**Where and When? Tues, 24 April**
5:30-7:00. Statler Aud: … ..

7:30-9:00. Statler Aud: ….

Review session: Sun, 22 Apr., 1-3pm, TBA

Read the Exams page of the course website, as with P1, for info on conflicts and completing P2Conflict, if necessary, by the end of Wed, 18 April.

cs.cornell.edu/courses/CS2110/2018sp/exams.html.

To prepare:
(1) Practice writing programs (in Eclipse and by hand),
(2) Study JavaHyperText entry "study/work habits",
(3) Memorize definitions/principles,
(4) Study lecture slides,
(5) Attempt past prelims on the course website, and
(6) Consult a staff member about any material you don't understand.

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 that will NOT be covered**: Hashing, Parsing, Anonymous functions (lambdas).

**Topics to be covered on Prelim 2**

The test covers material taught through lecture 20 (12 April 2018).

**0. Everything needed for Prelim 1.** Read the Prelim-1 study guide.

**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.

**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.

**3. 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.

**4. Interfaces**. Review the interface lecture materials and make sure you understand the ideas. Know how a type can be defined using an interface and the relationship to abstract data types. Specifically included are interfaces Comparable, Iterator, and Iterable and how they are used.

5**. 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>, HashMap<K, V>, etc.

**6. Trees:** Trees, binary trees, data structures for binary and non-binary trees, BSTs. Expression trees and their traversals: preorder, inorder, postorder. Know the class invariant for red-black trees and how it impacts performance. *Tree rotations, AVL trees, and parsing are not covered.*

**7. Heaps.** Understand min-heaps and max-heaps, how a min-heap can be used to implement a priority queue, and how a max-heap is used in heapsort.

**8. Graphs**. Kinds of graphs (e.g. planar, sparse, dense). Adjacency matrix vs adjacency list. DFS and BFS, topological ordering, Dijkstra's shortest-path algorithm, 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.

**9. 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).

**10. Generics**. Understand how to make a class or a method generic. Be able to explain why ArrayList <Integer> is not a subclass of ArrayList<Object>.

**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 the appropriate entry in the JavaHyperText, study our Code style guidelines in JavaHyperText.

**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, TreeSet, and HashMap and know the *basic* methods available for Collections, arrays, Strings, etc.