Where and When?
* 5:30-7:00PM. Statler Aud., if your student ID is even
* 7:30-9:00PM. Statler Aud. if your student ID is odd.
* Do you have permission to have a quiet room or more time? Statler 196 (near Statler Aud.) at 5:30 if possible.
* *Review session:* Sat, 19 Apr, 1-3pm, Kimball B11

Conflict with the assigned time? Contact Megan L. Gatch <mlg34@cornell.edu> With her permission, you can come to the other one. If you cannot make either time, contact Megan immediately.

This handout explains what prelim2 may test.
To prepare (1) practice writing programs in Eclipse, (2) study the material in Piazza note @414, (3) read the textbook, (4) memorize definitions/principles, (5) study lecture slides, and (6) attempt past prelims on the course website:
http://www.cs.cornell.edu/Courses/cs2110/2014fa/exams.html

The overall length and balance of the exam will be similar to past prelim2s, but the exam will cover only topics presented on this page. Ignore past prelim 2 questions that touch on topics not listed below.

You are expected to know everything that was required for Prelim1. Look at the review handout for Prelim1, which can be found on the course website.

Here are some topics that might be covered:

1. Proofs —weak and strong induction. We may ask you to prove things using induction (e.g. that some simple closed form equation is the correct solution to an iterative/recursive equation). There will be an attachment on Piazza note @414 about this.

2. Loops and recursion. Use of invariants to develop loops and argue about their correctness. Use of induction to prove recursive methods correct. We used these on searching/sorting algorithms and graph algorithms. See an attachment to Piazza note @414.

3. Algorithmic complexity. Big-O complexity notation and the associated definitions. You should 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 Piazza note @414.

4. Abstract data types (ADTs) and how they can be defined in Java (using interfaces).

5. Searching and sorting. We have covered a number of sorting algorithms and you need to know them! Linear versus binary search, mergesort, quicksort, heapsort. How they work, complexity, data structures they use. You should be able to write mergesort and quicksort, using high-level statements for the parts that actually massage the array (e.g. "merge sorted partitions b[h..k] and b[k+1..n]"). Understand the min-heap data structure, how it can be used to implement a priority queue, and how it is used in heapsort.

6. Hashing. Hashing as presented in recitation. The relation between functions equals and hashcode.

7. Interfaces. Review the interface lecture materials and make sure you understand the ideas. Be familiar with the standard operations that are supported by common data structures implementing Collection<T>, List<T>, Set<T>, Map<T>, ArrayList<T>, etc.

8. Trees: Binary trees, data structures for binary and non-binary trees, BSTs. Grammars used to define languages. Parsing and parse trees, expression trees and their traversals: preorder, inorder, postorder. Tree traversals. Recursive descent parsing not required.

9. Graphs. Kinds of graphs (e.g. planar, sparse, dense). DFS and BFS, topological ordering, Dijkstra's shortest path algorithm, spanning trees (Kruskal and Prim). Expect questions that involve graphs: be ready 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. You will be not be asked to write GUI programs. We may ask you to read and understand small ones. You should know the three major classes that can contain components (JFrame, JPanel, and Box), what their layout managers are, and how they lay out components on the screen. Also, the three major points in listening 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 questions on coding. We plan to grade them with a bit more insistence on correct Java. We were more relaxed on prelim1. Don't expect a second "free pass" on that on prelim2. On prelim2, you might lose credit for code that is long, is inefficient, or reveals a poor grasp of Java features.

B. We expect you to know Java —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 or look in the JavaSummary.ppt.

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 types like ArrayList, HashSet, HashMap, and know the *basic* preexisting methods available for Collections, Arrays, Strings, etc