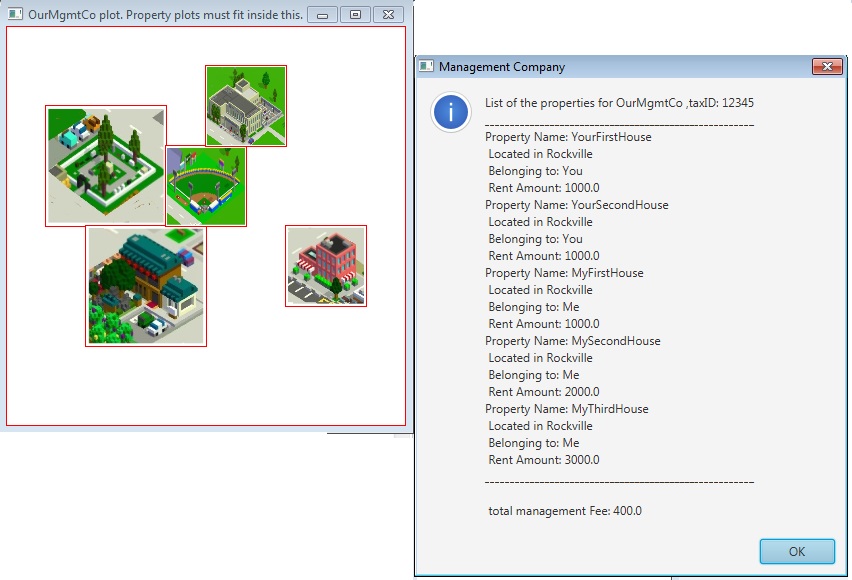
***Add property information - unsuccessful: too many properties***

******

***Result of “Max Rent” button Result of “Total Rent” button***

***Result of “List of Properties” button***



**Deliverables / Submissions:**

**Deliverables**

**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?
* What did you struggle with?
* What will you do differently on your next project?
* Include what parts of the project you were successful at, and what parts (if any) you were not successful at.
* 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 |  |  |
|  | * FirstInitialLastName\_Assignment4\_Complete.zip |  |  |
|  | **Program compiles** |  |  |
|  | **Program runs with desired outputs related to a Test Plan** |  |  |
|  | **Documentation file:** |  |  |
|  | * Comprehensive Test Plan |  |  |
|  | * Screenshots for each Junit Test |  |  |
|  | * Screenshots for each Test case listed in the Test Plan |  |  |
|  | * Screenshots of your GitHub account with submitted Assignment# (if required) |  |  |
|  | * UML Diagram |  |  |
|  | * Algorithms/Pseudocode |  |  |
|  | * Flowchart (if required) |  |  |
|  | * Lessons Learned |  |  |
|  | * Checklist is completed and included in the Documentation |  |  |