CSC 1350 Pre-Final Exam

Section (3/4)

November 13, 2019

NAME: _____

- This exam consists of five exercises.
- Use only Java^{TM}Standard Platform Edition 8.0 compliant syntax in your code.
- Blue book is required. Fill in the information on the cover of your blue book and on the exam sheet.
- Do exercise C.(i) on the exam sheet and all other exercises in your blue book.
- Calculators are not allowed.
- Use the back of the exam sheets if you need scratch paper.
- Read the instructions preceding each section carefully before beginning the section.
- Turn in the exam and your blue book before you leave.

DURATION: 120 Minutes

Table 1: Distribution of Points

Exercise	WORTH	SCORE
A	$x_1 = 20$	
B	$x_2 = 20$	
C	$x_3 = 20$	
D	$x_4 = 20$	
E	$x_5 = 20$	
Total	$\sum_{i=1}^{5} x_i$	
Exam Score	100	/100

DO NOT TURN THIS PAGE UNTIL YOU ARE TOLD TO DO SO.

Exercises

Instruction: Read each question carefully before answering it.

A. Consider the code segment below.

Listing 1: Code Segment

```
int[][] x = {{1},
             {2, 0},
             {3, 0, 0},
             {4, 0, 0, 0},
             {5, 0, 0, 0, 0}};

int i,j;
for (i=0; i<x.length; i++)
{
   for (j=1; j<x[i].length; j++)
   {
      x[i][j] = x[i][0]*(j+1);
   }
}
System.out.println(Arrays.deepToString(x));
```

What would this code segment output when it is executed?

B. Consider the implementation of the bubble sort algorithm below.

Listing 2: An Implementation of the Bubble Sort Algorithm

```
public static void bubbleSort(int[] data)
{
   boolean notFinished;
   int endIndxUnsortedRegion = data.length-1, temp;
   do
   {
      notFinished = false ;
      for (int i = 0; i < endIndxUnsortedRegion; i++)
      {
         if (data[i] > data[i+1])
         {
            temp = data[i];
            data[i] = data[i+1];
            data[i+1] = temp;
            notFinished = true;
         }
      }
      endIndxUnsortedRegion--;
   }while(notFinished);
}
```

(a) Given the array *int list[]* $= \{15, 41, 12, 12, 41, 7, 5, 9\}$, trace the action of bubble sort showing the contents of the array after each pass. Use the broken pipe symbol | to mark the boundary between the sorted and the unsorted regions of the array after each swap.

(b) How many passes are required to sort the array?

(c) How many pairs of elements of the array are compared during the execution of the algorithm?

(d) How many swaps are performed during the execution of the algorithm?

C. Binary search is a divide-and-conquer algorithm. It works by repeatedly reducing the size of the array to be searched until a target is found or is determined not to be in the array. Consider the declaration of the array *list* and the code segment given below.

Listing 3: An Implementation of the Binary Search Algorithm

```
int[] list = {2, 3, 5, 7, 11, 13, 17, 19, 23,
              29, 31, 37, 41, 43, 47, 53, 59};
int target = 37;

public static int binarySearch(int[] list,int target)
{
   int mid, low = 0, high = list.length -1;
   while (low <= high)
   {
      mid = (low+high)/2;
      if (target == list[mid])
         return mid;
      if (target < list[mid])
         high = mid - 1;
      else
         low = mid + 1;
   }
   return -1-low;
}
```

(i) Trace the action of the binary search algorithm, including listing the values of *low*, *high*, and *mid*, indices of the array, and the element at the middle position, given the array *list* and the search *target* 37. (In the table, use as many columns as needed.) For *found*, write **Y** when the target is found and **N** otherwise.

low						
mid						
high						
list[mid]						
found						

(ii) How many elements of the array are accessed during the search?

(iii) List the elements in the order they were accessed.

(iv) What value will the algorithm return?

(v) How many comparisons are made during the search?

D. Define a public *class Ellipsoid* that describes an ellipsoid in a three-dimensional space. No documentation is required but use the names of variables given in this exercise and descriptive names for any additional variables that you declare.

Definition An **ellipsoid** is a surface that may be obtained from a sphere by deforming it by means of an affine transformation. See Figure 1.

The volume and surface area of an ellipsoid can be calculated using the formulas below, where a, b and c are the elliptical radii.

$$volume = \frac{4}{3}\pi abc$$
$$area \approx 4\pi \left(\frac{a^p b^p + a^p c^p + b^p c^p}{3}\right)^{1/p}, \quad p \approx 1.6075$$

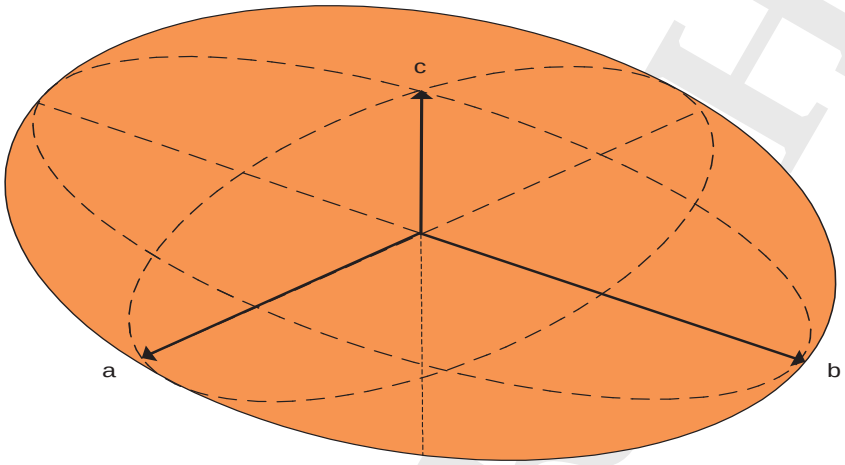

Figure 1: A Visual Depiction of an Ellipsoid

The class should consist of three floating-point instance variables *xRadius*, *yRadius* and *zRadius*, representing the radii along the x-axis, y-axis and z-axis, respectively. Here are additional requirements for the class:

(a) The class should consist of three constructors:
   i. a default constructor that creates an ellipsoid with all its radii equal to 1,
   ii. a copy constructor,
   iii. and a parameterized constructor that takes three floating-point parameters and creates an ellipsoid whose radii are set to those value or throws an *IllegalArgumentException* when any of the parameters is a negative number.

(b) The class should have three accessors *getXRadius*, *getYRadius* and *getZRadius*, that return the radii of the ellipsoid along the x-axis, y-axis and z-axis, respectively.

(c) The class should also have a mutator, *setEllipsoid*, that takes three floating-point parameters and modifies the radii of the ellipsoid using those values or throw an *IllegalArgumentException* when any of the parameters is a negative number.

(d) It should also have two methods, *area* and *volume*, that computes the area and volume of the ellipsoid. These method should use the Java library static constant *Math.PI* for π.

(e) Finally, the class should override the *toString* method so that it returns a string representation of the ellipsoid in the format:
"**Ellipsoid**[**x-Radius =** ..., **y-Radius =** ..., **z-Radius =** ...]".