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 covered 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 – Plot~~.java~~

You must create this class based on the given Plot Javadoc. You may add additional attributes and/or methods to include in this class. ~~html file; plot.html~~

Formatted: Not Highlight

The class *Plot* will contain:

Attributes:

Formatted: Font: Bold

Formatted: Indent: Left: 0.25"

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.

Methods:

Formatted: Font: Bold

Formatted: Indent: Left: 0.25", No bullets or numbering

1. Constructors
2. ~~1.~~ Getter/Setter methods

2. ~~A `toString` method to represent a Plot instance~~
3. ~~Constructors; Refer to Javadoc for Plot class.~~
4.3. A method named `overlaps` that takes a Plot instance and determines if it is overlapped by the current plot.
4. 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.
5. A `toString` method to represent a Plot instance. A plot should be represented in the following format:
   [x],[y],[width],[depth]
   Notice there is no space between attributes; for the exact format you can refer to the example given in the PlotTestGFA.java file.
5. ~~~~

Data Element class – Property ~~.java~~

You must create this class based on the given Property Javadoc. You may add additional attributes and/or methods to include in this class.

~~.html file; property.html~~

The class *Property* will contain:

Attributes:

Instance variables for property name, city, rental amount, owner, and plot. ~~Refer to Javadoc for the data types of each instance variable.~~

Methods:

1. Constructors
2. Getter/Setter methods
3. `toString` method to represent a Property instance. A property should be represented in the following format:
   [property name],[city],[owner],[rental amount]
   Notice there is no space between attributes; for the exact format you can refer to the example given in the PropertyTestGFA.java file.

1. ~~~~
   2. ~~`toString` method to represent a Property instance.~~

~~3. getter and setter methods.~~
~~4. Constructors:~~

~~a. 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. A second constructor will only have parameters for name, city, rental amount, and owner, and will generate a default x, y, width, and depth.~~
~~5. Refer to Javadoc of the Property class.~~

~~**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. Moved this paragraph below Data Manager class – ManagementCompany.java~~

Formatted: Not Highlight
Formatted: Indent: Left: 1"
Formatted: Indent: Left: 1", No bullets or numbering
Formatted: Highlight
Formatted: Font: (Default) +Headings CS (Times New Roman)
Formatted: Font: Bold
Formatted: Font: (Default) +Headings CS (Times New Roman), Bold
Formatted: Normal, Indent: Left: 0.25", No bullets or numbering
Formatted: Indent: Left: 0.5"
Formatted: Font: 11 pt
Formatted: Indent: Left: 0.5"
Formatted: Bulleted, Indent: Left: 0.5"
Formatted: Bulleted, Indent: Left: 1", No bullets or numbering, Widow/Orphan control, Adjust space between Latin and Asian text, Adjust space between Asian text and numbers
Formatted: Font: (Default) +Headings CS (Times New Roman)
Formatted: Normal, Indent: Left: 0.25", No bullets or numbering, Widow/Orphan control, Adjust space between Latin and Asian text, Adjust space between Asian text and numbers
Formatted: Normal, No bullets or numbering, Widow/Orphan control, Adjust space between Latin and Asian text, Adjust space between Asian text and numbers
Formatted
Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Indent at: 0.5"

Data Manager class – ManagementCompany~~.java~~

You must create this class based on the given ManagementCompany Javadoc. You may add additional attributes and/or methods to include in this class.

The class *ManagementCompany* will contain:

Attributes:

1. Instance variables for ManagementCompany name, Tax Id, management fee percentage.
2. Constant class variables:
    a. MAX_PROPERTY: a constant set to 5 – the max number of properties a management company can have.
    b. MGMT_WIDTH: a constant set to 10 – the width of the management company's plot
    c. MGMT_DEPTH: a constant set to 10 – the depth of the management company's plot
3. Instance array variable named properties that stores the properties of a management company.
4. Instance variable named plot of type Plot that defines the plot of the management company.
5. Instance variable named numberOfProperties that stores the current number of properties of a management company.

