Last Name: First Name:

Grades for the final will be posted on the CMS as soon as it is graded, probably some time late tomorrow. Grades for the course will be posted much later. Please do not email us asking to see your final this December. You can look at your final when you return in January. HAVE A GOOD BREAK!

You have 2.5 hours to complete the questions on this exam, which are numbered 0..8. Please glance through the whole exam before starting. The exam is worth 100 points.

Question 0 (2 points). Print your name and **net id** at the top of each page. Please make them legible. Hint: do this now.

Question 0. _____	(out of 02)
Question 1. _____	(out of 13)
Question 2. _____	(out of 10)
Question 3. _____	(out of 15)
Question 4. _____	(out of 10)
Question 5. _____	(out of 15)
Question 6. _____	(out of 15)
Question 7. _____	(out of 10)
Question 8. _____	(out of 10)
Total _____	(out of 100)

Question 1 (13 points) Executing statements, evaluating new-expressions, types. Consider the sequence of statements given below, which make use of classes `Alpha` and `Beta` declared at the bottom of the page.

First, draw all the declared variables (on the left below; don't put any values in yet).

Second, execute the sequence of statements, in order, *except* that for any statements that are syntactically illegal, do not execute such statements, but rather just cross them out and then skip them.

You need not draw frames for method calls. But you *must* draw all objects that are created by evaluating new-expressions. Include the parameter types for method parameters, but omit the parameters themselves. Do not erase things that have to be erased; instead, cross them out.

```
Alpha a;    a= new Alpha(2);
Beta b;     b= new Beta(3);
Alpha c;    c= b;
            c.f= 10;
int y;      y= a.m(b);
int z;      z= b.lead();
            z= c.lead();
```

```
public class Alpha {
   public int f;

   public Alpha(int n) {
     f= 2*n;
   }

   public int deal() {
     return f / 2;
   }

   public int m(Alpha a) {
     return a.deal();
   }
}
```

```
public class Beta extends Alpha {

   public Beta(int n) {
     super(2*n);
   }

   public int deal() {
     return f + 1;
   }

   public int lead() {
     return super.deal();
   }
}
```

Last Name: First Name:

Question 2 (10 points). Algorithms. Write the function to implement the Dutch National Flag algorithm, as explained in class. *Note that it processes* `b[p..q]`. This algorithm was discussed in terms of red, white, and blue objects. Here, consider red to mean a negative integer (< 0), white to mean 0, and blue to mean a positive integer (> 0).

You must do the following. (1) Write the precondition, as a diagram, in the place shown below; (2) Write the postcondition, as a diagram, under the precondition; (3) Write the invariant, as a diagram, in the place provided. (4) Write the loop, with its initialization before the invariant and the loop after the invariant. Of course, your code must be consistent with the invariant.

You may write "swap <something> with <something>" instead of giving explicit Java code for the swap.

```
/**

  precondition:


  postcondition:



*/
public static void DNF(int[] b, int p, int q) {


   /** invariant:

   */
  while (                                    ) {









  }
}
```

Last Name: First Name:

Question 3 (15 points). Recursion. You know about the hierarchy of directories and files on your hard drive. A partial list of the hierarchy on our hard drives for the CS1110 website appears to the right, with indentation to denote containment. For example, directory `assignments` contains file `assignments.html` and directory `a1`. Note how we put `'/'` after a directory name to emphasize that it is a directory name, although `'/' is` *not* part of the name.

```
CS1110/
  index.html
  assignments/
    assignments.html
    a1/
      a1.pdf
      a1.html
      temp/
        a1-scratch.html
  announcements.html
```

Each instance of the class declared on the next page represents either a file (if field `items` is `null`) or a directory (if field `items` is not `null`). We have omitted all methods, like constructors, that you do not need to know about to answer this question.

On the next page in the space provided, write the bodies of functions `path` and `directory`. **Important:** The only loop you may write is one to process the items in Vector `items`.

You may need these methods: For v a Vector:
• v.size() is the number of items in v.

• v.get(i) evaluates to the element at position i in the vector v.

We give examples of what each function should produce.

(1) Consider a call `v.path()`, where `v` contains the name of the instance that represents item `a1` in the table to the right above. The call should produce the string

"CS1110/assignments/a1/ "

Function path() should, as part of its body, call the same function in the containing directory, field `in` .

