**CS/COE 1501 Project 3**

**Goal:**

To explore an advanced application of priority queues in order to gain a deeper understanding of the data structure.

**High-level description:**

You will be writing a basic application to help a user select an apartment to rent. You will write a menu-based user interface driver program (to be run in the terminal, no GUI), but most of the project will be in implementing a priority queue-based data structure. You should write a PQ-based data structure that stores objects according to the relative priorities of two of their attributes, making it efficient to retrieve objects with the minimum or maximum value of either attribute (whether each attribute should be retrieved by mix/max is defined at initialization). Your data structure should further be indexable to allow for efficient updates of entered items. You will want users to be able to enter details about apartments that they are considering renting. The user should then be able to efficiently retrieve the apartment with the highest square footage or lowest price. You will assume that the user is looking for apartments in multiple different cities. These retrievals should be possible on the set of all entered apartments or on the set of all apartments from a specific city (e.g., "lowest price in Pittsburgh", "most square footage in San Francisco").

**Specifications:**

1. First you must create a class to store data about apartments. Specifically, this class must contain the following information:
   * A street address (e.g., 4200 Forbes Ave.)
   * An apartment number (e.g., 3601)
   * The city the apartment is in (e.g., Pittsburgh)
   * The apartment's ZIP code (e.g., 15213)
   * The price to rent (in US dollars per month)
   * The square footage of the apartment
2. You must write a terminal menu-based driver program (again, no GUI). Specifically, your driver must present the user with the following options:
   * Add an apartment
     + This will (one at a time) prompt the user for all of the above-listed attributes to keep track of
   * Update an apartment
     + This option will prompt the user for a street address, apartment number, and zip code of an apartment, and then ask the user if they would like to update the price of the apartment.
   * Remove a specific apartment from consideration
     + This option will prompt the user for a street address, apartment number, and zip code of an apartment to remove from the data structure (e.g., if it is no longer available for rent)
     + Note that this mean you will need to support removal of apartments other than the minimum price or maximum square footage
   * Retrieve the lowest price apartment
   * Retrieve the highest square footage apartment
   * Retrieve the lowest price apartment by city
     + This option will prompt the user to enter a city and then return the lowest priced apartment for that city.
   * Retrieve the highest square footage apartment by city
     + This option will prompt the user to enter a city and then return the biggest apartment for that city.
3. Retrieval operations should not remove the apartment with minimum price or maximum square footage from the datastructure, just return information about that apartment. Apartments should only be removed via the "remove a specific apartment from consideration" menu option.
4. To ensure efficiency of operations, you must base your data structure around the use of heaps with indirection. Note that operations on either attribute (e.g., retrieve minimum price, retrieve minimum mileage) should be efficient. Updates and removals should also be efficient. Take care in selecting your approach to the indirection data structure to account for the types of keys you will need to store and the type and number operations that you will need to perform on them.
5. Because this project requires you to make a number of decisions about how to implement its requirements, you will need to write a documentation file explaining your implementation, and justifying your decisions. Name this file "documentation.txt". Be sure to describe your carefully document your approach to ease the effort required to trace through your code for grading. Be sure to include descriptions of the runtime and space requirements of your approach and use them in your justification of why you think your approach is the best way to go. If you can maintain runtimes of PQ operations that were discussed in lecture (e.g., constant to find; logarithmic to insert, remove, or update) for this situation you should do so. If you cannot, you should describe the particular aspects of this situation that keep you from being able to do so.

**Submission Guidelines:**

* **DO NOT SUBMIT** any IDE package files.
* You must name the primary driver for your program AptTracker.java.
* You must be able to compile your game by running "javac AptTracker.java".
* You must be able to run your program with "java AptTracker".
* You must document and justify your approach in "documentation.txt".
* You must fill out info\_sheet.txt.
* Be sure to remember to push the latest copy of your code back to your GitHub repository before the the project is due. At the deadline, the repositories will automatically be copied for grading. Whatever is present in your GitHub repository at that time will be considered your submission for this project.

**Additional Notes/Hints:**

* You are free to use code provided by the book authors in implementing your solution. It is up to you to decide if it would be easier to modify the provided code to meet the requirements of this project or if it would be easier to start with a clean slate with all of your own code.

**Grading Rubric**

* Adding an apartment works properly: 10
* Updating an apartment works properly: 10
* Removing an apartment works properly: 15
* Retrieval for all apartments works properly: 10
* Retrieval for a given city works properly: 15
* Overall efficiently of operations: 15
* Validity of justifications: 15
* Menu-based driver program works properly and has appropriately labeled options: 5
* Assignment info sheet/submission: 5