Methods:

~~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.~~

~~Do we need all this detail here? Just refer to Javadoc~~
~~It will contain instance variables of name, taxId, mgmFeePer, 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 getter and setter methods:~~

1. Constructors ~~(refer to Javadoc for more details)~~
2. Getter/Setter methods
~~1.~~
~~2.~~ Method addProperty - ~~(~~**~~overloaded 3 versions~~**This is an overloaded method which has multiple versions. In each version you should call an appropriate existing overloaded method if possible. This method~~):~~
~~2.1. Method addProperty version 1:~~
~~2.1.1. 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.~~
~~2.2. Method addProperty version 2:~~
~~2.2.1. Pass in four parameters of types:~~
- ~~String propertyName,~~
- ~~String city,~~
- ~~double rent,~~
- ~~String ownerName.~~

~~2.2.2. Calls *Property* copy constructor.~~
~~2.3. Method addProperty version 3:~~
~~2.3.1. Pass in eight parameters of types:~~

Formatted: Font: Italic
Formatted: Font: 12 pt
Formatted: Font: 12 pt
Formatted: Font: 12 pt
Formatted: Font: 12 pt
Formatted: Font: Bold
Formatted: Font: (Default) Times New Roman, 12 pt, Strikethrough
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Indent: Left: 0.25"
Formatted: Font: (Default) Times New Roman, 12 pt, Not Bold
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Indent: Left: 0.25"
Formatted: Indent: Left: 0.25"
Formatted: Font: (Default) Times New Roman, 12 pt, Not Highlight

- ~~String propertyName,~~
- ~~String city,~~
- ~~double rent,~~
- ~~String ownerName,~~
- ~~int x,~~
- ~~int y,~~
- ~~int width~~
- ~~int depth.~~

~~2.3.2. Calls *Property* copy constructor.~~

~~2.4.~~3. ~~addProperty methods~~ will return ~~the index of the array where the property is added.~~ one of the following values depending on success or failure of the adding the property:

- If there is a problem adding the property, this method will return
  - ~~-1~~ If the array is full, it will return -1
  - ~~-2~~ If the property is null, it will return -2
  - ~~-3~~ If the plot for the property is not encompassed by the management company plot, it will return -3
  - ~~or~~
  - ~~-4 i~~If the plot for the property overlaps any other property's plot, it will return -4.
- Otherwise if the property is successfully added, it will return the index of the array where the property was added

~~3.~~4. Method getTotalRent ~~**totalRent**~~ – This method accesses each "Property" object within the *properties* array, sums up the property rent and returns the total amount. ~~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 getHighestRentPropperty ~~**maxRentProp**~~ - Returns the Property object ~~property~~ with the highest rent amount 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 removeLastProperty - Removes(nullifies) the LAST property in the *properties* array
7. Method isPropertiesFull - Checks if the *properties* array has reached the maximum capacity
8. Method getPropertiesCount - Returns the number of existing properties in the array
9. Method isMangementFeeValid - Checks if the management company has a valid (between 0-100) fee

~~5.~~

~~6.~~10. Method toString - R~~r~~eturns 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~~

Formatted: Font: (Default) Times New Roman, 12 pt, Not Highlight
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted
Formatted: Font: (Default) Times New Roman, 12 pt, Not Highlight
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Not Highlight
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Not Highlight
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Not Highlight
Formatted: Font: (Default) Times New Roman, 12 pt, Not Highlight
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, No underline, Font color: Text 1, Pattern: Clear
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Italic
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Strikethrough
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Not Highlight
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Italic
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Italic
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Not Bold
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Indent: Left: 0.25"

~~total management Fee: 294.0~~

List of the properties for Railey, taxID: 555555555

Almost Aspen,Glendale,Sammy Smith,4844.0
Ambiance,Lakewood,Tammy Taylor,4114.0
Bear Creek Lodge,Peninsula,Bubba Burley,4905.0
Sunsational,Beckman,BillyBob Wilson,2613.0
Mystic Cove,Lakepointe,Joey BagODonuts,5327.0

total management Fee: 1308.18

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

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.

~~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

- For each class listed above, a corresponding GFA test has been provided. GFA (Good Faith Attempt) is the minimum set of requirements for the project. Run each provided JUnit test file and ensure that all tests succeed. **Do not modify any of these JUnit tests files, since the instructor will be using the original file(s).**
- For each assignment class that you create, you must create a JUnit test file. Name your test file as the following format: [classname]TestStudent; for example; PlotTestStudent
- Make sure your test files cover as much as possible test cases. Ensure your test cases all succeed. Since the instructors will be using their own JUnit test files that thoroughly covers each public method. If you have not tested every single method, your chance of failing a test case would be high.
- Make sure to test each constructor.
- You can use the provided GFA test to review test cases and in particular the `toString` method.

