NAME: ________________________________ NETID: ________________________

CS2110 Fall 2013 Final

Write your name and Cornell netid. *There are 5 questions on 14 numbered pages. Check now that you have all the pages. Write your answers in the boxes provided. Use the back of the pages for workspace. Ambiguous answers will be considered incorrect. The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. Good luck! And have a nice winter break!*

Note: Please be careful to use correct Java syntax. Many people lost points on prelim2 for omitted (), ;, {}, etc. We deduct when we notice Java syntax errors!

	1	2	3	4	5	Total
Score	/20	/20	/20	/20	/20	
Grader						

1. (20 points) You've taken a job with Facebook and are working on a new app called "AllAboutMe". The purpose of this application is to show you photos of yourself that were posted by other people. You'll solve the problem in three steps. Step (b) uses the method from step (a), and step (c) uses (b).

Here are two classes used by Facebook to represent data:

```
public class User {
        public String name;
        public Set<User> friends;
        public Set<Photo> photos;
}
```

```
public class Photo {
        public JPG image;
        public Set<String> tags;
}
```

Each User has a name, a set of friends, and a set of photos. Each Photo has an image (a jpeg object) and a set of tags. These tags could include user names, which is the case that interests us.

A comment from the graders: In Java, the Set<T> interface is sometimes supported by types of objects that don't enforce traditional (mathematical) set semantics. In particular, a mathematical set has the rule that even if you insert something multiple times into the set, it only retains one copy. In Java the HashSet<T> class has that semantic. But Java allows various kinds of lists and trees to claim that they implement Set<T> even if they don't actually do so. The idea is that if you as the developer wanted to create a version of a list or a tree that has set behavior, you can do that (e.g. check to see if the object contains x before doing an insert of x). Then your version of a list or a tree would in fact be a set in the mathematical sense and worthy of the claim that it implements Set<T>. Why are we explaining this? Well, in grading problem 1, we looked at your solution to (c) before we graded part (a), so that if you used a HashSet<T>, we didn't insist that you check to see if the set ps already contains the photo. If your Set<T> constructor in (c) used something like an ArrayList<T>, which definitely doesn't check for single-copy semantics, we deducted. We gave a few people a break: they tried to create a new Set<T>, as if Set<T> was a class, not an interface. In fact Set<T> can't be instantiated (you can't instantiate an interface), but we decided that we weren't explicit enough about this in class, so we circled this mistake on part (c) if you made it, but we didn't deduct, and then on part (a) we allowed you to omit the check ps.contains(p), since you probably were assuming that the check was built-in behavior for ps.

(a) [6 points] Write the body of the following method. Assume that who, u and ps are non-null.

```
/** Search the photos in User u to see if any are tagged with name who. If any
  * photo with that tag is not already in set ps, add that photo to ps.*/
public static void collectPhotosOfName(String who, User u, Set<Photo> ps) {

        for (Photo p: u.photos)
          if(p.tags.contains(who))
              // Note that ps is a HashSet (see part c), duplicate adds are ignored!
              ps.add(p);
}
```

(b) [6 points] Write the body the method below, using the method of part (a).

```
/** If u is not in set seen, then:
  * (1) Add u to seen and add any of u's photos that are tagged with who to ps,
  * (2) Recursively do the same with all friends of u. */
public static void recursiveCollect(String who, User u, Set<Photo> ps, Set<User> seen) {

      if (seen.contains(u)) return;
      seen.add(u);
      collectPhotosOfName(who, u, ps);
      for (User f: u.friends)
            recursiveCollect(who, f, ps, seen);

}
```

(c) [7 points] Last, write the body of the method below, which solves the overall problem for which we want a method: Find all pictures tagged with a user' name that that user's friends, their friends, etc., have. Use the method of part (b). *Hint: User u's own photos shouldn't be included!*

```
/** Return the set of all photos that friends of u, their friends, their
  * friends, etc. have that are tagged with u's name. */
  public static Set<Photo> allAboutMe(User u) {

     Set<Photo> ps = new HashSet<Photo>();
     Set<User> seen = new HashSet<User>();
     seen.add(u);
     for (User f: u.friends)
         recursiveCollect(u.name, f, ps, seen);
     return ps;
}
```

d) [2 points] Explain briefly why the method you wrote in (c) won't have infinite recursion.

The key insight is that because we track the users we've seen (including the original User object), we search any given User object at most once. Since Facebook has a finite number of users, at most we will search all of them. Notice that this answer would also apply if someone comes up with a non-recursive solution to the problem as a whole (we saw one or two people who solved the problem iteratively).

