We grade the final Thursday. Grades for the final are expected be posted on the CMS in late afternoon. Grades for the course will take a few days more. You can look at your final when you return next year. HAVE A NICE HOLIDAY!

Please submit all requests for regrades for things other than the final BY NOON TOMORROW. Use the CMS where possible; email Gries otherwise.

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

Question 0.	_________	(out of 01)
Question 1.	_________	(out of 12)
Question 2.	_________	(out of 12)
Question 3.	_________	(out of 12)
Question 4.	_________	(out of 12)
Question 5.	_________	(out of 12)
Question 6.	_________	(out of 15)
Question 7.	_________	(out of 12)
Question 8.	_________	(out of 12)
Total	___________	(out of 100)

Question 0 (1 point). Print your name and **net id** at the top of each page. Please make them legible.

Question 1 (12 points). Loops and invariants. Write a loop (and its initialization) that places the odd elements of array `b` in the beginning of array `c`, in reverse order in which they appear in b. For example, if `b` contains {3, 1, 4, 5, 9, 8, 7}, the first five elements of `c` will be {7, 9, 5, 1, 3}.

The precondition, invariant, and postcondition are given below. You *must* use the given invariant. Please help yourself and us by looking very carefully at the invariant. Drawing the invariant as a picture-diagram may help. Of course, use the variables that are named in the invariant.

// precondition: The size of `b` is at least 0 and `c` is big enough to contain the odd elements of `b`.

// invariant: `c[0..k-1]` contains the odd elements of `b[h..b.length-1]`, in reverse order.

while ( ) {

}
// postcondition: `c[0..k-1]` contains the odd elements of `b`, in reverse order.

Question 2 (12 points). Exception handling and Vectors.

(a) On the back of the previous page, write a class definition for a class `NotLowerCaseLetterException` whose instances may be thrown.

(b) An instance of the class declared below maintains a list of Strings. Each String is supposed to contain only lowercase letters (in 'a'..'z'). Write, in this order, the bodies of procedure `check`, the constructor, procedure `append` (remember that **char** is a numerical type so you can compare characters) and procedure `appendMessage`. Note the comment in the body of procedure `appendMessage`, which explains how we want you to write the body.

```
/** An instance maintains a list of Strings that contain only characters in a..z. */
public class LettersStringList {
   // The list of Strings. Each String contains only characters in 'a'..'z'.
   private Vector<String> list;

   /** Throw a NotLowerCaseLetterException with a suitable detail message if s contains
         a character that is not in 'a'..'z'. */
   private static void check(String s) {









   }

   /** Constructor: a new instance with an empty list of Strings. */
   public LettersStringList() {


   }

   /** Append s to the list. If all the letters of s are not in 'a'..'z', throw a NotLowerCaseLetterException. */
   public void append(String s) {




   }

   /** If s contains only chars in 'a'..'z', append s to the list. Otherwise, print "Mistake in parameter." */
   public void appendMessage(String s) {
      // This body should NOT use an if-statement. Instead, use a try-catch-statement.









   }
}
```

Question 3 (12 points). Miscellaneous.

(a) Below is the skeleton of class C. To the right of it, state the reason for making a class abstract and indicate on the class itself how to make it abstract.

```
public class C {
        …
        …
        …
}
```

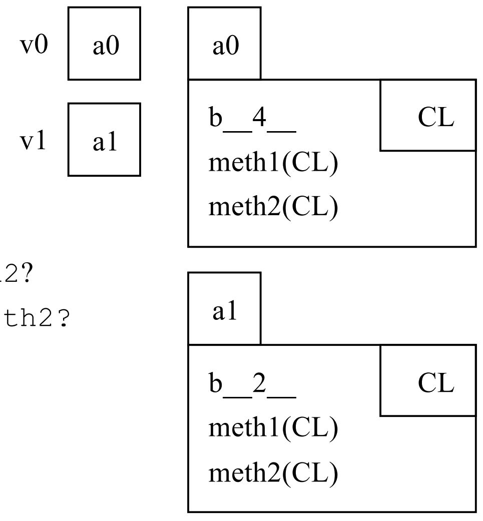
(b) Write down the steps in executing an assignment `x= e;` for some variable `x` and expression `e`.

(c) Consider the folders and variables shown to the right and that method meth1 is this:

```
public meth1(CL p) {
   meth2(this);
   meth2(p);
}
```

Suppose this call is executed: `v0.meth1(v1);`

1. Draw the frame for the call `v0.meth1(v1);`
2. What is the value of the argument of the first call to `meth2`?
3. What is the value of the argument of the second call to `meth2`?



Question 4 (12 points) Executing code. On this page are class definitions for classes `Item` and `Book`. On the next page is a class Store. Execute this call:

Store.session();

Write the output of println statements directly beneath the println statements, in the space provided (on the next page).