~~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.~~

Formatted: Indent: Left: 0.25", Adjust space between Latin and Asian text, Adjust space between Asian text and numbers

Formatted: Not Highlight

Formatted: Strikethrough

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Font: (Default) Courier New

Formatted: List Paragraph, Indent: Left: 0", Right: 0", Space Before: 0 pt, Tab stops: Not at 0.75"

Formatted: Bulleted, Left, Indent: Left: 0.75", Right: -0.13", Space Before: 6 pt, Line spacing: single, Tab stops: 0.75", List tab

Assignment Details

- 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.~~according to the provided Plot Javadoc.
-
- Write a PlotTestStudent JUnit test class that has a test method for each public method of the Plot.java except the setUp and tearDown methods.
- Write a Data Element Class named **Property** according to the provided Property Javadoc.
- Write a PropertyTestStudent JUnit test class that has a test method for each public method of the Property.java except the setUp and tearDown methods.
- Write a Data Element Class named **ManagementCompany** according to the provided ManagementCompany Javadoc.
- Write a ManagementCompanyTestStudent JUnit test class that has a test method for each public method of the ManagementCompany.java except the setUp and tearDown methods.

A Graphical User Interface (GUI) is provided using JavaFX. Do not modify this file. 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 of your project in Eclipse.** The images do not need to display in order for the GUI to continue running. When the GUI starts a window is created as in the following screen shots which allows the user to enter applicable data and display the resulting property. The GUI will use the same classes and methods for their operation.

- ~~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.~~

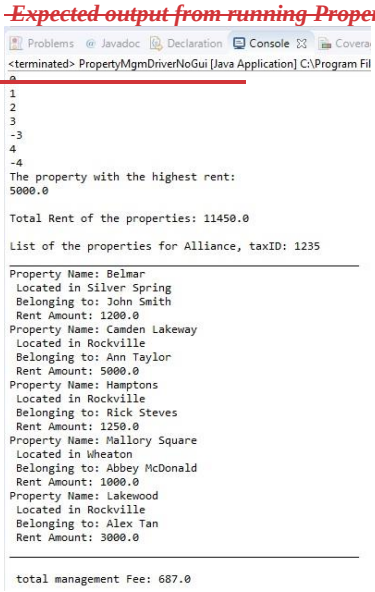
~~When the GUI driven application starts (provided), a window is created as in the following screen shots which allows the user to enter applicable data and display the resulting property. The driver and the GUI will both use the same classes and methods for their operation.~~

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

Formatted: Font: 12 pt
Formatted: Font: 12 pt, Bold
Formatted: Font: 12 pt
Formatted: Font: Bold
Formatted: Font: 12 pt
Formatted: Font: 12 pt, Bold
Formatted: Font: 12 pt
Formatted: Normal, No bullets or numbering
Formatted: Indent: Left: 0.5", No bullets or numbering
Formatted: Font: 12 pt
Formatted: Font: 11 pt
Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0.25" + Indent at: 0.5"
Formatted: Font: 11 pt
Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0.25" + Indent at: 0.5"

Examples

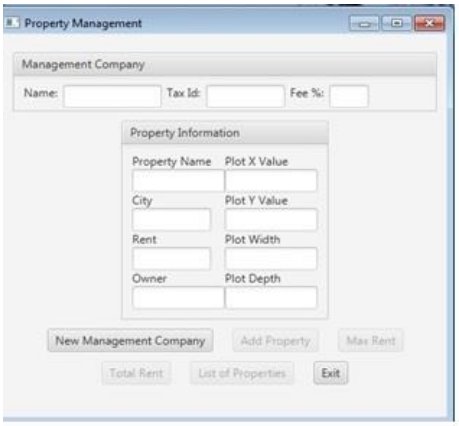
~~Expected output from running PropertyMgmDriverNoGui.java~~



