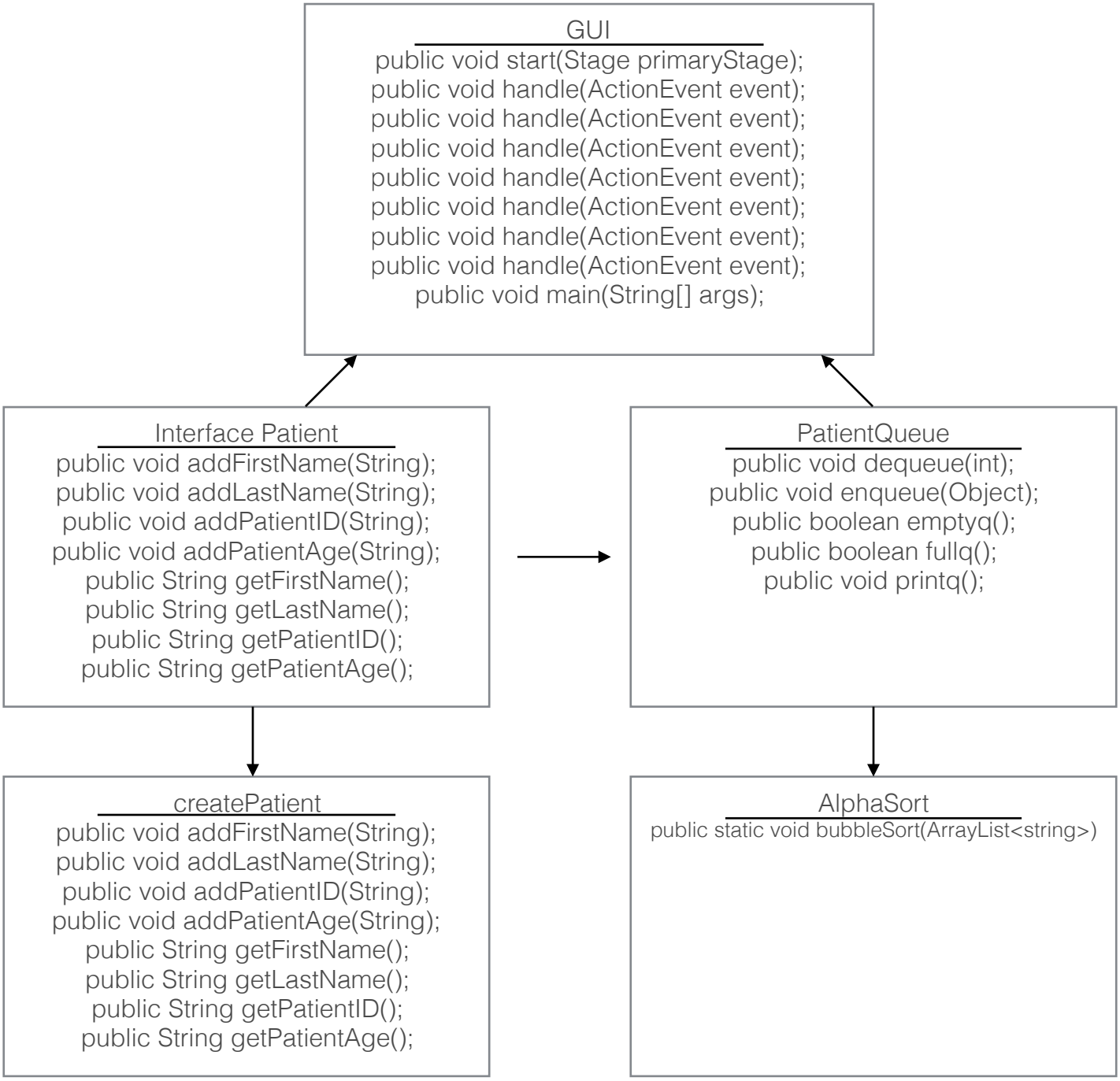
Joseph Medina
5/5/16
Design Document

UML Diagram



Joseph Medina
5/5/16
Design Document

Data Fields

For my program, I included only four data fields: Patient first name, patient last name, patient ID, and patient age. Due to time constraints, I was unable to implement the extra data fields I had in mind: patient DOB, number of times patient has visited, and patient middle name. If I had the time, I would these extra data fields would allow me to refine my searching and deleting functions. For example, with my current system, if you have two patients with the same exact first/last name, the queue will operate similar to how you would except a queue: the last patient that was added will be deleted/ searched first, and the first patient that is added will be deleted/searched last. If I were able to implement an extra field, say DOB - I could avoid this problem as the likelihood of two patients with the same exact first name and last name and data of birth would be greatly decreased.

Data Structures

For my program, the data structure I chose to utilize was the queue. I decided to utilize this data structure because of its attribute of having a FIFO structure(first in, first out.) Specifically for a hospital system this proves useful as it allows you to prioritize the first patients that check in the system. Considering all patients have the same level of urgency(which we know in the real world is not true,) the queue allows the doctors and nurses to prioritize the patients that arrived and checked in first so that we can dequeue these patients. Note: Since I created a patient interface, I allowed for future expandability in my program; that is, my program could be expanded to support multiple different patient types such as head trauma patients, cardiovascular patients, etc. Each of these patients would have their own queue which makes more sense for the real world as it wouldn't make sense to have a patient coming for a check up to be in the same queue as a patient with head trauma.

Sorting Algorithm

The sorting algorithm that I decided to utilize was the bubble sort algorithm. Bubble sort is an efficient algorithm that has a constant average, best, and worse case scenario: The big O notation: On^2.

Some disadvantages of the bubble sort is that it becomes inefficient when it comes to larger arrays. This isn't much of a concern for this specific use

case because doctors and nurses should be moving through the queue quickly enough that we don't have to do n-1 comparisons for a large value of n. A more efficient algorithm to use could be a counting sort and instead of sorting the patients by first name, sort the patients by patient ID number. This would be very efficient with Big O notation being: On.

Assumptions

An assumption I made is that the person using the program will always add a patient before utilizing any other function. Due to time constraints, I was unable to implement an else statement if the user searches/deletes a name that doesn't exist in the list. Another assumption that I made in my project is that every patient has the same value of urgency. I implemented my program using a patient interface for future expandability. In its current form, a "general" patient is being created in which all patients being created and added to the queue share the same priority. An example of future expandability would be to create different types of patients; createCancerPatient, createHeadTraumaPatient, etc, are all example classes that could be created that would implement the same types of information: first name, last name, patient ID, and patient age. These different types of patients would have their own queues since the queues are created using the data type of the type of patient. So, we would have a separate queue for cancer patients, head trauma patients, etc.