Name (last name ALL CAPS): NetID:

Grades for the final will be posted on the CMS as soon as it is graded, some time tomorrow. Grades for the course will be-posted next week. You can look at your final when you return in the Fall. HAVE A GOOD SUMMER!

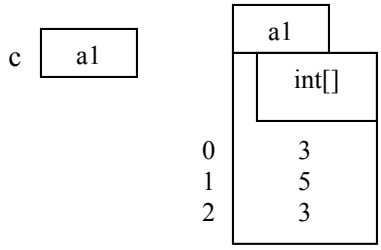
Please submit all requests for regrades for things other than the final BY 8PM TONIGHT. Use the CMS where possible. Regrades for the prelims will not be considered.

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 (2 points). Print your name and **net id** at the top of each page. Please make them legible.

Question 0. _____	(out of 02)
Question 1. _____	(out of 12)
Question 2. _____	(out of 13)
Question 3. _____	(out of 13)
Question 4. _____	(out of 13)
Question 5. _____	(out of 13)
Question 6. _____	(out of 11)
Question 7. _____	(out of 13)
Question 8. _____	(out of 10)
Total _____	(out of 100)

Question 1 (12 points) Executing method calls. Suppose **int** array `c` contains 3 integers, as shown to the right below.



Execute the method call

```
AT.testIsp(c);
```

where class `AT` is given below. (We have labeled the statements with numbers (e.g. L2), which you can use as program counters in a frame for a call). Stop executing when you are ready to execute one of the return statements labeled L5 or L7.

For each call during execution (except the call on `println`), draw the frame for the call.

Hint: You will have to draw 2, 3, or 4 frames for calls.

```
public class AT {
    public static void testIsp(int[] b) {
        L1: System.out.println(isp(b, 0, b.length-1));
    }

    private static boolean isp(int[] b, int h, int k) {
        L2: if (k – h >= 1) {
                L3: boolean c=   b[h] == b[k];
                L4: if (!c)
                        L5: return false;
                L6: return isp(b, h+1, k–1);
        }
        else {
                L7: return true;
        }
    }
}
```

Question 2 (13 points) Recursion and loops. Write a recursive function (defined below) to sum all the values in its argument, whether the argument is an object of class Integer, an object of class Integer[] (1-dimensional array), an object of class Integer[][] (2-dimensional array), an object of class Integer[][][] (3-dimensional array), etc. This is neat! The function is specified below.

To help you, we give you a few ideas.

1. You can use operation **instanceof** to determine whether the base case holds.
2. If ob is an object of wrapper class Integer, use ob.intValue() to obtain the **int** wrapped by the object.
3. If you know that parameter obj is an array (of any number of dimensions), you can cast it to type Object[] and then use it as an array, meaning you can write a for-loop that processes its elements. How will you process its elements?

```
/** =  sum of all integer values in obj.
      Precondition: obj is an object of one of the classes: Integer, Integer[], Integer[][], Integer[][][], etc.
                    If obj is an array, none of its elements is null.
      Examples: Below, a boldface integer like 4 represents an Integer object that contains that integer.
      For the argument 5, the value 5 is returned.
      For the array {1, 2, 3}, 6 is returned because 1+2+3 = 6.
      For the array {{1, 2, 5}, {3, 4}}, 15 is returned because 1+2+5+3+4 = 15.
      For the array {{{1}, {0, 3}, {}}, {{1,2,3}, {3}}}, 13 is returned because 1+0+3+0+1+2+3+3 = 13.
      */
public static int intDeepSum(Object obj) {




}
```

Question 3 (13 points). Loops and arrays. Write the body of the method specified below. It can be used to fill in any side of a triangle with successive values, as shown to the right. So, you can use it to fill in values of the right side, starting with 0, the base, in reverse order, starting at 6, or the left side, in reverse order, starting with 12. In the diagram to the right, xx stands for a value in which we are not interested.

```
          00
       17    01
    16    xx    02
 15    xx    xx    03
14   xx    xx    xx    04
13  xx    xx    xx    xx    05
12  11  10  09  08  07  06
```

