NAME: ______________________________________ NETID: ________________________________

CS2110 Spring 2013, Prelim 1
March 7, 2013

The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. Good luck!

Write your name and Cornell netid on top! There are 8 questions on 11 numbered pages, front and back. Check now that you have all the pages You can separate the pages of this exam if it helps, but please restaple it using the stapler at the front of the room before handing the exam in.

We have scrap paper available, so you if you are the kind of programmer who does a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam, just so that we can make sense of what you handed in!

Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the space we provided, so if something seems to need more space, you might want to skip that question, then come back to your answer and see if you really have it right.

In some places, we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to make the problem more difficult. This does not mean that it's good style.

We've included one five-point extra credit question, so your total score can reach 103 points. However, P1 contributes Math.Min(P1-score, 100) towards your final grade.

	1	2	3	4	5	6	7	8	XC	Total
Max	16	20	10	10	12	12	10	10	3	103
Score										
Grader										

Extra credit question: For 3 points. Suppose that **X** and **Y** are of type `int` and contain values in the range 0..1023. Write code to swap X and Y without using additional memory (e.g. no extra variables). *Hint: We know a few ways to do this… any correct solution will get full credit.*

X = X+Y; Y = X-Y; X = X-Y;

X = X-Y ; Y = X+Y ; X = Y-X ;

X = X*Y ; Y = X/Y ; X = X/Y ;

X = X^Y; Y = X^Y; X = X^Y;

X = X<<32+Y ; Y = X >> 32 ; X = X &0xFFFF ;

1. (16 points) This first question tests your ability to read Java code provided by someone else and to understand what it will do. Tell us what the following two Java programs will print
Note: `Integer.parseInt` (s), for a string s, converts s to an integer.

(Part a: 8 points)

Run "ShapeShifter" with the string "5" as an argument.

```
public class ShapeShifter {

       public static void main(String[] args) {
              int n = Integer.parseInt(args[0]) ;
              int k = 0;
              int x = 1;
              while(x <= n) {
                     x = x + x;
                     k= k + 1;
              }
              System.out.println(transmogrify(k, n));
       }

       public static String transmogrify(int a, int b) {
              a= a-1;
              if(a == 0)
                  return b%2 + " ";
              return transmogrify(a, b/2) + b%2 + " "; // % is the remainder operation
       }
}
```

(4 pts) Output of ShapeShifter.main(new String[] { "5"}):

1 0 1

(4 pts) Now in no more than 25 words tell us what ShapeShifter is computing. (We expect a SHORT answer: Our sample solution is a 6-word sentence.)

It prints the argument in binary.

(Part b: 8 points)

Run “Fragment” with the string “63” as the argument. Note: If `Flist` is an `ArrayList<String>` containing X, Y and Z, then a call to `System.out.println(Flist)` prints “[X Y Z]”

```
public class public class Fragment {
      static ArrayList<Integer> Flist = new ArrayList<Integer>();

      public static void main(String[] args) {
            subDivide(Integer.parseInt(args[0]));
            System.out.println(Flist);
      }

      public static void subDivide (int n) {
            for(int f = 2; f <= n; f++)
                  if(n%f == 0) {
                        Flist.add(f);
                        subDivide(n/f);
                        return;  // stop searching and return to caller
                  }
      }
}
```

(4 pts) Output of Fragment.main(new String[] { “63”}):

[3 3 7]

(4 pts) Now in no more than 25 words tell us what Fragment is computing. (We expect a SHORT answer: Our sample solution is a 4-word sentence.)

It factors the argument.

2. (20 points) True or false? Circle the "T" for true. Circle the "F" for false.