(2) Consider the call `v.directory()`, where, again, `v` contains the name of the instance that represents item `a1`. This call should produce the string

"\nCS1110/assignments/a1/\nCS1110/assignments/a1/a1.pdf\nCS1110/assignments/a1/a1.html\n
CS1110/assignments/a1/temp/\nCS1110/assignments/a1/temp/a1-scratch.html "

which, when printed, will look like this (because of the new-line characters '\n'):

CS1110/assignments/a1/
CS1110/assignments/a1/a1.pdf
CS1110/assignments/a1/a1.html
CS1110/assignments/a1/temp/
CS1110/assignments/a1/temp/a1-scratch.html

Last Name: First Name:

```
import java.util.*;

/** An instance represents a directory or file on a hard drive .*/
public class HDItem {

   private String name; // The name of this directory or file

   /** If items is not null, this instance is a directory, and its contents are the HDItems in vector items.
       If items is null, this instance is a file. */
   private Vector<HDItem> items;

   /** The directory that contains this instance (null if this is the top-level directory) */
   private HDItem in;

   /** = the path for this item, with "/" at the end if this item is a directory */
   public String path() {
      // Notes: Do not use loops.
      // This instance gives you the last name in the path. The preceding names, if any, come from field in.









   }


      /** = a String that gives the paths of all elements in this directory or file (including those in
          subdirectories), with  each path being preceded by "\n" so they are on separate lines.    */
   public String directory() {
      // Notes: Use a loop only to process the elements of Vector items (if they need processing).
      // The path for this instance is the first one to produce.
      // All others come from Vector items, if items is not null.









   }
}
```

Last Name: First Name:

Question 4 (10 points) Subclasses and Breakout. In Breakout (A7), each brick was an instance of `GRect`, and the bricks were not saved in an array or vector. We simply did something like this:

(1) // add a new brick of width w and height h to the canvas
```
add(new GRect(w,h));
```

With this implementation, the only way to tell whether an object is a brick is to determine that it is *not* the paddle, *not* a label that is on the canvas, and *not* any other object that is on the canvas, like this:

(2) // If g is a brick, remove it from the game
```
if (g!=paddle && g!=label1 && g!=label2 && ... ) {
    remove g;
}
```

This is tedious and error prone. For example, if we add another feature to the game that places more objects on the canvas, will we remember to change this statement?

To solve this problem, we could declare a subclass of `GRect` and use it to determine directly whether `g` is a brick. Below, write the subclass. It should have one constructor so that the new-expression in (1) can be suitably changed (don't assume a fixed brick width or height, and don't worry about the color, fillcolor, or whether it is filled). Include proper specifications.

After the declaration of the class, rewrite statements (1) and (2) to make use of this new subclass.

Last Name: First Name:

Question 5 (15 points) Loops. Consider determining the level of nesting of parentheses in a string. Examples are given to the right. This can be done with a loop that processes the characters of the string from left to right, keeping track of the number of '(' that have been found for which no ')' has yet been seen and keeping track of the nesting of '('. **Hint**: think of how the level of nesting so far is related to the number of unmatched '(' so far --- can the latter ever be bigger than the former? The latter *can* be smaller.

String	Level of nesting
""	0
"a+2"	0
"a+(2)"	1
"(a)+(2)"	1
"((a+2) * (3))"	2
"(((a+2)))"	3

We also want to know if the parentheses are balanced. For example, they are not balanced in the following strings:

"a)" "a(" ")(" "(a+3()" "(()))" ")()"

Complete the method that is given below, using a for-loop that processes a range of integers. The post-condition for the loop is given. You must write a loop invariant, and the code you write must use (be consistent with) that loop invariant.

```
/** = the level of nesting of parentheses in s.
 Throw an IllegalArgumentException with message "unbalanced parens" if parentheses are unbalanced. */
public static int level(String s) {
   // Store in n the level of nesting of parentheses in s, but throw the exception if not balanced.
   int op=              ;
   int n=               ;
   // invariant:



   for (                                                    ) {









   }

   // postcondition: s[0..s.length()-1] is balanced except that it has op open '(' with no matching ')'.
   //                n = level of nesting of '(' in s[0..s.length()-1].
   if (op > 0) {// Don't forget to fill this in


   }
   return n;
}
```

Last Name: First Name:

Question 6 (15 points) Subclasses. Class `HDItem`, on page 4, is used to represent all items in a hard drive: the top directory, other directories, and files. Suppose we decide instead to use subclasses to represent items, with the following structure:

class `HDItem` (an abstract class)
  class `Directory` extends `HDItem` (instances are used for directories)
    class `TopDirectory` extends `Directory` (the one instance is for the top directory)
  class `File` extends `HDItem` (instances are used for files)

We give the outline of all four classes on the next page, showing only the methods you need for this question.

