James Murphy
CPSC 240 / Object-Oriented Analysis & Design
Programming Project 1 / Plan

`class Patient`

`member variables`

- `firstName` – the patient's first name, entered by staff member at check-in desk
- `lastName` – the patient's last name, entered by staff member at check-in desk
- `triagePriority` – an integer ranking from least severe (`000`) to most severe (`999`) patient conditions, entered by staff member at check-in desk; ties are broken by `arrivalTime` comparison (earliest wins)
- `patientID` – a unique six-digit integer generated by the software (`000001` – `999999`), is assigned upon insertion to the queue (initialized as 0)
- `arrivalTime` – a timestamp generated by the system clock upon creation of patient entry

`methods`

- accessors – `get()` for each field for use in comparisons between fields or display to screen
  - `getFirstName()`
  - `getLastName()`
  - `getTriagePriority()`
  - `getPatientID()`
  - `getArrivalTime()`
  - `setpatientID()` – called when a `Patient` object is ready to have its `PatientID` field set
- constructor – `Patient()` takes arguments the following arguments and fills the fields of a `Patient` object; will also call system clock and retrieve a unique 6-digit number
  - `firstName`
  - `lastName`
  - `priority`
- output – `toString()` transforms patient information into a string for File I/O

`public class PatientQueue`

`member variables`

- `length` – an integer counter (starting from 0) which tracks how many people are waiting for medical attention; increments upon insertion of `Patient` object and decrements upon removal. If this value ever negative, an error has occurred.
- `patientIDCounter` – an integer counter (starting from 1) whose value is assigned to a `Patient`'s `patientID` field, then incremented afterward. This will overflow to `1` upon `999999 + 1`
- `priorityQueue` – a queue object of type `PriorityQueue<Patient>` for storing the `Patient` objects (maybe this will be a `List` of patients and enqueueing will be appending to the list and other accesses will be O(n) searches of the list? This would be more useful for sorting by different metrics, but for large numbers of patients could

cause poor performance. What's a typical number of patients in an ER anyway?)

`methods`

- `dequeuePatient()` – remove frontmost patient (and output as return value) from list once they begin receiving care; decrements `length`
- `enqueuePatient()` – add a patient to the queue and fill their `patientID` field; increments `length`
- `saveState()` – runs upon program exit; saves `length` and `patientIDCounter` values to a file, and calls `toString()` ~~method from `Patient` class on each `Patient` to write them to a separate file~~
- `PatientQueue()` – constructor without arguments starts a `PatientQueue` from default values
- `PatientQueue()` – constructor with arguments resumes a `PatientQueue` from a previous session
  - `numericState` – a parameter of type `String` which is the filename that stores `length` ~~and `patientIDCounter`~~
  - `patientState` – a parameter of type `String` which is the filename that stores patient data
- `positionOf()` – takes an argument of `String` type which should contain the name of a patient and returns how many people are ahead of that patient in the queue. Throws an exception if the requested patient does not exist.
- `formatPatientData()` – returns an `ArrayList<String>` where each element is patient information. The list is sorted according to patient priority.

The underlying implementation of the queue will largely depend on A) whether I am allowed to use implementations of the Queue interface and B) whether a dumb List-based implementation would be unacceptable in terms of performance, which in turn depends on expected throughput of the ER.