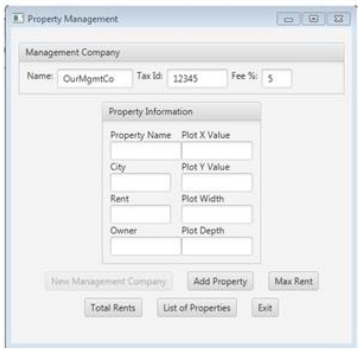
Expected output from running with GUI:

PropertyMgmGui.java at startup

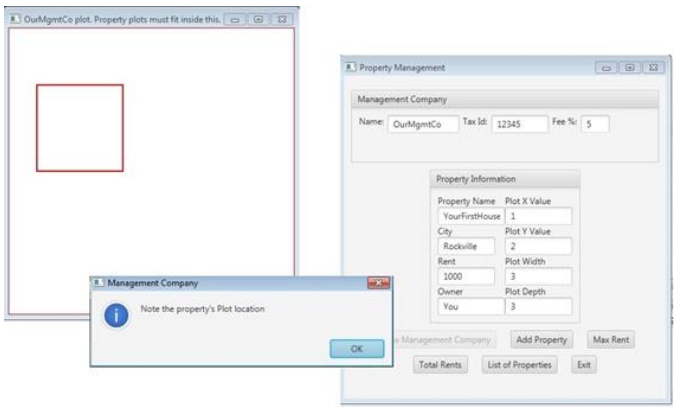
Formatted: Normal, Indent: Left: 0"



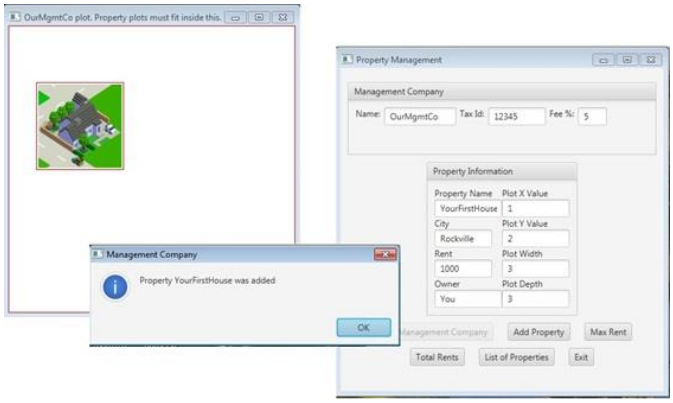
Add Management Co Info (Note Mgmt. Co Plot)



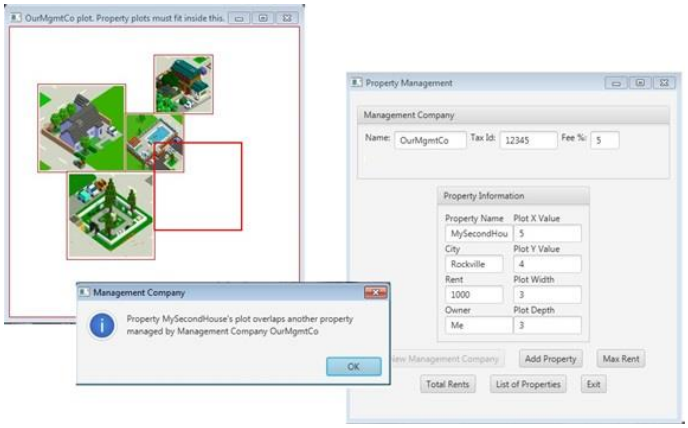
Add property information - the Plot outline



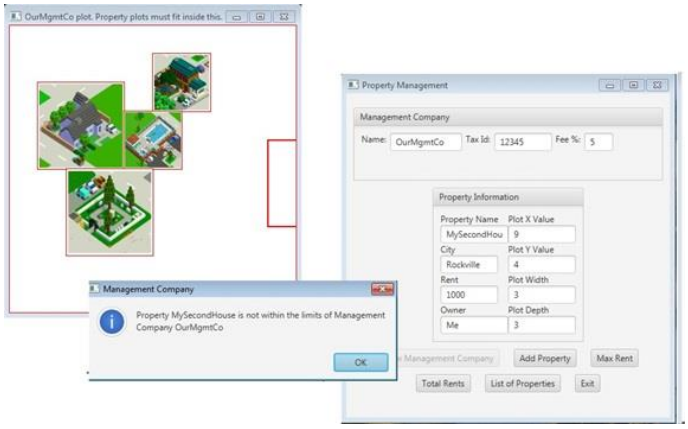
Add property information - successful addition



