(Where and When? Tuesday, 25 April)
5:30-7:00. Statler Aud: last name begins with M..Z.

7:30-9:00. Statler Aud: last name begins with A..L.

° Do you have permission to have a quiet room or more time? See the note about P2 on the course website.

° Review session: Sun, 23 April, 1-3pm, Olin 255

Note: If a conflict prohibits you from taking the exam at the assigned time, you MUST complete assignment P2Conflict on the CMS before the end of Wed, 19 April. (If you can't take it at the assigned time, you may take it at the other time; we just need to know about it.) P2Conflict requires you to give us details.

Read the statement about Prelim 2 on the course exam webpage for complete details: cs.cornell.edu/courses/CS2110/2017sp/exams.html.

The test covers material through lecture 21, on 18 April. Details appear below.

You are expected to know everything that was required for Prelim 1. Look at the Prelim 1 handout.

To prepare (1) Practice writing programs in Eclipse,
(2) Study Piazza note @96 (Suppl. Study Material),
(3) Memorize definitions/principles,
(4) Study lecture slides,
(5) Attempt past prelims on the course website, and
(6) Read the optional text or other text.

The overall length and balance of the exam will be similar to past prelim 2s, but the exam covers only topics presented on this page. Ignore past prelim-2 questions that touch on topics that are not listed below.

Topics to be covered on Prelim 2

0. Everything needed for Prelim 1.

1. Loops and recursion. Use of invariants to develop loops and argue about their correctness. We used these on searching/sorting algorithms and graph algorithms. See an attachment to Piazza note @96.

2. Algorithmic complexity. Big-O complexity notation and the associated definitions. Understand how to derive a big-O complexity formula for an algorithm, best-case/worst-case/average complexity, the notion that what this counts is some sort of "operation we care about" and not every line of code, etc. See an attachment to pinned Piazza note @96.

3. How a type can be defined using an interface.

4. Searching and sorting. Know these algorithms: Linear search, binary search, insertion sort, selection sort, mergesort, partition algorithm of quicksort, quicksort, heapsort. "Knowing" means: being able to develop them given their specifications, using high-level statements for the parts that massage the array (e.g. "merge sorted partitions b[h..k] and b[k+1..n]"). If you don't understand what we mean, look at the appropriate lecture notes. Know the average- and worst-case complexity of these algorithms. Understand min-heaps, how a min-heap can be used to implement a priority queue, and how a max-heap is used in heapsort.

5. Hashing. Hashing as presented in recitation. The relationship between methods equals and hashcode.

6. Interfaces. Review the interface lecture materials and make sure you understand the ideas.

7**. Java Collections framework.** Be familiar with the standard operations that are supported by common data structures implementing Collection<T>, List<T>, Set<T>, Map<K,V>, ArrayList<T>, etc.

8. Trees: Trees, binary trees, data structures for binary and non-binary trees, BSTs. Expression trees and their traversals: preorder, inorder, postorder. *Tree rotations, AVL trees, and parsing are not covered*.

9. Graphs. Kinds of graphs (e.g. planar, sparse, dense). Adjacency matrix vs adjacency list. DFS and BFS, topological ordering, Dijkstra's shortest-path algorithm, *Not spanning trees*. Be able to write DFS and BFS given a specification. Expect questions that involve graphs: be able to tell us which algorithm is the best choice for solving a problem, precisely what that algorithm does, why it would solve a problem, and how costly it might be.

10. GUIs. We will not ask you to write GUI programs. We may ask you to read and understand small sections of code that place components using the usual (JFrame, JPanel, Grid, Box, and layout managers. Know the three steps required to listen to events (see lecture slides).

11. Keep in mind the following:

A. Being able to write correct Java code is critical. We will continue to have coding questions. We plan to grade them with a bit more insistence on correct Java. You may lose credit for code that is long, is inefficient, or reveals a poor grasp of Java features.

B. We expect you to know Java and our coding guidelines —not just the bits and pieces of Java used on slides in class. If there is some aspect of Java that worries you, read about it in Appendix A of the optional text, look in the JavaSummary.ppt, study our Code style guidelines on the course website, google it.

C. Use the powerful built-in Java tools. We give maximum credit for concise, elegant code that doesn't reinvent the wheel. Know how to use standard Java classes like ArrayList, HashSet, and HashMap and know the *basic* methods available for Collections, arrays, Strings, etc.