Faculty of Arts & Sciences
Department of Computer Science
CMPS 200—Introduction to Computer Programming
Fall 2005–2006
Monday, December 19, 2005

Exam 2

Version 1

Name:________________________________ **Student Id:**__________________________

Signature:______________________________ **Section:**

Lect I	**10:00–11:00**
Lect II	**1:00 –2:00**

Answers to Part I

Question	**A**	**B**	**C**	**D**	**E**
1.					
2.					
3.					
4.					
5.					
6.					
7.					
8.					
9.					
10.					
11.					
12.					
13.					
14.					
15.					
16.					
17.					
18.					
19.					
20.					
21.					
22.					
23.					
24.					
25.					

Part I	50	
Part II	50	
1.1	4	
1.2	15	
1.3	15	
1.4	7	
2	9	
Total	**100**	

Part II

Answer the following questions in the space provided.

Problem 1. Musical Instruments

Class `MusicInstrument`

`MusicInstrument` is an abstract class that represents a basic musical instrument. It defines the following data values:

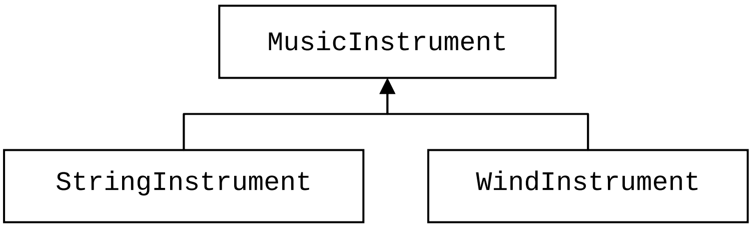
- **`name`**: represents the name of the musical instrument (a String).
- **`computeVolumeLevel`**: represents the volume level of the instrument (a double).

These instance variables are declared so they are inherited by subclasses of the `MusicInstrument` class.

The `MusicInstrument` class has a constructor and two methods:

- **`getVolume`**: is an **abstract** method to be implemented by subclasses.
- **`toString`**: returns a string representing information about the `MusicInstrument` object (its name).

The `MusicInstrument` class has two subclasses: `StringInstrument` and `WindInstrument`.



Class `WindInstrument`

The `WindInstrument` class specializes the `MusicInstrument` class by having a number of holes that produce different sounds or tunes when air is blown in the instrument. The `WindInstrument` class defines a new instance variable, and a constant coefficient:

- **`numberOfHoles`**: initially set to a number less than 10 when a new `WindInstrument` is created.
- **`WCOEF`**: wind coefficient used to calculate the volume level of the instrument. It is set to 8.5.

The `WindInstrument` class has a constructor and two methods:

- **`computeVolumeLevel`**: computes and returns the volumeLevel of the wind instrument.
- **`toString`**: returns information about the `WindInstrument` object (name, and number of holes).

Class `StringInstrument`

The `StringInstrument` class specializes the `MusicInstrument` class by having a number of string chords that produce different sounds or tunes when bent. The `StringInstrument` class defines new instance variables:

- **`numberOfStrings`**: initially set to a positive number when a new `StringInstrument` is created.
- **`length`**: the length of the strings.
- **`SCOEF`**: string coefficient used to calculate the volume level of the instrument. It is set to 2.0.

The `StringInstrument` class has a constructor, and two methods:

- **`computeVolumeLevel`**: computes and returns the `volumeLevel` of the string instrument.
- **`toString`**: returns information about the `StringInstrument` object (name, number of strings, and length of its strings).

Problem 1.1. Fill in the blanks to complete the definition of the `MusicInstrument` class.

```
public abstract class MusicInstrument
{
    // (2 points) Declare instance variables
    // Note that the instance variables will be inherited by
    // the subclasses of the MusicInstrument class

   ....................................................................................

   ....................................................................................

   ....................................................................................

   ....................................................................................



    // Constructor MusicInstrument initializes an instrument’s name, and
    // volumeLevel

    public MusicInstrument (String instName)
    {
      name = instName;
      volumeLevel = 0.0;
    }


    // method toString returns a string with the name of the instrument

    public String toString()
    {
      return "Musical Instrument: " + name;
    }

    // (2 points) method computeVolumeLevel: an abstract method that takes
    // no parameters and returns a double.

   ....................................................................................

   ....................................................................................


} //end of class MusicInstrument
```

Problem 1.2. Fill in the blanks to complete the definition of the **`WindInstrument`** class.

```
// (1 point) header for class WindInstrument

......................................................................................

{
    // declaration of constants
    private final double WCOEF = 8.5;

    // (1 point) Declaration of instance variable numberOfHoles

    ..................................................................................

    // (5 points) WindInstrument Constructor:
    // Two parameters: name and numberOfHoles.
    // Uses the ternary conditional operator to make sure that the
    // number of holes parameter is positive and less than eleven.
    // It defaults an illegal value to 5.0.

    ..................................................................................

    ..................................................................................

    ..................................................................................

    ..................................................................................

    ..................................................................................

    // (3 points) method computeVolumeLevel:
    // Implement implements the abstract method to Compute the volumeLevel
    // as the product of the number of holes and the wind coefficient.
    // Returns the computed volumeLevel.

         .............................................................................

         .............................................................................

         .............................................................................

         .............................................................................

         .............................................................................

    // (5 points) method toString:
    // Returns a string with the information about the instrument,
    // in addition to the number of holes.

         .............................................................................

         .............................................................................

         .............................................................................

}
```