2. (20 points) True or false?

a	[✓ T]	F	It is important to provide synchronization for any data that might be updated by more than one thread or that might be updated by some thread and read by other threads.
b	T	[✓ F]	Within a minimum spanning tree for an undirected graph, the path between any pair of nodes is also the shortest path between those two nodes in the underlying graph.
c	[✓ T]	F	If an undirected graph is disconnected, we can form one minimum spanning tree per component, but we can't form a single minimum spanning tree for the entire graph.
d	[✓ T]	F	If an undirected connected graph has v nodes, the minimum spanning tree will have v-1 edges and no cycles.
e	T	[✓ F]	If a field in some class is marked with **static**, it can be initialized but can't later be changed.
f	T	[✓ F]	In a cloud computing setting, one useful tool is a version of a HashMap spread over the nodes in a data center. Such a hash map is called a Distributed Hash Table, or DHT.
g	T	[✓ F]	If a computer has C cores and plenty of memory, and you write a correct, deadlock-free program with T properly synchronized threads (C≥T), it should be T times faster than a single-threaded program, no matter what problem it solves.
h	[✓ T]	F	A min-heap satisfies the invariant that the value in the root node is the minimum among the root value and the values of its child nodes.
i	T	[✓ F]	During execution of a program using a GUI, if we add a component to a Box object, the layout manager will automatically re-arrange everything on the monitor, without us having to call a method to tell it to re-arrange the GUI.
j	T	[✓ F]	When we use a try { something } catch (Exception e) { recovery-code }, if an exception occurs within the "something" block, the recovery-code block will run, after which Java re-executes the instruction or line of code that triggered the exception.
k	T	[✓ F]	Suppose we create an ArrayList<Dog> x, add a single Dog object to x, and then pass x to a method that has a single argument y of type Dog. Java will "auto-unbox" the object we added to x, and when the method runs, y will reference that object.
l	T	[✓ F]	Suppose that d is an instance of Dog and that the *only method* defined in class Dog is bark(). If you set String s = d.toString(); the bark method will automatically be called and its result will be converted to a string and saved in s.
m	T	[✓ F]	In Java, if x and y are declared to be of type double[][] and are both square, NxN matrices, then x = x*y saves the matrix product of x times y into x.
n	T	[✓ F]	If a class includes a recursive method, it is illegal for that method to access static class variables (from the same class), even if they are marked public.
o	[✓ T]	F	If a class includes a static method that needs to access instance variables from the class, it must qualify those accesses by using an explicit object reference.
p	T	[✓ F]	If an ADT specifies that some method m should have complexity O(n log n), and your implementation of that method could have worst-case complexity O(n^3), Java will warn you about this at compile time and it won't be possible to run m until you fix the implementation.
q	T	[✓ F]	If the first line of a JUnit test is an assert statement and you are running that test, the assertion will be *continuously monitored* by Java until the test finishes. If it is ever violated (even temporarily) while the unit test is still running, the unit test will fail.
r	[✓ T]	F	The following is not an infinite loop: int x= 1; while (x > 0) {System.out.print("+"); x= x+1;}
s	T	[✓ F]	One representation of a hash table represents data in a 1-dimensional array and resolves collisions using "linear probing." True or false: For this implementation a good way to delete an item is to set the array element to which it was mapped to null.
t	T	[✓ F]	If thread T1 is a producer and thread T2 is a consumer and they interact through a shared Bounded Buffer object b, neither will ever need to wait when calling the methods in b.

3. (20 points) Suppose we store N distinct strings (e.g. no two strings have equal values) into a binary search tree (BST) with the following fields:

```
public class BSTNode {
    public String value;
    public BSTNode left;
    public BSTNode right;
}
```

(a) [5 points] Write the body of the method shown below

```
/** Return the number of nodes in BST x (return 0 if x is null). */
public static int numberOfNodes(BSTNode x) {
    if (x == null)
        return 0;
   return 1 + numberOfNodes(x.left) + numberOfNodes(x.right);
}
```

b. [6 points] Write the body of the method below. Your solution **should call the method of (a)**

```
/** Using numberOfNodes from (a), return the number of BST nodes in the subtree rooted at x
      that have values smaller than v, using standard string comparison.  */
private static int numberOfSmaller(BSTNode x, String v){
    if (x == null)
          return 0;
    if (x.value.compareTo(v) == 0)
          return numberOfNodes (x.left);
    if (x.value.compareTo(v) > 0)
          return numberOfSmaller(x.left, v);
   return 1 + numberOfNodes(x.left) + numberOfSmaller (x.right, v);
}
```

(c) [9 points] For each of the following questions, circle the best answer

In a BST of height h, the minimum number of nodes would be …

i. h+1
ii. 2^h
iii. h^2

In a BST of height h, the maximum number of nodes would be …

i. 2(h+1)
ii. $2^{h+1}-1$
iii. h^2

In a BST of height h, a worst-case search for a value that is present requires

i. h+1 comparisons
ii. log(h) comparisons
iii. A constant number of comparisons, but the constant depends on the order in which the elements were inserted

4. (20 points) Suppose we are given a directed graph representing the highway system in some country. Vertices are locations, edges represent road segments between them, and each edge has a weight corresponding to the average travel time for that road segment.