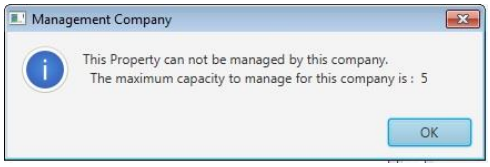
Add property information - unsuccessful: overlaps



Add property information - unsuccessful: Mgmt Co Plot does not encompass Property Plot
Note: red rectangle's width extends to right of window.

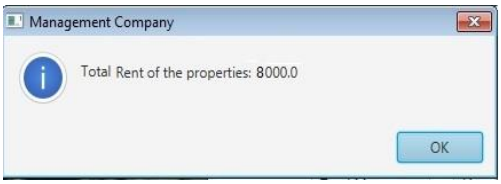


Add property information - unsuccessful: too many properties

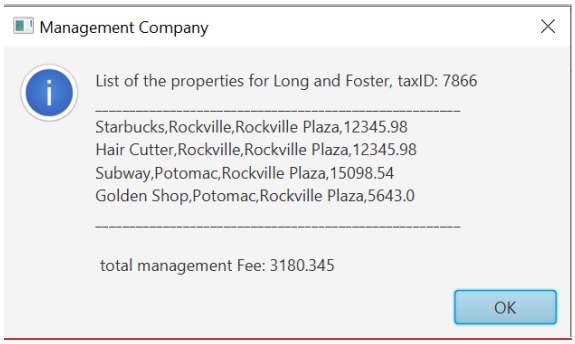


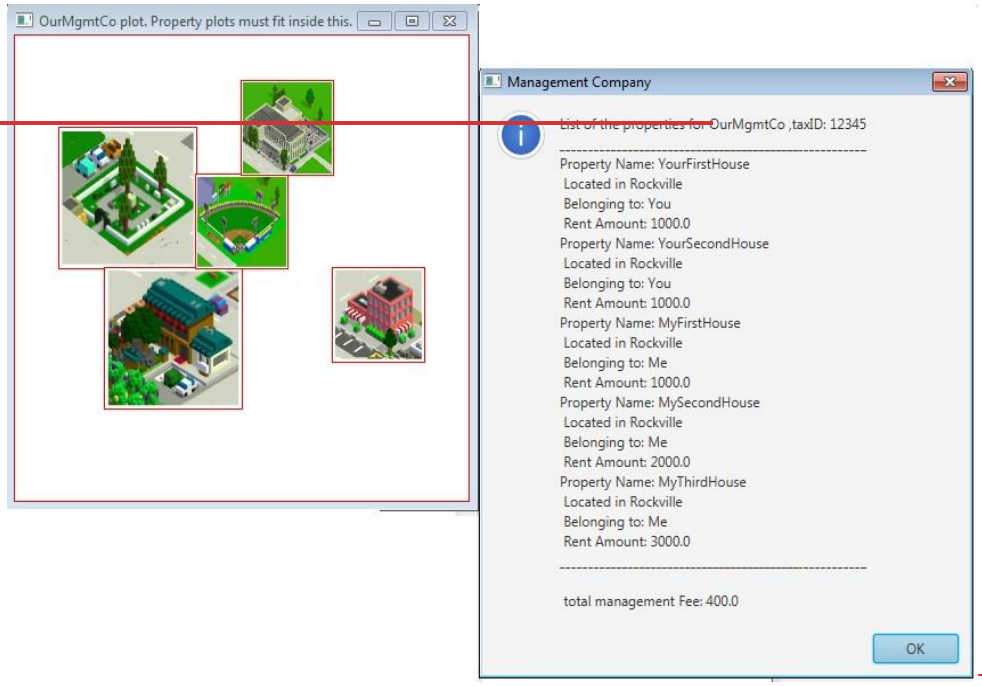
Result of "Max Rent" button

Result of "Total Rent" button



Result of "List of Properties" button



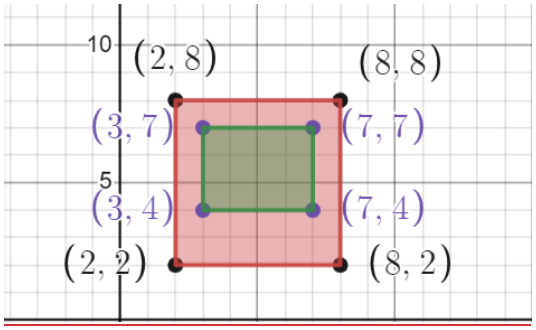


Below you can find examples of overlap and encompass methods; I have used https://www.desmos.com/calculator to plot and create graphs below.