a	T		If **X** is an object with some integer field f initially set to 3, and we execute `Y = X;` and then `Y.f = Y.f + 1;` the value of `X.f` will be 4.
b	T		If A is a superclass of B and variable myObj is declared to be of type A and we execute myObj = new B();, then the dynamic type of myObj is B but the static type is A.
c	T		If you extend an abstract class to create a concrete class, at a minimum you need to implement any abstract methods defined in the abstract class.
d		F	Every recursive method always needs to have at least one integer parameter that gets smaller each time it is called recursively, and a base case that checks for the values 1 and 0.
e	T		If we have a class **`Toy`** for all the toys in a store, and a subclass **`MyLittlePony`** for toys that are My Little Ponies, class **`MyLittlePony`** can define additional fields and methods not defined in **`Toy`**.
f	T		If **`Dog`** is a subclass of **`Animal`**, then ***public*** methods defined in class **`Animal`** will be available for use in an object of type **`Dog`**, unless they are overridden in **`Dog`**.
g		F	A Java class that implements an interface is not permitted to define methods or fields other than the ones specified by the interface
h	T		A Java class can extend at most one class but can implement many interfaces.
i	T		A static method cannot access instance fields or methods without specifying an object instance and using a qualified reference (e.g. x.something or x.m())
j		F	An instance method is not permitted to access static fields or methods even if they are defined in the same class.
k		F	If an expression is successfully type-checked by the Java compiler, it won't throw runtime exceptions.
l		F	If a method is declared to be private, it can be accessed from outside the class if you first cast the class to public, as in "`x = (public)foo.somePrivateMethod();`"
m	T		If method m is declared: **`static void m(ArrayList<Object> arg),`** you will get a compile-time error if you declare **`X`** to have type **`ArrayList<String>()`** and try to invoke **`m(X);`**
n		F	If a variable **`myPet`** of type **`Pet`** is currently *null*, then **`myPet.Bark()`** will do nothing. (Here, assume that method **`Bark`** is defined by class **`Pet`**.)
o		F	In Java, if **`x`** is of type String, the statement **`if (x ==`** *null*) **`S;`** will never execute **S** because the condition would always be false.
p		F	Suppose Dog is a subclass of Animal and we execute: **`Dog m = new Dog("Midnight"); Animal a = (Animal)m; Dog d = (Dog)a;`** The last line of the sequence will fail, because once we turn **m** into an Animal in the second line, we can't change our minds and reverse the transformation this way.
q		F	Searching for an element in a general tree containing N items would always require far fewer than N "compare" operations.
r	T		If B is a subclass of A, and an object of type B has been created but you are using a variable **`x`** with declared (static) type A to point to it, you would need to explicitly downcast the variable by writing **`((B) x).m()`** to access a public method m available only in class **`B.`**
s		F	If **`B`** is a subclass of **`A,`** **`y`** is an object of type **`A,`** and **`x`** is a variable declared to be of type **`B,`** then we could assign **`x = (B)y;`** to make **`X`** point to object **`y.`**
t	T		If container implements interface **`List<T>,`** then the Java statement "**`for (T elem: container) S;`**" will execute statement **S** once for each object in **`container`**, with **`elem`** pointing to that object.

3. (10 points) Your new boss at Purgatory.com has decided to print a novel about life in Purgatory but wants to save money by not hiring an author (Lucifer gave her the job on condition that she would cut the budget). Her idea is to generate the entire book from a grammar. *“”* denotes an empty string. The grammar lacks punctuation characters (periods, commas, etc), so don’t worry about punctuation.

> **Sentence** = **Noun Modifier Verb ObjectPhrase** | **Noun Verb**
> | **Sentence Linkage Sentence**
> **ObjectPhrase = Object Noun**
> **Linkage** = after | before | but | while | and
> **Noun =** Tina | Harry | Sally | Jim
> **Modifier =** joyfully | painfully | slowly | “”
> **Verb =** stood | sat | fell | jumped |walked
> **Object =** on | over | under

You feed this into a program you found on the web that takes a grammar as its input and prints all the 1-word sentences (if any), then all the 2-word sentences, then all the 3-word sentences, etc. The program stops if it has printed every possible sentence. It prints in a normal 12-point font.

(a) (6 points) Your boss is very excited about the program you found and has decided that you should run the program until it has generated a novel a million pages long, but then cut it off “mid-sentence” because, as she explains, it will leave the reader fascinated and wondering what happens next. She plans to make use this book in the “welcome to Purgatory” class that runs every fall before the start of the new semester. Is it possible to use this grammar to generate a million pages of text (without running the program more than once)? Why or why not? *Hint: Focus your answer on the rules in the grammar!*

Yes, it can. The very first rule includes a recursive case: *Sentence Linkage Sentence.* There are also simple ways to create sentences that would consume space in the text, and many linkage options). The book will be exceptionally dull, but one can generate an unbounded number of pages of text by simply creating some short sentence, picking a linkage, and then repeating the process.

