CS2110, Spring 2019. Preparing for Prelim 1

Prelim: 5:30-7:00 Tuesday, 12 March
For students whose Cornell netid begins with two letters in the range
km..zz

Prelim: 7:30-9:00 Tuesday, 12 March
For students whose Cornell netid begins with two letters in the range
aa.kk

The course webpage says what to do if you cannot make the assigned time, for whatever reason. READ THE INSTRUCTIONS ON

http://www.cs.cornell.edu/courses/CS2110/2019sp/exams.html

If you can take the prelim at the assigned time, you don't have to do anything but study and show up.

Review session: Sunday, 10 March, 1:00–3:00. Kimball B11

This handout explains what you have to know for the prelim 1. The course website contains several previous CS2110 prelims. To prepare for the prelim, you can (1) Practice writing Java programs/methods in Eclipse, (2) Read the JavaHyperText, (3) Memorize definitions, principles, (4) Study past prelims, on the course website, and (5) Do what is suggested in the JavaHyperText entry "study/work habits".

In looking at past prelims, if you see a question that is outside the scope of the prelim as defined below, then skip that question. Before asking on the Piazza "is this topic covered on the prelim?" look through this two-page document. If you don't find the answer here, then ask.

Prelim 1 covers material all material in lectures/recitations through **Thursday, 28 Feb.** Here is more detail:

1. Java strong typing. Everything has to be declared before it can be used. The primitive types **int**, **double**, **char**, **boolean** (know the basic operations on them). The corresponding wrapper classes Integer, Double, Character, Boolean. You don't have to know the detailed methods in each wrapper class, but know the two reasons for having wrapper classes (be able to treat a primitive-type value as an object; provide useful static fields and methods). Understand casting between numeric types and the fact that **char** is a numeric type. Autoboxing and unboxing.

2. OO. This is a big one. Master the following:

(a) Declaration of a variable
(b) Declaration of a class and subclass
(c) What fields/methods a subclass object has
(d) The class invariant
(e) Access modifiers public and private
(f) Getter/setter methods
(g) Declarations of functions and procedures
(h) What the name of an object is: Foo@...
(i) Evaluation of a new-expression
(j) Value **null**
(k) Static versus non-static
(l) Constructors: purpose. Principle that superclass fields are initialized first. What the first statement in a constructor body must be. What Java inserts in a class if there is no constructor.
(m) Overriding methods
(n) Overloading method names
(o) Class Object and the class hierarchy. What Object.toString() and Object.equals (Object) return.
(p) The uses of **this** and **super**: fields of **this**, calling other constructors of **this** class, calling constructors of **super** class, calling the superclass's implementation of a method
(q) Casting among class types —downcasting

and upcasting; the latter can be done automatically.

(r) Type of a variable v and its use in determining, say, whether v.m(…) is syntactically legal. Compile-time reference rule.

(s) Reason for making a class abstract; reason for making a method in an abstract class abstract.

(t) Four kinds of variable in Java: field, class variable (static), parameter, local variable

(u) Use of arrays (note: an array is an object): declaration of 1-2 dimensional arrays, length field, how one references an element (e.g. b[i]). Array initializers. Be able to write methods that use arrays, using appropriate syntax.

(v) Simple generic types and their use —e.g. ArrayList<JFrame>, LinkedList<Integer> Java type-checking rules for calling a method that expects a generic type for one of its arguments.

(w) Interface declaration and implementing an interface —what that means. Casting with interfaces.

(x) The Java Collections classes. Basic knowledge of structure —as discussed in recitation.

(y) Knowledge of interface Comparable and its abstract method.

(z) Exception handling: class Throwable; how to throw an exception; the try statement, with its try-block and catch-blocks.

3. Class String. You may be asked to write code that uses class String. Know methods charAt, indexOf, lastIndexOf, contains, substring, length. You are welcome to use other methods too, but we'll test on this subset.

4. Recursion. Know how to write a recursive function. Know the difference between how recursive calls are executed (in terms of placing a frame for a call on the call stack, etc.) and how one understands a recursive function (Understand the body in terms of a recursive call doing what the specification says, not how it gets executed.). Know the steps in executing a method call and be able to execute them yourself, by hand.

5. Linked lists. NOT ON PRELIM 1

6. Loop invariants. Understand a loop in terms of a loop invariant and the four loopy questions: Start (make invariant true)? Stop (invariant together with false loop condition imply result)? Progress (loop body makes progress toward termination)? Invariant (repetend keeps the loop invariant true)? Be able to develop a loop given the precondition, postcondition, and loop invariant. Be able to generalize a precondition and postcondition given as array diagrams to a loop invariant.

7. Testing. Know the material covered in recitation on testing and the material discussed under JavaHyperText entry "testing". Know the basics of JUnit testing as done in Eclipse: the need for the annotation @Test, the use of procedure assertEquals(expected-value, computed-value), etc. See the JavaHyperText entry "JUnit testing".

8. Searching/sorting. Be able to *develop* linear search, binary search in a sorted array, insertion sort, selection sort, partition algorithm of quicksort, quicksort, and mergesort (not the algorithm to merge adjacent sorted segments). Stable versus unstable sorting algorithms.

9. Complexity. Know the definition of f(n) is O(g(n)). Be able to prove simple theorems like n*n + 2 is O(n*n). Be able to look at a simple program and determine its order of execution. Know the basic complexity categories, e.g. O(1), O(log n), O(n), O(n log n), O(n^2), O(n^3), …, O(2^n).