Problem 1.3. Fill in the blanks to complete the definition of the `WindInstrument` class.

```
// (1 point) header for class StringInstrument

.........................................................................................

{
   // Declaration of constants
   private final double SCOEF = 2.0;

   // (1 points) Declaration of instance variables numberOfStrings and
   //    length

   ......................................................................................

   ......................................................................................

   // (5 points) StringInstrument Constructor:
   // Three parameters: name, numberOfStrings, and length.
   // Uses the ternary conditional operator to make sure that the number of
   // strings, and length parameters are positive.  Defaults illegal values
   // to 6.0 and 0.5 respectively.

   ......................................................................................

   ......................................................................................

   ......................................................................................

   ......................................................................................

   ......................................................................................

   ......................................................................................

   // (3 points) method computeVolumeLevel:
   // Computes volumeLevel as the product of the number of strings and
   // the string coefficient divided by the length.
   // returns the computed volumeLevel.

   ......................................................................................

   ......................................................................................

   ......................................................................................

   ......................................................................................

   ......................................................................................

   ......................................................................................

   ......................................................................................
```

```
// (5 points) method toString:
// returns a string with the information about the instrument,
// in addition to the number of strings and their length.
```

....

....

....

....

....

....

....

Problem 1.4. `Orchestra` is a driver class. Its main method creates two groups of musical instruments: a string section (`WindInstrument` array) and a wind section (`StringInstrument` array). This method computes the total volume level of each section and issues a report displaying all instruments with their names and properties. It invokes the `isBalanced` class method to determine if the orchestra's volume is balanced and reports it back to the user. The driver class's sample output and its main method are given below.

Sample Output:

```
----------------------------
Instruments in this band are:
String Instruments
-------------------
Musical Instrument: Violin with 4 strings of length 0.45
Musical Instrument: Guitar with 6 strings of length 0.6
Musical Instrument: Lute with 12 strings of length 0.5
Musical Instrument: Cello with 4 strings of length 0.7

WindInstruments
-------------------
Musical Instrument: Flute with 5 holes
Musical Instrument: Clarinet with 4 holes
Musical Instrument: trombone with 2 holes

The volume level of the band is balanced
```

```
public class Orchestra
{
   public static void main (String[] args)
   {
      // Build a group of string instruments & a group of wind instruments
      StringInstrument[] stringGroup = {
         new StringInstrument("Violin", 4, 0.45),
         new StringInstrument("Guitar", 6, 0.60),
         new StringInstrument("Lute",12, 0.50),
         new StringInstrument("Cello",4,0.70)
     };

     WindInstrument[] windGroup = {
        new WindInstrument("Flute",5),
        new WindInstrument("Clarinet",4),
        new WindInstrument("trombone",2)
     };

     // Display the report
     System.out.println ("---------------------------");
     System.out.println ("Instruments in this band are:");
     System.out.println ("String Instruments");
     System.out.println ("-----------------");

     for (StringInstrument s: stringGroup)
       System.out.println (s);

     System.out.println ("\nWindInstruments");
     System.out.println ("--------------");
     for (WindInstrument w: windGroup)
       System.out.println (w);

     System.out.print("\nThe volume level of the band is ");
     System.out.println(isBalanced(stringGroup, windGroup)
                        ? "balanced" : "not balanced");
   }
   ...
 }
```

Implement the `isBalanced` method to complete the driver class `Orchestra`.

```
    // (7 points) Class method isBalanced:
    // Two parameters: StringInstrument array and WindInstruments array.
    // Computes the total volumeLevel for each array.
    // Returns true if the difference between the volume levels of the
    //   two arrays does not exceed 10.0; false otherwise.
```

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

```
}
```

Problem 2.Sorted Integer List Class

You work for a software company that is developing a library of Java utility classes. Your boss comes to you one morning and tells you that one of your group members has resigned abruptly and that work within the group is being reorganized. As a result of this reorganization, you are asked to implement several methods in order to complete the partially implemented `SortedIntList` class.

The purpose of the class `SortedIntList` is to organize integers in ascending (increasing) order. The class already defines two private instance variables, `list` (`int` array) to store the integers and `count` (`int`) to keep track of the number of integers already stored in the `list` array. The declarations of the instance variables are as follows:

```
private int   count;
private int[] list;
```

The class already implements a constructor which takes no parameters. It also implements the `insert` method which takes an `int` as a parameter, inserts it in the appropriate location in the array so as to maintain the sorted nature of the list, and increments count by 1. The `insert` method will increase the capacity of the object if it does not have room to store the `int` value passed in as a parameter.

Implement the **`delete`** method in order to complete the implementation of the `SortedIntList` class. This method takes an `int` parameter and deletes it from the list of `int` numbers stored in the `SortedIntList` object. It updates the count of numbers if need be. The method returns `true` if it finds and successfully deletes the number of the list; it returns `false` otherwise.

To illustrate the functionality of this method, let us consider the following example. A `SortedIntList` instance stores the squares of the first 10 integers in its `list` array; the value of its `count` variable is 10.

0	1	2	3	4	5	6	7	8	9
0	1	4	9	16	25	36	49	64	81

Invoking `delete(3)` on this object returns `false` since the value 3 is not found in the array. Invoking `delete(25)` returns `true`; it changes the value of `count` to 9 and the contents of the list array to as follows where the ellipses character (`...`) indicates that the contents of location 9 of the array are irrelevant:

0	1	2	3	4	5	6	7	8	9
0	1	4	9	16	36	49	64	81	...

Implement the `delete` method to complete the `SortedIntList` class: **(9 points)**