The upper diagram to the right is a nice way to look at the triangle, but actually, the elements are in a two-dimensional array b. In the diagram to the right, the top of the triangle is b[1][2]. If the top is in b[x][y], the left side of the triangle always appears in b[x][y], b[x+1][y], b[x+2][y], ….

```
xx xx xx
xx xx 00
xx xx 17 01
xx xx 16 xx 02
xx xx 15 xx xx 03
xx xx 14 xx xx xx 04
xx xx 13 xx xx xx xx 05
xx xx 12 11 10 09 08 07 06
```

Your method body should consist of a single loop, with initialization. We have given the invariant for the loop, because with the invariant, developing the loop should be easy. Your loop *must* use the invariant. It should refer only to the variables mentioned in the invariant. Do not use or declare any other variables. Note that r and c will have to be changed in the repetend.

```
/** Store integers v, v+1, ... ,  v+n-1 into elements of b starting at b[r][c].
    If direction = 1, store them in b[r][c], b[r+1][c+1], b[r+2][c+2], ….
    If direction = 0, store them in b[r][c], b[r][c-1], b[r][c-2], ….
    If direction = -1, store them in b[r][c], b[r-1][c],b [r-2][c], ….
    Precondition: b is big enough to contain the length-n triangle side under consideration. */
public static void fillRow(int[][] b, int r, int c, int n, int  direction, int v)


   // invariant: The first k values, v, v+1, …, v+k-1, have been placed in the designated elements of b.
                 The next value to place, which is v+k, should be placed in b[r][c].

   for (                          ;                          ;                          ) {









  }
  // Postcondition: the n values v, v+1, v+2, …v+n-1 have been placed appropriately in b

}
```

Question 4 (13 points). Subclasses. A `Vector<Integer> v` contains a list of objects of class `Integer`, e.g. `v` might be [**8**, **3**, **5**]. Here the boldface numbers indicate objects of class `Integer` that wrap (contain) the numbers.

Below, write a class `Ring` that is an extension of class `Vector <Integer>` in which a list like [**8**, **3**, **5**] is handled as a *ring* instead of a list. This means simply that the list has no end: the first element follows the last. Thus, for ring `r` containing [**8**, **3**, **5**], `r.get(3)` is **8**, `r.get(4)` is **3**, etc. Also, `r.get(6)` is **8**.

Subclass `Ring` must have the following methods, *with appropriate specifications*. Write only these methods.

1. `Ring` has a constructor with no parameters; it creates a ring with one value in it, **0**.
2. `Ring` has a constructor with an **int** parameter `n`; it creates a ring consisting of the digits of `n`. For example, **new** `Ring(643)` constructs the ring [**6**, **4**, **3**]. Precondition: n>=0. Note that if `n=0`, the ring should contain one value: **0**.
3. Method `get(i)` of `Vector<Integer>` is overridden by `Ring`. The restriction on `i` is changed to `i>=0`. Whatever value of `i` is given, the notion of wraparound is used to determine the element of the ring to be accessed, as discussed above.

Methods in class Vector<Integer>

```
/** Constructor: an empty Vector */
public Vector<Integer>()

/** = number of elements in Vector */
public int size()

/** Append x to Vector's end*/
public boolean add(Integer x)

/** = element i.
      Precondition: 0 <= i < size() */
public Integer get(int i)

/** Set element i of the Vector to x
and return the element initially
there*/
public Integer set(int i, Integer x)

/** a representation of this list, in the
form "[e0, e1, e2, …, en]" where the
ei are the elements, and adjacent pairs
are separated by ",". */
public String toString()
```

Assume this method is also in class Ring. You don't have to write it, but you can use it.

```
/** Store the digits of n in v, with
      most significant first.
    If n = 0, v will be empty.
    Pre: v not null and is empty. */
public static void putIn
            (Vector<Integer> v, int n)
```

Question 5 (13 points). Classes. For our purposes, a repeating decimal is a number `n` in the range `0<=n<1` that can be written in the form of a non-repeating part followed by a repeating part. For example, ¼ is .25000000… Here, the non-repeating part is 25 and the repeating part is 0. Here's a nice representation of this number: 25[0]. The table to the right gives more examples.

number:	.24567567…
nonrepeating part:	24
repeating part	567
representation	24[567]
number:	.74333333…
nonrepeating part:	74
repeating part	3
representation	74[3]
number:	. 33333…
nonrepeating part:	
repeating part	3
representation	[3]

Such repeating decimals are also called *rational numbers*.

