# CSC 1350 Final Exam

Sections (3)/4

December 11, 2019

NAME: Kha Le

- This is a four-part exam.
- Use only Java™ 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.
- Answer exercise A.(a) on the exam sheet and all other exercises in your blue book.
- Write legibly and put the question number before each answer written in your blue book.
- Digital/Electronic devices are not allowed.
- Use the back of the exam sheets if you need scratch paper.
- Turn in the exam and your blue book before you leave.

DURATION: 120 Minutes

Table 1: Distribution of Points

| Exercise | WORTH | SCORE |
|---|---|---|
| A | $x_1 = 25$ | 25 |
| B | $x_2 = 25$ | 25 |
| C | $x_3 = 25$ | 25 |
| D | $x_4 = 25$ | 25 |
| Total | $\sum_{i=1}^{4} x_i$ | 100 |
| 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. Binary search is a decrease-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 implementation of the binary search algorithm shown in Listing 1 below.

Listing 1: An Implementation of the Binary Search Algorithm

```
int list[] = {1, 12, 14, 16, 17, 17, 18, 20,
              21, 24, 26, 27, 28, 29, 30};

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

(a) Trace the action of *binarySearch* by completing the chart in Table 2 given the call *binarySearch(list,26)*, where *list* is the array declaration above and 26 is the search key. [15 points]

Table 2: A Hand-trace of Binary Search

| low | high | low≤high | mid | list[mid] | searchKey==list[mid] | searchKey<list[mid] |
|---|---|---|---|---|---|---|
| 0 | 14 | true | 7 | 20 | false | false |
| 8 | 14 | true | 11 | 27 | false | true |
| 8 | 10 | true | 9 | 24 | false | false |
| 10 | 10 | true | 10 | 26 | true | |

(b) How many elements of the array are examined during the search? [2.5 points]

(c) List the elements in the order they are examined. [2.5 points]

(d) What value will the method return? [2.5 points]

(e) How many comparisons are made during the search? [2.5 points]

B. Consider implementation of the bubble sort algorithm shown in Listing 2 and answer the exercises that follow.

Listing 2: An Implementation of the Bubble Sort Algorithm

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

(a) Given the array *int list[] = {6, 4, 2, 1, 5, 9}*, trace the action of the bubble sort algorithm in Listing 2 showing the contents of the *list* 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 pass through the unsorted region of the array. [10 points]

(b) List all the pairs of elements of the array that are compared during each pass. Use labels such as *during pass 1:, during pass 2:,* etc. Enclose each pair of elements being compared in a pair of parentheses in the order in which they appear in the array at the time they are compared. Separate each pair by a comma. [5 points]

(c) How many swaps are performed during the execution of the method? [5 points]

(d) How many passes are required to sort the array? What is the total number of comparisons that are carried out when sorting the array? [5 points]

C. Give the output when the code segment in Listing 3 is executed. [25 points]

Listing 3: Code Segment

```
final int DIM = 3;
int magic[][] = new int[DIM][DIM];

int i, j;
for (i = 0; i < magic.length; i++)
{
  for (j = 0; j < magic.length; j++)
  {
    if (i == 0)
    {
      if (j < 2)
        magic[i][j] = 2 - i - j;
      else
        magic[i][j] = magic[i][j-1] + magic[i][j-2];
    }
    else if (j == 0)
      magic[i][j] = magic[i-1][DIM-1] + magic[i-1][DIM-2];
    else if (j == 1)
      magic[i][j] = magic[i][j-1] + magic[i-1][DIM-1];
    else
      magic[i][j] = magic[i][j-1] + magic[i][j-2];
  }
}
System.out.println(Arrays.deepToString(magic));
```

D. Implement a *public Cylinder* class that describes a solid as shown in Figure 1.

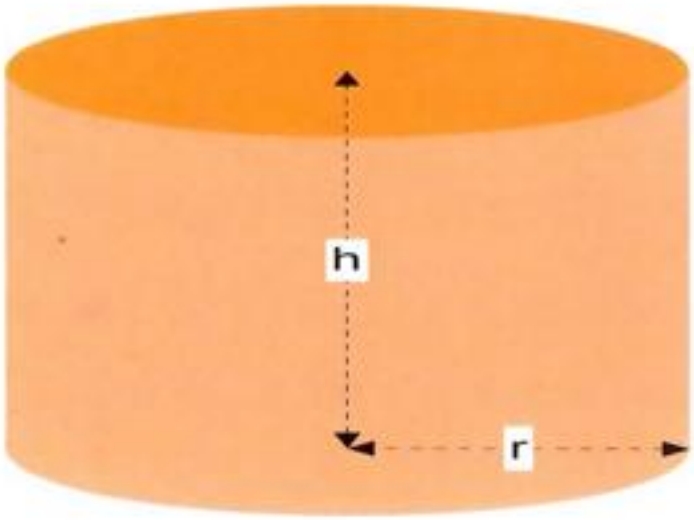


Figure 1: A Visual Depiction of a Cylinder

E. The *public Cylinder* class consists of two *private* instance variables, *height* and *radius*, both of which are *double* precision numeric values representing the height and radius of a cylinder. The class also consists of the following *public* methods.

- A default constructor that creates a cylinder whose height and radius are both 0.
- A parameterized constructor that has two parameters, $h$ and $r$, and creates a cylinder using these parameters when neither is negative. The parameters are both double precision numeric values and they represent the height and radius of the cylinder. This method should throw an *IllegalArgumentException* when either parameter is negative.
- An accessor method, **getHeight**, that returns the height of the cylinder.
- An accessor method, **getRadius**, that returns the radius of the cylinder.
- A void mutator method **setCylinder** that takes two parameters, $h$ and $r$, both of which are double precision numeric values, and uses them to modify the height and radius of the cylinder, respectively, when neither is negative. This method should throw an *IllegalArgumentException* when either parameter is negative.
- Override the *toString()* method so that it returns a string representation of the cylinder in the format "Cylinder[height = $\cdots$, radius = $\cdots$]", where the ellipses are replaced with the numeric values for the height and radius of the cylinder.

$$area = 2\pi r\,(r + h) \qquad \text{(eqn 1)}$$

Answer the following exercises:

(a) Provide the implementation for the *Cylinder* class but no documentation is required. [17.5 points]

(b) Write a declaration statement that creates an object of the *Cylinder* class called *can* whose height is 6 inches and whose radius is 4 inches. [2.5 points]

(c) Provide a statement that outputs the area of *can* in the format *area* = $\cdots$ *squared inches*, where the ellipsis represents the numeric value for the area *can*. Use the formula given in Equation eqn 1 along with the relevant method(s) to compute the area. Use the standard Java™ static *Math* class constant for $\pi$. [2.5 points]

(d) Write a statement that preserves the height and doubles the radius of *can* using the relevant accessor and mutator methods. Do not solely use numeric literals as arguments to the mutator method. [2.5 points]