(a) Class `HDItem` on page 4 contains 3 fields: `name`, `items`, and `in`. For each of these fields, declare it in the single class on the next page that is most appropriate for that field (write the declarations on the righthand side of the page), and explain briefly why you placed it in that class. You do not have to give the meanings of the fields as comments.

(b) For field `items`, you either declared it as static or declared it as non-static. Explain the reason for your choice here.

(c) Complete the body of function `equals` in abstract class `HDItem` on the next page.

(d) We have declared function `path` in all three subclasses on the next page (with empty bodies). Below, explain whether or not you would declare `path` in abstract class `HDItem`. Give your reason for your choice. If you decide to put it in the abstract class, indicate whether the method should be abstract or not, and why.

(d) Complete the bodies of function `path` in classes `File`, `Directory`, and `TopDirectory` on the next page. TO SIMPLIFY YOUR CODE, ASSUME THAT THE FIELDS ARE PUBLIC. [You may find it interesting to compare your answers to those you gave in question 3; but don't write anything down about the comparison; we make this remark purely for your own edification.]

Last Name: First Name:

```
/** An instance represents a directory or file on a hard drive */
public abstract class HDItem {
   /** = "obj is an HDItem with the same name as this one"
         (we have simplified the specification in order to make this exam question simpler). */
   public boolean equals(Object obj) {









   }
}

/** An instance is a directory on the hard drive  (not the top directory) */
public class Directory extends HDItem {
   /** = the path for this item, with "/" at the end because it is a directory. */
   public String path() {




   }

/** An instance is the top directory on the hard drive */
public class TopDirectory extends Directory {
   /** = the path for this item, with "/" at the end because it is a directory. */
   public String path() {






   }

/** An instance is a file on the hard drive */
public class File extends HDItem {
   /** = the path for this item. */
   public String path() {




   }
}
```

Last Name: First Name:

Question 7 (10 points). Douglas Hofstadter, best known for his book *Gödel, Escher, Bach: an Eternal Golden Braid*, for which he was awarded the 1980 Pulitzer Prize for General Non-Fiction, defined the following two *mutually recursive* functions:

```
public static int female(int n) {
   int f;
   L1: if (n == 0) {
         L2: return 1;
   }
   L3: f= female(n-1);
   L4: return n - male(f);
}
```

```
public static int male(int x) {
   int m;
   L5: if (x == 0) {
         L6: return 0;
   }
   L7: m= male(x-1);
   L8: return x - female(m);
}
```

The sequences of values `female(0)`, `female(1)`, `female(2)`, ... and `male(0)`, `male(1)`, `male(2)`, ... have been shown to be related to the Fibonacci series 0, 1, 1, 2, 3, 5, 13....

Assume these methods are declared in a class `C`. Execute the statement `s= C.female(1)`, drawing frames for method calls. *Stop executing when you have drawn the third frame and assigned arguments to parameters in that frame.* If you are supposed to erase a frame or change a program counter, cross it out rather than erase it. Use labels `L1`, `L2`, ... as the program-counter values in your frames.

Last Name: First Name:

Question 8 (10 points). Short answers.

(a) Assume that `x` and `y` are declared as **`int`**s. Write a try statement that sets `x` to the value of `x/y`. Its catch clause should catch an `ArithmeticException`, and it should print the statement "whoops, you divided by 0" if the `ArithmeticException` is caught.

(b) What three things need to be done in order to get a `JFrame` to listen to a mouse click? Speak in general terms —e.g. if you have to write a method, you don't have to know its exact name.

(c) Below is a sequence of statements. First, draw the variables declared in it. Then, execute the statements, in order. Make sure you draw any objects (arrays) that are created during execution.

```
int[][] b= new int[2][];
b[0]= new int[]{1, 3, 5};
b[1]= new int[] {2, 4};
```