Write a class `RepeatingDecimal` that implements repeating decimals in a way that makes it easy to get at (and perhaps change) any digit of such a repeating decimal. For example, if `r` is an object of the class that contains 25[0], then function call `r.digit(0)` = 2, `r.digit(1)` = 5, and `r.digit(i)` = 0 for `i>1`.

Here are the rules for you to follow.

1. Put in class invariants and specify all methods.
2. Keep the non-repeating part in a `Vector<Integer>`, the repeating part in a `Ring` (see question 4).
3. Write a constructor that is given the non-repeating part and repeating parts as **`int`**s. You don't need a loop or anything like that; class `Ring` contains the methods you need.
4. Function `digit(int i)` should return the digit of the repeating decimal at position `i`.
5. Function `toString()` gives the nice representation of this number, e.g. "24[567] " for the first example in the table above. You can write this function without loops if you use the `toString` function in `Vector` and remember that `s.replaceAll(s1, s2)` replaces all occurrences of String `s1`in `s` by String `s2`.

Question 6 (11 points). Exception handling. (a) On the back of the previous page, write a class definition for a class `OverflowException`, whose instances may be thrown.

(b) An instance of the class declared below maintains a list of prime numbers in an array. Complete, in this order, the bodies of procedures `check`, `add`, and `addPrime`. Note the comment in the body of procedure `addPrime`, which explains how you should write the body.

```
/** An instance maintains a list of prime numbers, with duplicates allowed */
public class Primes {
   public static final int MAX= 20; // max number of primes allowed

   private int[] plist= new int[MAX]; // List of prime numbers is in p[0..n-1].
   private int n= 0; // 0 <= n <= MAX

   /** Throw an IllegalArgumentException with a suitable message, if p is not a prime */
   private static void check(int p) {
      if (p < 2)
         _____________________________________________________________;
      // throw an exception if some integer in 2..p-1 divides p
      // invariant: no number in 0..k-1 divides p

      for (int k= _____________; _________________; ____________________ ) {
         if (p%k == 0)
            ____________________________________________________________;
      }
   }

   /** Constructor: an instance with an empty list of primes */
   public Primes() {}

   /** Add prime p to the list.
       Throw an IllegalArgumentException with suitable message if p is not a prime.
       Throw an OverflowException if there is no room for p. */
   public void add(int p) {




   }

   /** If p is a prime, add it to the list; otherwise, print "Mistake: p is not a prime."
       Throw an OverflowException if there is no room for this prime. */
   public void addPrime(int p) {
      // This body should NOT use an if-statement. Instead, use a single try-catch statement */




   }
}
```

Question 7 (13 points). Algorithms. Write a function that implements algorithm partition, as given below. Note that it partitions b[p..q]. You must: (1) Write the precondition, as a picture, right under the first line. (2) Write the postcondition, as a picture, under the precondition. (3) Say as part of the specification what value is returned. (4) Write the invariant, as a picture, in the place provided. (5) Write the loop, with its initialization before the invariant and the loop after the invariant. (6) Write the return statement.

```
/** Algorithm partition, on an array segment b[p..q]

  precondition:



  postcondition:



  what is returned?
*/
public static int partition(int[] b, int p, int q) {



    /** invariant:

    */









}
```

Question 8 (10 points) Miscellaneous.

(a) Name the layout managers associated with objects of class JFrame and Box and explain how components are laid out with each of them.

(b) What is the purpose of making a class abstract, and how do you make a class abstract?

(c) What is the purpose of making a method abstract, and how do you make a method abstract?