Grades for the final will be posted on the CMS as soon as it is graded, hopefully tonight but perhaps tomorrow. Grades for the course will take a few days more. You can look at your final when you return in the fall. HAVE A NICE SUMMER!

Please submit all requests for regrades for things other than the final BY 8PM TONIGHT. 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 (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 stores the second largest number in **int** array `b` in variable `big2`. 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. If it helps for you to draw the invariant as a picture, do so.

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)

```
// precondition: values in b are all different and b.length ≥ 2.

// invariant: big1 is the largest int in b[k..b.length-1],
//            big2 is the second largest int in b[k..b.length-1], and
//            0 <= k <= b.length-2

while (                                        ) {




}
// postcondition: big1 is the largest int in b[0..b.length-1] and
//                big2 is the second largest int in b[0..b.length-1].
```

Question 2 (12 points). Exception handling. (a) On the back of the previous page, write a class definition for a class `OutOfSpaceException` whose instances may be thrown.

(b) An instance of the class declared below maintains a list of integers. There is a maximum size for this list, which is given in the constructor call. Note the class invariant. You job is to write the bodies of the constructor, procedure `add`, and procedure `messageAdd`. Please note the comment in the body of procedure `messageAdd`, which explains how we want you to write the body.

```
/** An instance maintains a list of integers */
public class intList {
   private int[] list; // The list of elements is list[0..n-1],
   private int n;      // so the list contains n elements.

   /** Constructor: a new list with 0 elements and
       a capacity of m elements (it can hold at most m elements. */
   public intList(int m) {




   }

   /** append v to the list —but Throw an OutOfSpaceException if there is no space for it */
   public void add(int v) throws OutOfSpaceException {









   }


   /** If there is room in the list for v, then append it; otherwise print message "no space" */
   public void messageAdd(int v) {
      // This body should NOT use an if-statement. Instead, use a try-catch-statement.









   }
}
```

Question 3 (12 points). Interfaces. At the bottom of this page is an interface `Komparable`.

(a) Write a class `Time` that:

(1) Maintains the hour of the day (range 0..23) and minute of the hour (range 0..59) in two fields;

(2) Has a constructor with one parameter, the minutes of the day (in range 0..24*60–1);

(3) Implements interface `Komparable`. Look carefully at the spec of the method declared in Komparable. For times `t1` and `t2`, an obvious choice is for `t1.less(t2)` to be true if `t1` comes before `t2`.

(4) You don't have to declare any other fields or methods in the class.

(b) Write an initializing declaration of an array `b` whose elements implement `Komparable` and that has two elements of class `Time`: the first has `Time` of 1 hour and the second a `Time` of 2 hours 1 minute.

}

```
public interface Komparable {
    /** = "This object is less than t".
        Throw a ClassCastException if this object and t are not of the same class*/
    public boolean less(Object t);
}
```

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

Recordings.recordingSession();

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 recordingSession. Then, as you execute the sequence, draw all objects created and execute any assignment statement faithfully. You don't have to do this, but you will have a better chance of getting correct answers if you do.

```
public class Sound {
  /* Total duration of all instances of Sound */
  private static double totalDuration=  0.0;

  private double duration; // duration of sound

  /** Constructor: new Sound with duration m*/
  public Sound( double m) {
     duration=  m;
     totalDuration=  totalDuration + duration;
  }
/** = duration of this sound */
  public double getDuration()
     { return duration; }

  /** = the total duration of all sounds */
  public static double getTotalDuration()
     { return totalDuration; }

  /**Rreplace sound with new one of duration m */
  public void overwrite(double m) {
     totalDuration =  totalDuration – duration + m;
     duration =  m;
  }

  /** Append to this sound a new sound of
       duration m */
  public void append(double m) {
       duration=  duration + m;
       totalDuration=  totalDuration  + m;
  }
}
```

```
public class Bird extends Sound {
  /* total number of instances of Bird created */
  private static int numBirds=  0;

  private String bird; // name of bird
  private String date; // date recorded (in form
                       // yyyy.mm.dd)

  /** Constructor: a new Bird with name b,
                   date d, and duration m*/
  public Bird(String b, String d, double m) {
    super(m);
    bird=  b;      date =  d;
    numBirds=  numBirds + 1;
  }

  /** = "<name> - <date> : <duration>" */
  public String toString(){
     return bird + " - " + date + " : " +
                getDuration();
  }

  // = this bird's name
  public String getBird()
     { return bird;  }
}
```

```
public class Recordings {
  public static void recordingSession() {
      Sound first= new Bird("Robin", new String("2008.05.08"), 6);
      Bird forth= new Bird("Starling", "2007.01.01", 12);
      Sound chirp= new Sound(30);
      Bird fifth= new Bird("Oriole", new String("2008.01.01"), 10);
      Sound tweet= forth;
      Bird second= (Bird) first;
      tweet.overwrite(10);
      first.append(3.0);
      System.out.println(second);



      System.out.println(chirp);



      System.out.println(tweet);



      System.out.println("Length of sound: " + Sound.getTotalDuration());



      System.out.println("Bird is: " + second.getBird());



      System.out.println("Bird is: " + ((Bird)first).getBird());



      System.out.println("Birds are the same?"  +  (second.getBird() == fifth.getBird()));



      System.out.println("Birds are the same?"  +  (second.getBird().equals(fifth.getBird())));




  }
}
```

