CS 110  
Practice Final Exam  
originally from Winter, 2003

Instructions:  closed books, closed notes, open minds,  
3 hour time limit.

There are 4 sections for a total of 49 points.

| Part I: Basic Concepts, Boolean Logic (pp. 2-3) | 14 pts |
| Part II: Writing Methods (pp. 4-8) | 22 pts |
| Part III: Linear and Binary Search (p. 9) | 8 pts |
| Part IV: Inheritance and Dynamic Binding (pp. 10-11) | 5 pts |
Part I – Basic Concepts and Boolean Logic  (14 points total)

1. Match each programming concept on the left with the appropriate description on the right.
   (6 points)

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>a) descriptions not provided in practice exam</td>
</tr>
<tr>
<td>interface</td>
<td>b) Definitions can be found in the book, or looked up online</td>
</tr>
<tr>
<td>abstract class</td>
<td>c)</td>
</tr>
<tr>
<td>class documentation</td>
<td>d)</td>
</tr>
<tr>
<td>access identifier</td>
<td>e)</td>
</tr>
<tr>
<td>instance variable</td>
<td>f)</td>
</tr>
<tr>
<td>class variable</td>
<td>g)</td>
</tr>
<tr>
<td>local variable</td>
<td>h)</td>
</tr>
<tr>
<td>scope</td>
<td>i)</td>
</tr>
<tr>
<td>constructor</td>
<td>j)</td>
</tr>
<tr>
<td>method</td>
<td>k)</td>
</tr>
<tr>
<td>method signature</td>
<td>l)</td>
</tr>
<tr>
<td>method invocation</td>
<td>m)</td>
</tr>
<tr>
<td>parameter type</td>
<td>n)</td>
</tr>
<tr>
<td>formal parameter</td>
<td>o)</td>
</tr>
<tr>
<td>actual parameter</td>
<td>p)</td>
</tr>
<tr>
<td>return type</td>
<td>q)</td>
</tr>
<tr>
<td>inheritance</td>
<td>r)</td>
</tr>
<tr>
<td>subclass</td>
<td>s)</td>
</tr>
<tr>
<td>superclass</td>
<td>t)</td>
</tr>
<tr>
<td>dynamic binding</td>
<td>u)</td>
</tr>
<tr>
<td>responsibility</td>
<td>v)</td>
</tr>
<tr>
<td>collaboration</td>
<td>w)</td>
</tr>
<tr>
<td>linear (sequential) search</td>
<td>x)</td>
</tr>
<tr>
<td>binary search</td>
<td>y)</td>
</tr>
<tr>
<td>aggregation</td>
<td>z)</td>
</tr>
<tr>
<td>recursion</td>
<td>aa)</td>
</tr>
</tbody>
</table>
2. Circle all of the code fragments below that are logically equivalent to the following statement:

```java
for ( i = 10; i > 0; i-- )
    System.out.println("Hi!");  //prints out the word Hi 10 times
```

(a) i = 10;  equiv
while ( i > 0 )
{    
    i--;
    System.out.println("Hi!");
}

(b) for ( i=1; i < 11; i++ )
    System.out.println("Hi!");  equival;

(c) i = 11;  not equival;
do
{
    System.out.println("Hi!");
    i--;
}
 while ( i > 0 );

(d) for ( i=0; i < 10; i++ )
    System.out.println("Hi!");  equival;

(e) i = 0;  not equival;
while ( i < 10 )
{
    System.out.println("Hi!");
    i--;
}

(f) for ( i=0; i < 20; i+=2 )
    System.out.println("Hi!");  equival;

3. Is the following code segment logically equivalent to the code formed by removing all occurrences of the word else? Explain your answer. (2 points)

```java
if ( i > 0 )
    return false;
else if ( j <= i )
    return false;
else return true;
```

Yes, it would be equivalent if the word else was removed, because the return statement transfers execution to the code elsewhere in the program. Any code after a return statement will not be executed.
Part II – Write Methods  (22 points total)

4. Write a method, `isReverse`, that takes two `ArrayList` parameters, x and y, and returns true if the values in y are the same as those in x, except in the opposite order. That is, `isReverse` should return true if the two lists have the same number of elements and the first element of x is the last element of y, the second element of x is the second to last element of y, and so on. The method should return `false` otherwise.

(4 Pts)

```java
public boolean isReverse(ArrayList<String> x, ArrayList<String> y)
{
    if (x.size() != y.size())  // can’t be reverse if
        return false;  // sizes are different
    for (int i=0; i < x.size(); i++)
    {
        int j = y.size() - 1 - i;
        if (x.get(i).equals(y.get(j)) == false)
            return false;
    }
    // at this point, there were no mis-matched
    // elements, so return true
    return true;
}
```

Alternate:

```java
public boolean isReverse(ArrayList<String> x, ArrayList<String> y)
{
    if (x.size() != y.size())  // can’t be reverse if
        return false;  // sizes are different
    for (int i=0, j = y.size() - 1; i < x.size(); i++, j--)
    {
        if (!x.get(i).equals(y.get(j)) )
            return false;
    }
    // at this point, there were no mis-matched
    // elements, so return true
    return true;
}
5. The standard Java java.util.GregorianCalendar class has a method that determines whether or not a given year is a leap year.

   public boolean isLeapYear(int year)

Write a driver that tests this method. Note that any year that is divisible by 4 is a leap year except those years that are divisible by 100 but not by 400 (for example, 1900 is not a leap year but 2000 is a leap year). Explain why you included each test case. (4 Pts)