Consider the following graph algorithms from class: BFS, DFS, Dijkstra's, and MST (Prim's method).

(a) [5 points] Which method would be the most suitable for recommending the fastest way to get from some starting location to a specific destination (e.g. from Ithaca to Miami South Beach)? Why?

Dijkstra's algorithm fits this problem well: the algorithm is a form of BFS that finds the shortest path to each node and, if we simply write down those paths, will find the shortest path from the starting node to any given destination.

(b) [5 points] What is the best way to find an alternative route if you are following the fastest route (as computed using the method you recommended in (a)), but discover that the recommended route is blocked because of road construction? E.g. you are driving from Ithaca to Miami South Beach, but the recommended route that runs from Philadelphia to Washington DC is incredibly slow. Don't write code, but do explain exactly how this problem can be efficiently solved (i.e. what algorithm to use, how to modify the graph, if you think it needs to be modified, etc). ***Your solution should stick to the original route except on this section, avoid all the road segments in impassable section, and still seek to minimize the driving time.***

A good solution would be to rerun Dijkstra's from the place where the detour starts, but to exclude edges that are blocked by the impass (just set the edge weights to infinity or delete them). Since we want to get back to the original path, we can select the first node beyond the blocked part of the thruway as our target. As soon as Dijkstra's "settles" the distance and path to that target node, we've found a location on our original route (on the other side of the blockage, of course) so we can stop running Dijkstra's and use the path it found as our detour. If the edge weights in the graph are correct now that we corrected the edges for the blocked part of the road, this will be the shortest way around the problem.

(c) [5 points] Suppose that two friends are coming with you. One lives in Syracuse (northeast of Ithaca), and one lives in New York City (southeast of Ithaca and Syracuse). Can an unmodified, standard implementation of Dijkstra's algorithm be used to compute a shortest path that starts in Ithaca but that passes through Syracuse and New York, and then ends in Miami? Explain.

Dijkstra's gives the shortest path and associated distances from a starting node to each other reachable node in a graph, but the shortest path from Ithaca to Miami might not include either Syracuse or New York. However, we can run the algorithm multiple times (without changing the algorithm).

So compute the shortest routes [Ithaca→Syracuse]+[Syracuse→New York]+[New York→Miami], and then do it again, for [Ithaca→New York]+[New York→Syracuse]+[Syracuse→Miami]. Add them as shown. Whichever of these two options is best will be the optimal route solving our problem.

(d) [5 points] Suppose we label some of the location nodes with restaurants. If you are given a route from Ithaca to Miami, a departure time, and a time when you would like to stop for food, how could you write a program that would recommend 10 options for stopping, ordered by their distance from where you will be at that stopping time? Don't give code, but make sure to tell us precisely how a route will be represented and precisely what your solution would do. Assume there are thousands of candidate restaurants on the route you'll be following.

A route is a list of nodes. We could scan the list, calculating the expected time when each node would be reached, until we find the node (or perhaps two nodes) closest to the target stopping time. Then we can search starting at that part of the list, looking forward and backward until we have the 10 closest options. These will all be restaurants ***right on the route***. That was what the question actually intended.

Some people assumed we should also include other nearby restaurants that might be *off* the chosen route. To do that, we could use BFS to enumerate the 10 closest options. The only thing is that if you do include a restaurant that is off the route, you need to double the associated "cost" because once you've eaten, you'll need to drive back to the original route again. Thus a Crate And Barrel right on the route but 5 miles from where we ideally wanted to stop has no "extra" cost, but one 5 miles off the route has an effective cost of 10 miles: we would need to drive to it, then drive back from it. We should factor that in. That makes this a slightly more complex algorithm to describe: for this version, you need to say that the distance-cost of selecting a give restaurant is its distance along the chosen route plus twice any driving that must be done along roads that are "off" the chosen route.

5. (20 points) We are building a multi-threaded program that maps the locations of animals in a new kind of "open" zoo where many of animals co-exist in a huge shared exhibit space, within which they can wander about as they wish.

Each animal wears a small tracking device. For each animal, the zoo map program creates a thread, which loops: every 30 seconds it reads the animal's location, then updates the map by deleting a pin that represents the previous location and placing a new pin representing the new location. An additional thread reads the map once every few seconds and redisplays it on monitors throughout the park. Visitors use the map to find the herd of zebras, the monkeys, etc.