We suggest that you start by drawing all local variables of method Store.session. Then, as you execute the sequence, draw all objects created and execute any assignment statement faithfully. You probably won't get the correct answers if you don't do this.

```
public class Item {
  /* Total cost of all Items created */
  private static int totalCost=  0;

  private int cost; // Cost of this item (in dollars)

  /** Constructor: new Item with cost c.*/
  public Item( int c) {
     cost=  c;
     totalCost=  totalCost + c;
  }

  /** = Cost of this item */
  public int getCost()
     { return cost; }

  /** = the total cost of all Items */
  public static int getTotalCost()
     { return totalCost; }

  /**Replace cost of this item by c */
  public void replace(int c) {
     totalCost=  totalCost – cost + c;
     cost=  c;
  }

  /** Add d to this Item's cost*/
  public void add(int d) {
       cost=  cost + d;
       totalCost=  totalCost  + d;
  }
}
```

```
public class Book extends Item {
  /* total number of instances of Book created */
  private static int numBooks=  0;

  private String book; // title

  /** Constructor: a new book with title t
                        and cost c*/
  public Book(String t, int c) {
     super(c);
     book=  t;
     numBooks=  numBooks + 1;
  }

  /** = "<title>: <cost>" */
  public String toString(){
     return book + ": " + getCost();
  }

  /** = this book's title */
  public String getTitle()
     { return book;  }
}
```

```
public class Store {
  public static void session() {
      Item one= new Book("Power of Now", 24);
      Book forth= new Book("Truth Wins Out", 12);
      Item pricey= new Item(30);
      Book five= new Book("Honesty Over All", 10);
      Item treat= forth;
      Book two= (Book) one;
      treat.replace(10);
      one.add(4);
      System.out.println(two);


      System.out.println(pricey);


      System.out.println(treat);


      System.out.println("Cost of Item: " + Item.getTotalCost());


      System.out.println("Book is: " + forth.getCost());


      System.out.println("Book is: " + ((Book)one).getTitle());


      System.out.println("Are books the same?"  +  (two.getTitle() == five.getTitle()));


      System.out.println("Are books the same?"  +  (two.getTitle().equals(five.getTitle())));


  }
}
```

Question 5 (12 points). Recursion. We want to compress long strings that contain many adjacent equal characters (but no digits). For example, the compression of "aaaaaaaaaaaabbaaaaaazzzz" would be "a12b2a6z4". Thus, the compression of a string that contains no digits is a copy of the string but in which each sequence of adjacent equal characters (e.g. "$$$$$$$$$$") is replaced by that character followed by the number of times it occurs (e.g. "$10").

Write the following two functions. Use recursion. Do not use iteration (loops). Actually, the first function is far better written using iteration, but we want to see how well you do with recursion.

Remember our principle of using already written functions as much as possible, in order to reduce our work.

```
/** = if s is "", then 0;
      otherwise: the number of times the first character of s appears at the beginning of s.
      E.g. for s = "xxxy#xxxxz", the answer is 3. */
public static int numberOfFirst(String s) {







}

/** = the compression of s
        Precondition: s does not contain a digit in '0'.. '9'. */
public static String compress(String s) {







}
```

Question 6 (15 points). Classes. Read the complete question before writing anything.

The US Passport Office is developing a programming system to maintain information about passports. They want a Java class `Passport`, each instance of which will contain (1) the name of a person (a String), (2) a state designation (e.g. NY for New York, CA for California), and (3) the passport number that is assigned to the person.

We assume that all people have different names (we could us social security numbers to differentiate people, but for purposes of simplicity, we don't.)

The first passport number to be assigned is 1. The next one is 2, then 3, and so on. Class `Passport` must keep track of which numbers have been assigned. One way to implement this is to have a variable, declared appropriately, that contains the number of `Passport`s that have been created.

`Passport` should maintain a `Vector` of all instances of `Passport` that have been created.

There should be a function that returns an existing `Passport` for a person. (What should it return if there isn't one? You must specify this.)

Class `Passport` has a constructor, but the normal user should not be able to use it. Instead, if someone wants a new passport, they have to call a method `assign`, with their name and state as arguments. If a `Passport` already exists for the person, it is returned; otherwise, `assign` assigns them a `Passport` number, creates a `Passport` object of Passport, and returns (the name of) the object. Here is an example of a call on `assign` and an assignment of the value it yields.

```
Passport pass= Passport.assign("David Gries", "NY");
```

Write class `Passport`. It should contain declarations of all the variables and methods necessary to do what is described above. Variables and methods should be static or non-static, private or public, as required by good programming practices and to have the class work properly. Make sure you put a comment on each variable to describe its meaning and any constraints on it.

Do not write method bodies. Points will be deducted if you do. Instead, put good specifications as comments on the method headers and use {} for each method body. Write any getter methods (but not their bodies) that you feel are needed under good standard programming practices.

This page left intentionally nonblank. Use it for the answer to question 6 if you want.

Question 7 (12 points). Miscellaneous.

(a) What layout manager is associated with a `JPanel`, and how does it lay out its components?

(b) Give a definition of Java keyword **`this`**. For example, what does it mean in the following procedure call?

```
jbutton.addActionListener(this) ;
```

(c) Draw a variable `b` of type **`int`**`[][]`. Then execute the following assignment, drawing all the objects created and properly assigning to all variables. Put the answer below.

```
b= new int[][] {{2, 4}, {6, 7}};
```

.

Question 8 (12 points). Algorithms.
Write algorithm selection sort as a procedure, with a suitable specification and method header (giving the parameters, for example). The specification should include the precondition and postcondition. You may write them as formulas, pictures, English, or a mixture of these. Here are some requirements.

1. The procedure must sort (only) array segment b[p..q], and not the whole array, where p and q are two of the parameters of the procedure.

2. Use a loop, not recursion. You can use a for-loop or a while-loop. It does not matter.

3. You *must* use a suitable loop invariant. If the invariant is not suitable, you may receive very few points for the procedure body because the loop (and initialization) should be written using the invariant, precondition, and postcondition by following the four loopy questions. If you don't know the invariant, you don't know the algorithm.

3. The repetend of the loop should be written abstractly in English, Java, or a mixture of both; do *not* write a nested loop.