(b) (4 points) For each of the following, say whether it is a sentence generated by the grammar. Ignore capitalization and whitespace.

(i)	**Y**	N	Tina sat on Harry after Sally fell and Jim joyfully walked under Harry
(ii)	**Y**		Sally sat
(iii)		**N**	Joyfully Harry walked painfully
(iv)		**N**	Tina and Harry painfully sat on Sally after Jim

4. (10 points) Choose the (single) best alternative.

(a) Suppose that classes B and C both extend class A.

A. It is not possible to cast an object of type B to type A.
B. If you use an object of type C in a situation where Java expects an object of type A, Java will "autobox" C into an object of type A.
C. Java would not let you cast an object of type C to type B, or vice versa.

(b) The `static` keyword:

A. Tells Java that something will never change
B. Says that there is one copy of a field or method and that it is associated with the class
C. Forces Java to interpret the associated field or method as if it were defined in the parent class.

(c) Generics:

A. Can be used with primitive types (e.g. ArrayList<int>), not just object types.
B. Allow the type checking features of Java to understand what the programmer is doing and to prevent many kinds of errors by catching them as compile-time errors.

(d) `String s = "The value of x is " + x; // You may assume x is an object`

A. Needs an explicit cast `(String)` unless variable x is of type `String`.
B. Is legal but would use an inherited toString() method for x unless you define one of your own.
C. Will not compile because Java can't determine whether or not x is **null**.

(e) A class might declare multiple constructors:

A. Because there may be more than one way to initialize it
B. To ensure that the superclass is initialized first
C. As a way to override the default handling of operations like .equals and .hashcode

(f) An object of type `HashSet<String>` could be used to:

A. Check rapidly to see if a particular string has been seen previously
B. Alphabetize a set of strings
C. Automatically assign an integer value to each string in a set so that you can easily convert from a string to its corresponding integer or from the integer back to the corresponding string.

(g) An overloaded method:

A. Has multiple definitions, each with a different type signature
B. Is needed when a subclass wants to define a method differently than its parent did.
C. Must not use the same name as any existing method (must define a completely new name)

(h) A problem with JUnit testing is that:

A. The assertions are very slow and can be too costly for use in "production" code
B. When you change your code you may have to change the JUnit tests too
C. JUnit tests are not executable

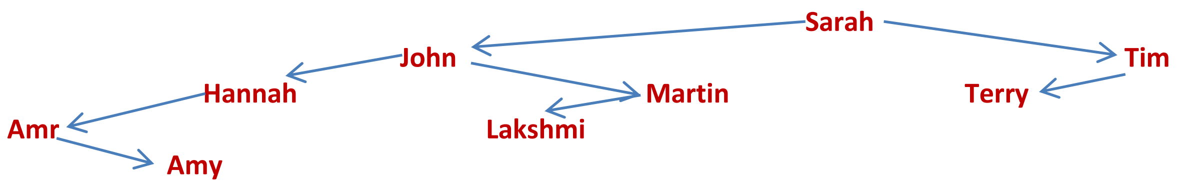
(i) If a class extends an abstract class:

A. It cannot add new methods or fields not already defined by the abstract class
B. It cannot implement any interfaces that the abstract class didn't implement
C. We would describe it as a subclass of the abstract class.

(j) Suppose class `A` defines method `m`, `B` is a subclass of `A` that overrides `m`, and `X` is declared to have type `A` (for example perhaps we executed `A X = new B()`).

A. X.m() will invoke the version of m defined in A even if B overrides that definition and provides a different definition.
B. X.m() will invoke the version of m defined in B, and the version defined by A, if any, will not be invoked at all unless the version in B calls super.m() explicitly.
C. You would need to call ((B)X).m() to be sure you are calling the version defined by B.

5. (12 points) (a) Draw the binary search tree obtained by inserting these strings in the following order: *“Sarah“,“John“,“Hannah“,“Martin“,“Amr“,“Lakshmi“,“Tim“,“Terry“,“Amy“.* Keep in mind that a BST is sorted. Assume that the normal lexicographic (dictionary) comparison ordering is employed.