Formatted: Font: 14 pt

Formatted: Font: 14 pt

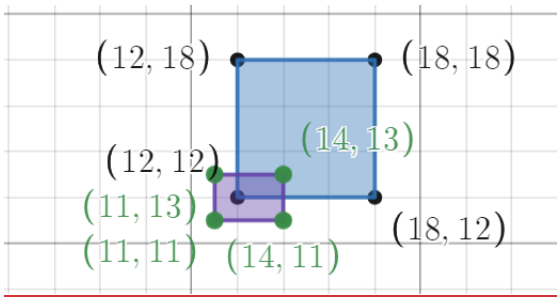
Formatted: Font: 14 pt




```
plot1 = new Plot(2,2,6,6);
plot2 = new Plot(3,4,4,3);
```

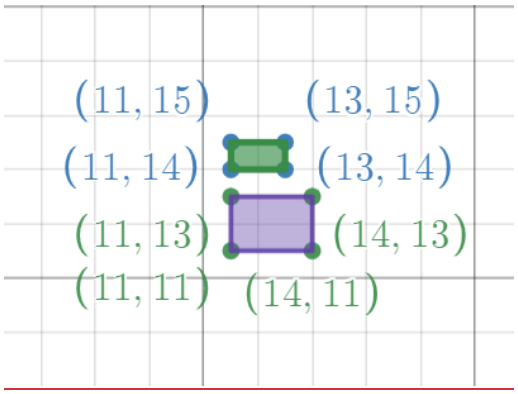
plot2(green plot) is entirely inside plot1(pink plot), therefore it overlaps.;

plot1 is also contained in plot2.



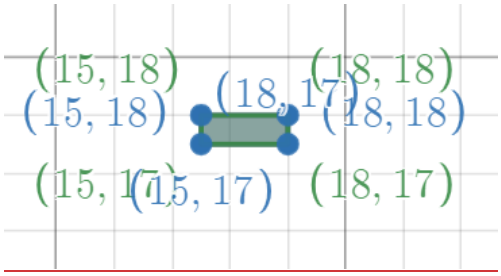

```
plot1 = new Plot(12,12,6,6);
plot2 = new Plot(11,11,3,2);
```

plot2(purple plot) overlaps the lower left corner of plot1(blue).



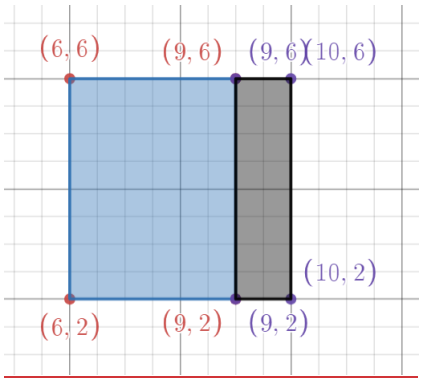

```
Plot1 = new Plot(11,11,3,2);
Plot2 = new Plot(11,14,2,1);
```

Plot2 does not overlap plot1.




```
Plot1 = new Plot(15,17,3,1);
Plot2 = new Plot(15,17,3,1);
```

Plot2 is exactly same as plot1.




```
Plot1 = new Plot(6,2,3,4);
Plot2 = new Plot(9,2,1,4);
```

Plot1 and plot2 share an edge, they do NOT overlap.