(a) [5 points] You've been hired as consultant over the winter break to use your cs2110 skills to help fix a bug: animals keep flickering on the displayed maps. For example, there are moments when all the zebras simply vanish. Looking at the display thread and the location update thread you see code like the following:

```
/** Called every 5 seconds to redraw the map */
public static void redisplayMap() {
   synchronized (theMap)  {
         repaintMapGUI(theMap);
         refreshGUI();
   }
}
```

```
/** Called every 30 seconds to update location pin */
public static void updateLoc(Animal what) {
    synchronized (what) {
          oldLocation= what.where;
           what.where= what.RFID.readLocation();
   }
   synchronized (theMap)
           erasePin(oldLocation);
   synchronized(theMap)
            paintPin(what.animalSpecies, what.where);
}
```

Briefly explain why the animals sometimes vanish.

If the redisplayMap thread is running concurrently with the updateLoc threads, it could obtain the synchronization lock on theMap between the erasePin operation and the paintPin operation for some animals. Those will seem to have disappeared.

b. [5 points] Rewrite the code for updateLoc to correct the problem.

```
/** Called every 30 seconds to update location pin */
public static void updateLoc(Animal what) {
    synchronized (what) {
            oldLocation = what.where;
            what.where = what.RFID.readLocation();
    }
   synchronized (theMap) {
              erasePin(oldLocation);
              paintPin(what.animalSpecies, what.where);
   }
}
```

c. [10 points] A year has passed and you've graduated from Cornell with a PhD in Romance Languages, but there aren't any jobs for people with PhDs in English, so you are back at the zoo. The zoo's software director (who also has a PhD, in Medieval History) explains that there were simply too many threads, so he modified the program so that each location thread could update locations for hundreds of animals. Now he only has four location update threads.

He also added new functionality: now there are threads that track the animal feeding schedules (by keeping the carnivores well fed, they can share the exhibit with the grass-eating animals), threads that track medication schedules, etc. Now the program suffers from deadlocks! Reviewing the code you see a great number of synchronized code blocks, often nested one inside another. The director explains that while the program often use synchronized it never uses wait/notify, Locks or Semaphores.

i. [2 points] Define the term *deadlock*. *Hint: You do not need to list the three conditions for deadlock. We are only asking for a definition.*

Required part of the answer (full credit if you say this): A deadlock is a situation in which two or more threads are forced to wait for one another. Optional extra details (no extra credit): Even in a program with no wait operations and no Lock or Semaphore objects, if one thread is inside a synchronized block, a second thread will need to wait before it can synchronize on the same object. This form of wait can give rise to a deadlock if two or more threads form a cyclic wait.

ii. [2 points] If the Map refreshing thread became deadlocked, what would happen?

The map refreshing activity would cease, hence the map displayed in the park would freeze up, with no additional updates even if the animals continue to move around.

iii. [3 points] The director has heard of deadlocks and knows his program has deadlocks, but has difficulty visualizing the basic idea. When he learned Java, his instructor taught him that a deadlock was impossible in a program with no wait statements. Give a very simple example of how a deadlock could arise in Java, using two threads T1 and T2, two objects, x and y, and synchronization statements (but no wait or notify statements).

Discussion of the example was <u>not requested</u> and is <u>not required</u> (you will get full credit just for filling in the example). We gave partial credit for people who explained in English but failed to give a good example. As seen below, the essential thing is that T1 needs to be holding a lock on one object, perhaps x, and waiting to lock the other, perhaps y. And T2 simply needs to do the opposite.

Note: Quite a few people mistakenly assumed that a "synchronize" locks the object and blocks access to it. This is not so. The synchronize statement itself can block, but the object per-se remains accessible *even to a thread that doesn't call synchronize.*

What thread T1 does	What thread T2 does

synchronized(x) {
    synchronized(y) {
        anything
    }
}
	synchronized(y) {
	    synchronized(x) {
	        anything
	    }
	}

iv. [3 points] Give an example of a rule the zoo program could follow that would guarantee that no deadlocks can arise. Assume that the code still will need to use synchronized keywords and that the synchronization blocks still will need to nest but that no use of wait/notify occurs, and that there are no Lock or Semaphore objects.

Deadlock can be avoided by requiring that nested locking occur in some standard, fixed order, such as alphabetical order on the objects. If we had used that rule, T2 wouldn't be allowed to synchronize on x inside a block that synchronizes on y.

There are other ways to solve this. For example, all the threads could synchronize against some single shared object. This would mean that throughout the entire system there is just one thread in any critical section, anywhere, at one time, but it would certainly be safe and would work. It might just be slow.

As a note, our graders deducted points for silly solutions, like "just don't use threads" (the zoo probably had no choice at all), or "just don't get locks in a nested way" (that doesn't tell us what to do if there are two or more kinds of critical resources and sometimes, the application needs to update more than one of them).