Question 5 (12 points). Recursion. Write a recursive function, *complete with a precise and thorough specification*, for the following problem. Given an integer > 0, none of whose decimal digits are 0, the function "complements" each of its decimal digits, changing an 9 into 1, 8 into 2, …, 2 into 8, and 1 into 9.

For example, the result for the integer 93723 is 17387.

Here are some ground rules. The function should be static. Make sure the specification has an appropriate precondition (e.g. the parameter is > 0). Make sure recursive calls satisfy the precondition. Do not use strings.

Question 6 (15 points). Classes. Write two classes to maintain information about your friends' birthdays. *Write specifications for all methods*. You will be using class `Vector`. If you are not sure what the `Vector` methods are, do what you think is right and put comments that explain your method calls. Make fields and method private or public as appropriate.

(a) Write a class `BirthDay`. An instance maintains a month (in the range 1..12) and a day that has a range that depends on the month. To simplify things, assume that all months have exactly 30 days except February, which has 28. There should also be a static list of all `BirthDay` objects created so far (a `Vector` of `BirthDay`).

`BirthDay` needs a constructor that allows a user to give values for the month and the day, with a precondition that this is a valid date. It needs getter methods for the two fields. Finally, it needs a constructor with only one parameter, a month; the implicit day is the last day of the month. The second constructor must not set the fields directly but should use a call on the first constructor. Finally, `BirthDay` needs a `toString` function that yields the date in the form month.day.

The class can have other methods if you feel the need.

(b) Some friends celebrate their birthday on a different date due to conflicts with national holidays, final exams, etc. Write a subclass `VirtualBirthDay` of `BirthDay`. Besides having the actual month and day, a `VirtualBirthDay` has a shift of days. So, the birthday will be celebrate `x` days after/before the actual date if the shift was `+/- x`. You design the implementation. The constructor should allow the month, day, and shift to be given by a user. A precondition should be that the shift is in the range -10..10 and that the real and virtual birthdays be in the same year.

For example, the date 5.1 with a shift of -1 is 4.30, and the date 5.1 with a shift of -2 is 4.29.

Override method `toString` to yield the virtual birth date. FOR THIS PART, YOU MAY ASSUME THAT ALL MONTHS HAVE 30 DAYS, just to simplify things for you on this test.

Question 7 (12 points). Miscellaneous.

(a) Execute the following sequence and write down the output of any `println` statement to the right of those println statements.

```
Integer m1= new Integer(8);
Integer m2= new Integer(8);
if (m1.equals(m2))
   System.out.println("Test1");
if (m1 != m2)
   System.out.println("Test2");
if (m1 instanceof Object)
   System.out.println("Test3");
```

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

(c) Give a definition of Java keyword **this**. For example, what does it mean in a method call like this?

jbutton.addActionListener(**this**) ;

(d) To the right is a definition of a class Qdd.

First, draw the file drawer for Qdd just before execution starts.

Second, write down the steps in evaluating the expression **new** Qdd(3, 4). With each, actually do the step yourself, drawing any objects and calls for frames that are required, putting the objects where they belong (place frames for calls anywhere on the page).

```
public class Qdd {
   public static int s= 5;
   public int f;
   public Qdd(int f, int t) {
      this.f= f;
      s= s + t;
   }
}
```

If you have to execute a method call, draw the frame for the call, but you do not have to explain the steps in executing the method call.

Question 8 (10 points). Algorithms.
Write algorithm binary search as a function, 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 or as pictures.

Requirements. The function must search a complete sorted int array `b` for a value, as we have described many times. Your specification must say what value (an **int**) it returns.

To help you check our answer, we give your hints. After writing your answer, read the preceding paragraphs again and then think about the following.

1. Did you start out by writing a specification (precondition and postcondition) and header for the function? We want to see the specification first, written well. Does your specification say what value is returned?

2. Does your function search the complete array `b`?

3. Did you write a suitable invariant before writing the loop, and does your loop conform to the invariant?

4. Did you write a return statement?