Deliverables

Deliverables / Submissions and Deliverable format:

- The Java application must compile and run correctly, otherwise project grade will be zero.
- The detailed grading rubric is provided in the assignment rubric excel file.
- Your source code should contain proper indentation and documentation.
- Documentation within a source code should include
  - additional Comments to clarify a code, if needed
  - class description comments at the top of each program containing the course name, the project number, your name, the date, and platform/compiler that you used to develop the project, for example:

```
/*
 * Class: CMSC203
 * Instructor:
 * Description: (Give a brief description for each Class)
 * Due: MM/DD/YYYY
 * Platform/compiler:
 * I pledge that I have completed the programming
 * assignment independently. I have not copied the code
 * from a student or any source. I have not given my code
 * to any student.
   Print your Name here: _____
*/
```

~~**Deliverables / Submissions:**~~

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

Note: Only submit the files that are created/modified by per requirement. DO NOT submit the files that are already provided for you.

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

- **FirstInitialLastName_Assignment4_Complete.zip**, a compressed file in the zip format, with the following:
  - src folder:
    - Plot.java
    - Property.java
    - ManagementCompany.java

JUnit Test Files:

- PlotTestSudent.java
- PropertyTestSudent.java
- ManagmentCompanyTestSudent.java

Formatted: Font: 12 pt
Formatted: Tab stops: 0.45", List tab + Not at 0.25" + 0.5"
Formatted: Tab stops: 1.2", List tab + Not at 1"
Formatted: Tab stops: 0.45", List tab + Not at 0.25"
Formatted: Font: 12 pt
Formatted: Font: 12 pt
Formatted: Font: 12 pt
Formatted: Font: 12 pt
Formatted: Tab stops: 0.2", List tab + Not at 0" + 0.5"
Formatted: Font: 12 pt, Border: : (No border)
Formatted: No underline
Formatted: Font: 12 pt
Formatted: Indent: Left: 0.5", Tab stops: 0.9", List tab + Not at 0.5"
Formatted: Font: 12 pt
Formatted: Indent: Left: 0.95", Tab stops: 1.2", List tab + Not at 1"
Formatted: Font: 12 pt
Formatted: Font: 12 pt, Bold
Formatted: Font: Bold
Formatted: Indent: Left: 0.95", No bullets or numbering
Formatted: Font: 12 pt
Formatted: Indent: Left: 0.95", Tab stops: 1.2", List tab + Not at 1"
Formatted: Font: 12 pt
Formatted: Font: 12 pt

- Word document that includes (use provided template):
  1. UML Class Diagram for all classes
  2. Screenshots:
     a. Screen snapshots of the GUI with several properties (similar to screenshots in Assignment Description).
     b. Screen shot of src folder files in your GitHub repository
  3. If you have added any public methods in addition to the ones listed in the provided Javadoc, you must submit an updated version of your Javadoc.
  4. Lessons Learned: Provide answers to the questions listed below:
     a. Write about your Learning Experience, highlighting your lessons learned and learning experience from working on this project.
     b. What have you learned?
     c. What did you struggle with?

- **FirstInitialLastName_Assignment4_JavaFiles.zip**, a compressed file containing one or more Java files **(This folder SHOULD NOT contain any folders and it SHOULD contain Java source file only that are created/modified by you per requirement.)**
  - Plot.java
  - Property.java
  - ManagementCompany.java
  - PlotTestSudent.java
  - PropertyTestSudent.java
  - ManagmentCompanyTestSudent.java

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

~~**Note**: Only submit the files that are created/modified by per requirement. 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:*~~

Formatted: Indent: Left: 0.5", Tab stops: -0.25", List tab + Not at 0.25" + 1"

Formatted: Indent: Left: 0.75"

Formatted: List Paragraph, Indent: Left: 1.25"

Formatted: Indent: Left: 0.75"

Formatted: Indent: Left: 1.25"

Formatted: Indent: Left: 0.75", No bullets or numbering

Formatted: Font: 12 pt

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

2nd zip file: FirstInitialLastName_Assignment4_javaFiles.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 that are created/modified by per requirement.

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 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
1.	**Assignment files:**		
	• FirstInitialLastName_ Assignment 4_javaFiles.zip		

	• ~~FirstInitialLastName_Assignment4_Complete.zip~~		
~~**2.**~~	~~**Program compiles**~~		
~~**3.**~~	~~**Program runs with desired outputs related to a Test Plan**~~		
~~**4.**~~	~~**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~~		

Formatted: Don't adjust space between Latin and Asian text, Don't adjust space between Asian text and numbers