```java
import java.util.GregorianCalendar;
public class MyDriverClass
{
    public static void main(String[] args)
    {
        GregorianCalendar cal = new GregorianCalendar();
        System.out.println(“Year:   Expected    Actual”);
        System.out.println(“2003:   false    “ +
                   cal.isLeapYear(2003));  // not divisible by 4
        // Additional test cases:
        System.out.println(“1900:   false   “ +
                   cal.isLeapYear(1900));  // divisible by 100 but
                   // not by 400
        System.out.println(“2000:    true   “ +
                   cal.isLeapYear(2000));  // divisible by 100 and 400
        System.out.println(“2016:    true   “ +
                   cal.isLeapYear(2016));  // divisible by 4 but
                   // not by 100 or 400
        System.out.println(“2015:    false   “ +
                   cal.isLeapYear(2015));  // test both sides of 2016
        System.out.println(“2017:    false   “ +
                   cal.isLeapYear(2017));  // test both sides of 2016
```
Questions 6 - 8 refer to the Date, Flight and Airport classes provided on the supplemental sheet. You may assume that all methods you are not writing have been implemented correctly.

6. Implement the Flight method late, which returns true if the flight has arrived (the actual time is initialized) and the actual arrival time is later than the estimated arrival time. The late method also returns true if the flight has not arrived, but the current time is later than the estimated arrival time. In all other cases late returns false. Note that you can get the current date and time by constructing a new Date object.

(4 pts)

```java
public class Flight {
    public boolean late() {
        Date now = Date();
        Date eta = this.eta();
        Date ata = this.ata();
        if (ata != null) // flight has arrived?
            if (ata.after(eta)) // actual arrival after estimated arr
                return true;
            else
                return false;
        else // flight not arrived
            if (now.after(eta)) // current time after estimated
                return true;
            else
                return false;
    }
}
```

Alternate:

```java
public boolean late() {
    Date now = Date();
    Date eta = this.eta();
    Date ata = this.ata();
    if (ata != null && ata.after(eta)) // actual arrival after estimated arr
        return true;
    else if (ata == null && now.after(eta))
        return true;
    return false;
}
```
7. Implement the Airport method `nbrLateFlights`, which returns the number of the specified airline’s flights that are late today.

(5 pts)

```java
public class Airport {
    public int nbrLateFlights (String airline) {
        int count = 0;
        for (Flight f : schedule) {
            if (f.airline().equals(airline) && f.late())
                count++;
        }
        return count;
    }
}
```
8. Implement the Airport method mostLateFlights, which returns the name of the airline with the most late flights today. (5 pts)

   public class Airport
   {
   public String mostLateFlights()
   {
      if (airlines.isEmpty())
         return null;
      String latestAirline = airlines.get(0);
      int maxLate = 0;
      for (int i = 1; i < airlines.size(); i++)
      {
         String s = airlines.get(i);
         if (this.nbrLateFlights(s) > maxLate)
         {
            latestAirline = s;
            maxLate = this.nbrLateFlights(s);
         }
      }
      return latestAirline;
   }

   Alternate:

   public String mostLateFlights()
   {
      String latestAirline = null;
      int maxLate = 0;
      for (String s : airlines)
      {
         int newNum = this.nbrLateFlights(s);
         if (newNum > maxLate)
         {
            latestAirline = s;
            maxLate = newNum;
         }
      }
      return latestAirline;
   }

Part III – Sequential and Binary Search (8 points total)

9. We have discussed linear and binary search techniques. Consider an array, A, containing the following 16 elements:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>11</td>
<td>13</td>
<td>25</td>
<td>26</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>43</td>
<td>44</td>
<td>47</td>
<td>55</td>
<td>59</td>
<td>60</td>
<td>62</td>
<td>73</td>
</tr>
</tbody>
</table>

a) How many elements must we examine to determine whether the value 30 is in array A using a linear search? (1 pt)
   1 2 3 4 5 6

b) How many elements must we examine to determine whether the value 30 is in array A using a binary search? (1 pt)
   1 2 3 4 5 6

c) How many elements must we examine to determine whether the value 11 is in array A using a linear search? (1 pt)
   1 2 3 4 5 6

d) How many elements must we examine to determine whether the value 11 is in array A using a binary search? (1 pt)
   1 2 3 4 5 6

10. Suppose we define a structure, called a linked list, which can store all of the same types of information that an array can. The only way to access the elements in a linked list is to start at the beginning of the list and step through all the items in it. (We say a linked list does not have random access because you cannot access the $i^{th}$ element directly; you have to start at the beginning and step through until you get to the $i^{th}$ element.) Pictorially, a linked list may look like the following:

   Start: 11 -> 13 -> 25 -> 26 -> 29 -> 30 -> |

For each of the following types of data, would a linear or binary search be the better choice? (4 pts)

   a) an array of sorted data: **binary search**

   b) an array of unsorted data: **linear search**

   c) a linked list of sorted data: **linear search**

   d) a linked list of unsorted data: **linear search**
Part IV – Inheritance and Dynamic Binding (5 points total)

11. What inheritance relationships would you establish among the following classes?

   Student  
   Professor  
   TeachingAssistant  
   DepartmentChair  
   Person  
   Course  
   Seminar  
   Lecture  
   ComputerLab

Draw pictures (they could look like trees or like object diagrams) of the various classes and the inheritance relationships. Label the lines showing the relationships to make it clear the type and direction of the relationships. Do not try to indicate all the other relationships that might exist among the classes.

(5 pts)

Something like the following would work:

Person

Student

TA

Employee

Professor

Dept Chair

Course

Seminar

Lecture

ComputerLab (if room rather than course)