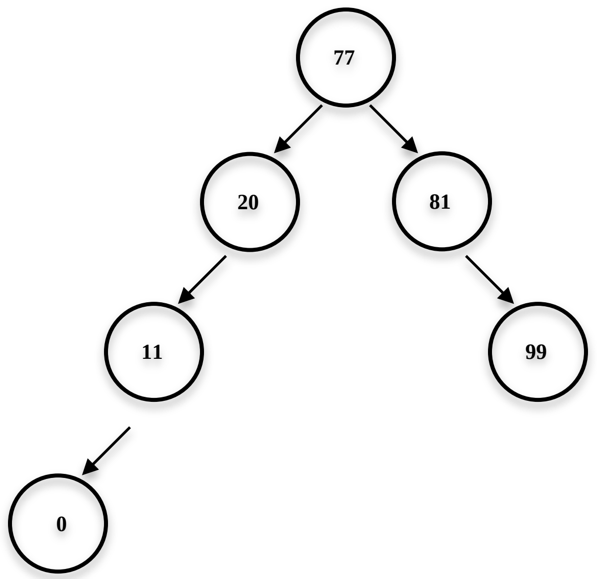
(b) Binary search trees allow us to perform lookups very quickly. To demonstrate that, write down the elements encountered (in order) while performing a lookup for “*Lakshmi*” on the tree from (a).

Sarah… John… Martin… Lakshmi

(c) To drive home the benefits of a BST, assume that instead of using a binary search tree the same elements were stored in an **`ArrayList<String>,`** by calling **`.add()`** in the order shown above. Write down the elements encountered (in order) while looking up "Lakshmi" in the list created in this way.

Sarah… John… Hannah… Martin… Amr… Lakshmi

6. (12 points) More on binary Search Trees (BSTs). Consider this tree:



a. (6 point) Is this a valid binary search tree? Explain briefly why, or why not.

Yes. The requirement is that for each node,

The node has 2 child-node fields, which may be null (so has $\leq$ 2 children)

(this.left==null || this.left.value < this.value) && (this.right == null || this.right.value > this.value)

In other words, anything to the left must have a value less than the node's value, and anything to the right must have a value larger than the node's value

b. (6 points) Now, assume that a **`treeNode`** has fields **`datum, left`** and **`right`** (exactly as seen in class). Write a method **`childBalance`** that takes a node and returns 0 if the number of its children on the left is the same as the number on the right, -*k* if there are *k* more children on the left than on the right, and *k* if there are *k* more children on the right than on the left.

Hints: Feel free to write helper functions. It often helps to write the code first on scrap paper, then copy the version you want us to grade (with comments) once you have it right. Remember that children includes all children, not just direct descendants. For the example in our picture, **`childBalance`** *would return -1 if called on the root, and would return 1 if called on the node containing value 81. Since node 99 has no children on either the left or the right, if called for node 99* **`childBalance`** *would return 0.*

```
// Note: We've written these as static methods, but instance methods are fine too
// After using helper method countNodes to determine how many nodes are on each side,
// childBalance returns the right count minus the left count
public static int childBalance(treeNode nd)
{
    return countNodes(nd.right) - countNodes(nd.left);
}

// Helper method countNodes recursively counts nodes in a subtree, including this one
// We use an in-order traversal and just add 1 for each non-null node visited
public static int countNodes(treeNode nd)
{
    if(nd == null) return 0;
    return countNodes(nd.left) + 1 + countNodes(nd.right);
}
```

7. (10 points) This question tests your knowledge of tree traversals as defined in the lecture on trees.

a. (6 points, 3 each)

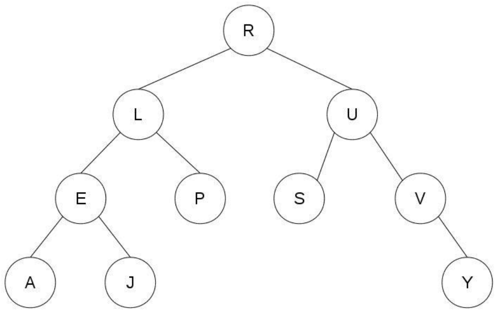
Write down the In-Order and Post-Order traversals of the binary tree in the given figure.

