NAME: _____

EE 322C Spring 2009 Exam 1 Name: _____

READ THIS FIRST. Please use the back side of each page for scratch paper. For some of the questions you may need to think quite a bit before you write down an answer. If you write while you think, use the back side of the page and copy your final (neatly written) answer to the front side. I will not give credit for illegible answers (and I reserve sole judgment for what is "legible"). The exam is out of 100 points, not all questions are equally hard or take equally long to complete. Some hard questions are not worth very much. Some easy questions are worth a lot. Good luck!

1. (20 pts) Indicate what the main program will print

```
class Foo {
   int x;
   Foo () { x = 13; }
   public void doit()  {  System.out.println("Base");   }
   public void fred() { System.out.println(x); }
}

class Bar extends Foo {
   int x;
   Bar() { x = 14; }
   public void doit() { System.out.println("Derived");  }
 }
class Question1 {
  public static void main(String[] not_used) {
     Foo x = new Foo();
     Bar y = new Bar();
     Foo z = y;

     x.doit();   // (5 pts) prints: _____


     y.doit(); // (5 pts) prints: _____


     y.fred(); // (5 pts) prints: _____


     z.fred(); // (5 pts) prints: _____
  }
```

2. (20 pts) Give an expression in “big-Oh” notation for the amount of time the following functions will take in the *worst case*.

a. (5 pts) _____

```
void doit(int N) {
   if (N % 2 == 0) {
      for (int k = 0; k != N; k += 2) {
      }
   }
}
```

b. (5 pts) _____

```
void doit(int N) {
  while (N % 2 != 0) {
    N = N / 2;
  }
}
```

c. (5 pts) _____

```
int doit(int x[], int N) {
   int s = 0;
   for (int k = 0; k < N; k += 1) {
      s += x[k];
      if (k > N / 2) {
         for (int j = 0; j < N; j += 1) {
            s += x[k];
         }
         return s;
      }
   }
   return s;
}
```

d. (5 pts) _____

```
void doit(int N) {
   for (int k = 1; k < N; k *= 2) {
      for (int j = 1; j < k; j += 2) {
      }
   }
}
```

3. Recall Project2 (our SortedSet class). Assume I want to redesign Project2 to use your ExArray class (Project 3) as the foundation. I'm considering the following implementation for the *add* method. Note that all elements inserted into a SortedSet must implement the Comparable interface (i.e., the class must be a subtype of Comparable).

```
class MySortedSet {
  ExArray data;
  public SortedSet() { data = new ExArray(); }
  public int size() { return data.size(); }
  public boolean add(Object x) {
    if (contains(x)) { return false; }
    data.add(x);
    sort(data, size.size());
    return true;
  }
```

   a. (6 pts) Which sorting algorithm provides the best performance, *Insert Sort* or *Quick Sort*. Explain.

   b. (8 pts) What is the **total** time complexity to insert N items into the SortedSet by calling add N times. Please assume the values are chosen at random and the SortedSet is initially empty. (please… I'm looking for the **total** time for N calls to add, not the time to call it once.)

4. Recall that in Project 2, you wrote a function *addAll* that computed the union of two sets. The argument to *addAll* was also a SortedSet, and we achieved a time complexity of O(N) (assuming both sets have N elements). Assume that function is renamed to *addAll_fast* and we want to design a new addAll method that is more general purpose. How would you design an addAll method that takes a Collection as an argument (i.e., not necessarily a SortedSet) if your goal was to minimize the time complexity?
    a. (6 pts) Assuming both "this" and "c" each contain N elements (they're not the same elements, they're just the same size), what is the time complexity of the following implementation of addAll? Recall that the elements in a *Collection* are not sorted.

```
public boolean addAll(Collection c) {
  Object[] vals = c.toArray();
  for (int k = 0; k < vals.length; k += 1) {
    add(vals[k]);
  }
}
```

    b. (8 pts) What is the time complexity of the solution below?

```
public boolean addAll(Collection c) {
  Object[] vals = c.toArray();
  sort(vals, vals.length);
  SortedSet that = new SortedSet();
  for (int k = 0; k < vals.length; k += 1) {
    that.data.add(vals[k]);
  }
  addAll_fast(that); // O(N) time complexity
}
```

5. (12 pts) I've written a "pseudo-collection" called PrimeBox. When a PrimeBox is collected, it "contains" all prime numbers between min and max. There aren't really any values stored in the PrimeBox, but it has all the functions a *Collection* normally has. I've shown only a few of the important functions below. I want you to write an Iterator class for my PrimeBox collection and write the *iterator* method.

```
interface Iterator { // don't bother writing remove
   public Object next();
   public boolean hasNext();
}

class PrimeCollection {
  int min;
  int max;
  public PrimeCollection(int s, int e) { min = s; max = e; }
  boolean contains(Object x) {
    if (! x instanceof Integer) { return false; }
    Integer p = (Integer) x;
    int v = p.intValue();
    if (v < min || v > max) { return false; }
    else { return isPrime(v); }
  }

  // it's a pseudo collection, new values can't be added
  public boolean add(Object x) { return false; }

  // not shown, returns true if and only if x is prime
  public static boolean isPrime(int x) { … }
```

a. (8 pts) Write the class for your Iterator here

b. (4 pts) Write the *PrimeBox.iterator()* function here

6. (20 pts) Design and implement a type (either an interface or a class) for Widget. Widgets are created by the function WidgetMaker.makeWidget(). The argument to this function is a Java String, and it can be either “RED” or “BLUE”. All Widgets have a doit function. When the doit function is called, the function must print either “RED” or “BLUE” depending on what argument was used when the Widget was created. For this project, you receive 10 points for designing a solution. You receive 5 points for accurately describing how much memory your solution requires (as a function of N). You receive 5 points for designing a solution that minimizes memory required (provided you can explain why your solution is minimal).

   a. (10 pts) Design your Widget. You can create as many classes or interfaces as you see fit. Be sure to complete the makeWidget function, and that your solution works with the **main** provided below. **You’ll have additional space on the next page**.

```
class WidgetMaker {
   public static Widget makeWidget(String s) { // WRITE THIS
   }
}
class ExamQ
   public static void main(String[] not_used) {
      Widget[] x = new Widget[N]; // for some very large N
      for (int k = 0; k < N; k += 1) {
         if (Math.random() < 0.50) { // flip a coin
            x[k] = WidgetMaker.makeWidget(new String(“RED”));
         } else {
            x[k] = WidgetMaker.makeWidget(new String(“BLUE”));
         }
      }
      for (int k = 0; k < N; k += 1) {
        x[k].doit(); // must print RED or BLUE
      }
   }
}
```

(6a cont.) Define your Widget (class or interface) and any other classes you need here.

b. (5 pts) How much memory (in bytes, please) is required to store N widgets? Explain.

c. (5 pts) What steps did you take to minimize the memory required?