**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 tes

**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 (a copy constructor and parameterized constructor) and getter and setter methods. 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 (a no-arg constructor, a copy constructor and a parameterized constructor)
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.

**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.

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

1. Constructor **managementCompany** – pass in arguments for the name of the management company, tax Id and management Fee, along with the X, Y, width, and depth of the overall plot that will be subdivided into plots for each property. A *ManagementCompany* object will be created.
2. Constructor **managementCompany** – pass in arguments for the name of the management company, tax Id and management Fee to create a *ManagementCompany* object. A default plot will be created with programmer-determined values into which to subdivide the plots for each property.
3. Method **addProperty (3 versions)** –
   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. Pass in four parameters of types:
      1. String propertyName,
      2. String city,
      3. double rent, and
      4. String ownerName.

Calls *Property* 4-arg constructor.

* 1. Pass in eight parameters of types:
     1. String propertyName,
     2. String city,
     3. double rent,
     4. String ownerName,
     5. int x,
     6. int y,
     7. int width, and

int depth. Calls *Property* 8-arg constructor.

addProperty methods will return the index of the array where the property is added. If there is a general problem adding the property (other than the next three) this method will return -1; if the property is null returns -2; if the array is full returns -3; if the plot for the property is not encompassed by the management company plot or if the plot for the property overlaps any other property’s plot returns -4.

1. Method **totalRent**– Returns the total rent of the properties in the *properties* array.
2. Method **maxRentProp**-returns the String representation of the property within the *properties* array that has the highest fee amount. For simplicity assume that each "Property" object's fee amount is different.
3. Method **maxPropertyIndex**- returns the index of the property within the *properties* array that has the highest fee amount. This method will be private.
4. Method **displayPropertyAtIndex**– pass in the index of the Property object in the *properties* array and return the string representation of the property. This method will be private.

**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 parallel the public 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.**

## Operation

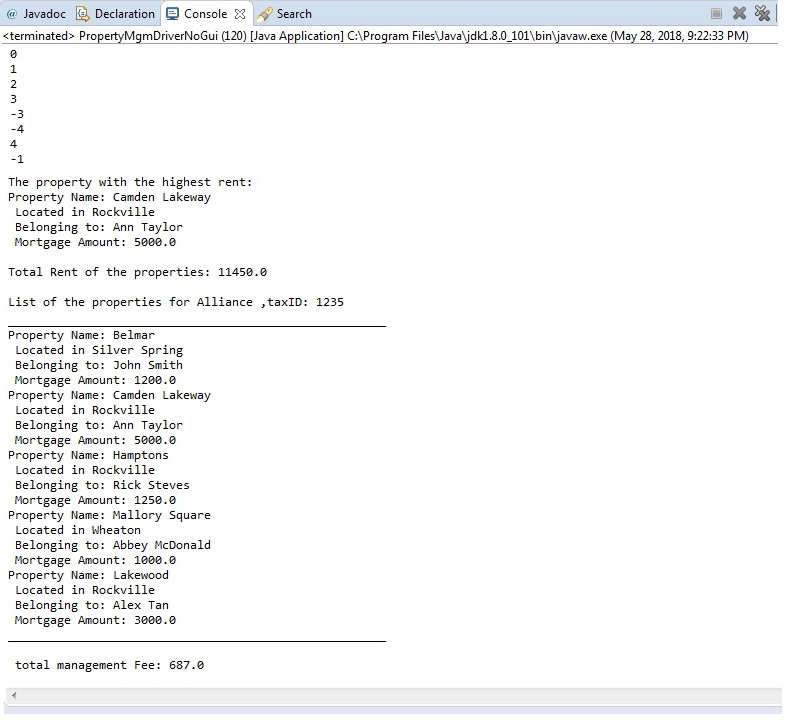
When driver-driven application starts, a driver class (provided) creates a management company, creates rental properties, adds them to the property manager, and prints information about the properties using the property manager’s methods.

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***

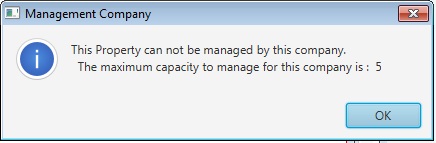
***Add property information - unsuccessful: Mgmt Co Plot does not encompass Property Plot***

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

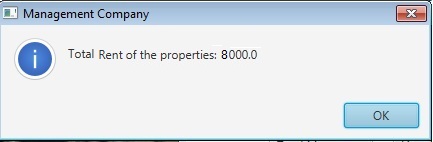
***A screenshot of a social media post

Description automatically generated***

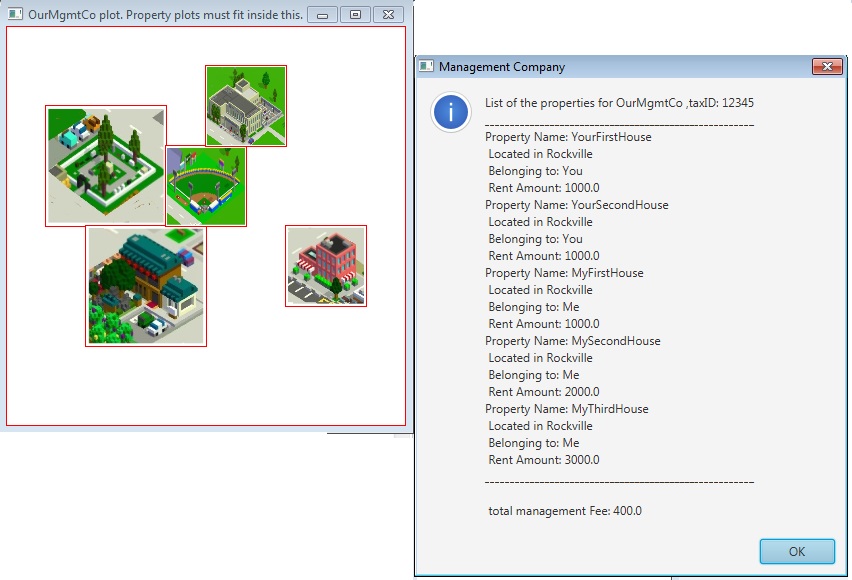
***Add property information - unsuccessful: too many properties***

******

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

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



**Deliverables / Submissions:**

**Deliverables**

**Week 1:** Design

* Turn in a UML diagram that includes box that contains pseudo-code for each of the methods specified in ManagementCompany.java. Your pseudo-code should be part-way between English and java. There is no need to spell out all the details of variable declaration, etc., but by the same token, the pseudo-code needs to have enough detail that a competent Java programmer could implement it.

**Week 2:** Submit two compressed files containing the follow (see below):

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

LastNameFirstName\_Assignment4.zip (a compressed file containing the following)

* + UML Diagram
  + Learning Experience
  + Anything else required by the rubric
  + doc (a directory) containing your javadoc files
  + src (a directory) *contains your (.java) files*
    - File1.java (example)
    - File2.java (example)
    - File\_Test.java (example)

LastNameFirstName\_Assignment4\_**Moss**.zip (a compressed file containing only .java files)

*NO FOLDERS!!*

File1.java (example)

File2.java (example)