In-Order:

A-E-J-L-P-R-S-U-V-Y

Post-order:

A-J-E-P-L-S-Y-V-U-R



(b) (4 points) Draw a binary tree with its nodes correctly labeled such that it has the following two traversals:

Pre-Order: A G T C H W U Z Y

In-Order: G A H C W T U Z Y

$$\begin{tikzcd}
 & \mathbf{A} \arrow[dl] \arrow[drrrr] & & & & & & & \\
\mathbf{G} & & & & & \mathbf{T} \arrow[dll] \arrow[dr] & & & \\
 & & & \mathbf{C} \arrow[dl] \arrow[dr] & & & \mathbf{U} \arrow[dr] & & \\
 & & \mathbf{H} & & \mathbf{W} & & & \mathbf{Z} \arrow[dr] & \\
 & & & & & & & & \mathbf{Y}
\end{tikzcd}$$

8. (10 points) Write a method `static ArrayList<Integer> cleanup(Integer[] v)` that returns an ArrayList containing the elements of v in order, with the conditions stated below. If you are more familiar with the Java Vector type, you can return a Vector<Integer> instead.

Here are the conditions: (1) any zeros at the start or end of v are removed, (2) any series of zeros inside v are replaced with a single zero, and (3) non-zero elements are replaced with their absolute values. Your code may assume that v contains at least one non-zero element.

For example, if v is initially {0, 0, -3, 6, 5, 0, 0, 18, 11, -99, 0}, cleanup returns {3, 6, 5, 0, 18, 11, 99}.

Hint: there are many ways to solve this; three are shown below and these aren't the only ways. We did not require some specific solution: Any solution that will return the correct answer is eligible for full credit, if coded in a clear way. But comments do help! As above: helper functions are fine if you find them useful. People who solved this first on scrap paper, then made a clean copy to hand in, tended to get higher scores because it was easier to understand their code.

```
// cleanup carries out the desired function using two boolean flags: "saw zero" (meaning
// the method just saw a zero) and "inside" meaning has seen at least one non-zero value
// The rule can then be expressed this way: look at the next number, and if it is zero, set
// saw-a-zero.  When we find a non-zero number : If we are inside, and saw-a-zero, output a zero
// Then outputl abs(n) to convert negative numbers to their absolute values. Since inside
// will initially be false, leading zeros will be ignored, and if the list ends with zeros,
// nothing will trigger an "add", so those would be ignored too.
static ArrayList<Integer> cleanup(Integer[] v)
{
    ArrayList<Integer> theList = new ArrayList<Integer>();
    boolean sawZero = false, inside = false;
    for(Integer n: v) {
       if(n == 0)
          sawZero = true;
       else // n is non-zero
       {
           if(inside && sawZero)
              theList.add(0);      // Turns inner sequences of 0 into a single 0
          theList.add(abs(n));  // Output the absolute value to deal with negative ints
          sawZero = false;       // We just saw a non-zero number
          inside = true;          // Remember that we're "inside" the sequence now
        }
     return theList;
}
```

```
--- version two:
static ArrayList<Integer> cleanup(Integer[] v)
{
    ArrayList<Integer> theList = new ArrayList<Integer>();
    for(Integer val: v)
       theList.add(abs(val));
    // Trim leading zeros
    while(theList.size() > 0 && theList.elementAt(0) == 0)
       theList.remove(0);
    // Trim trailing zeros
    while(theList.size() > 0 && theList.elementAt(theList.size()-1) == 0)
       theList.remove(theList.size()-1);
    // Notice that the first and last elements are now non-zero.  This code exploits that
    //  It scans for 0 0 sequences and removes the first zero if it sees one
    for(int n = 1; n < theList.size()-2; n++)
       if(theList.elementAt(n) == 0 && theList.elementAt(n+1) == 0)
       {
          theList.remove(n);
          // Our removal shifted things to the left…
          n--;
        }
    return theList;
}

--- version three:
static ArrayList<Integer> cleanup(Integer[] v)
{
    ArrayList<Integer> theList = new ArrayList<Integer>(v);
    // Find first non-zero element
    int start = 0;
    while(start < v.Length-1 && v[start] == 0)
        v = v+1;
    // Find last non-zero element
    int end = v.Length-1;
    while(end < v.Length-1 && v[end] == 0)
        v = v-1;
    // Now copy elements
    for(int n = start; n < end; n++)
    {
         theList.add(abs(v[n]));
         // If we just copied a 0, skip additional zeros.  This terminates because we know
         // the region from start..end within v terminates with a non-zero value
         while(v[n] == 0 && v[n+1] == 0)
            n = n+1;
    }
    return theList;
}
```