

Total Score _____ Your work is to be done individually. The exam is worth 120 points (twenty points of extra credit are available throughout the exam) and it has eleven questions. Unless a problem directly instructs you differently, there are **no known errors** within this document.

1. [10 pts] Java provides several kinds of loop. Stylistically, when should one choose to use:

- a *for* loop?

If you know at the start of the loop how many iterations you need.

If you need to use the loop counter explicitly.

//your solution should include the explicit loop counter.

- a *do while* loop?

The number of execution is from 1 to infinity.

- a *while* loop?

The loop number of execution is from 0 to infinity. The end of the loop can be determined on the fly by some condition.

//your solution needs to describe how the loop is ended.

2. [10 pts] Assume the variable **list** holds a reference to an ArrayList containing Month elements. Show how to print all the elements in the list two different ways:

- using an (explicit) iterator

```
import java.util.Iterator;
Iterator<Month> iter = list.iterator();
while(iter.hasNext())
{
    System.out.println(iter.next().toString());
}
```

- using a *for-each* loop

```
for(Month month:list)
    System.out.println(month.toString());
```

Problems 3, 4, and 5 are based upon programming a method that receives a positive integer value and outputs an inverted pyramid. For example, if the value is **3** the output would be:

```
*****
***
*
```

Note: on the first row, there are *no* leading spaces.

3. [5 pts] Provide a good *analysis* for this problem.

- Print 2 less asterisks on each line
- Start with $(2n-1)$ asterisks.
- Input: a positive integer
- Output: an inverted pyramid

4. [5 pts] *Design* a good solution to the problem.

- The outer loop will execute n times to print the rows, note that a new line should be created in the outer loop
- The inner loop will print both spacing and asterisks. Using `print()` instead of `println()`

5. [10 pts] *Implement* that solution in Java.

```
public static void InvertPyramid(int row)
{
    if(row > 0)
    {
        int star = 2*row - 1 ;
        for(int i = 0; i < row; i++)
        {
            for(int j = 0; j < i; j++)
                System.out.print(" "); //print out the leading spaces
            for(int j = 0; j < star; j++)
                System.out.print("*");
            star -= 2;
            System.out.println(); // a newline must be created in the outer loop
        }
    }
}
```

6. [10 pts] Create a Java method that takes an array of **int** values and returns an array containing *only* the **non-zero** values from the input array.

```
public static int[] nonzero(int[] a) {
    int counter = 0;
    for (int i = 0; i < a.length; i++) {
        if (a[i] != 0) {
            counter++;    //first calculate the number of non-zero values in the input array
        }
    }

    int[] b = new int[counter];
    for (int m = 0; m < b.length; m++) {
        for (int n = m; n < a.length; n++) //note: the starting is m but not 0
        {
            if (a[n] >= 0) {
                b[m] = a[n];
                break;
            }
        }
    }

    return b;
}
```

7. [10 pts] Create a Java method that receives an **int** between 0 and 9 inclusive and returning a reference to a **String** whose value is the name of the input number. For example, 5 → “Five” and 7 → “Seven”. *Note: a maximum of half credit (5 pts) will be awarded if your code uses any conditional statements/logic.*

```
public String intToString( int n )
{
    String name[] = new String[] { "Zero", "One", "Two", "Three", "Four",
                                   "Five", "Six", "Seven", "Eight", "Nine" };
    return name[n];
}
```

8. [20 pts] The binary number system has exactly two digits: 0 and 1. The decimal number system has exactly 10 digits: 0 – 9. The hexadecimal number system has exactly 16 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. [FYI: computer scientists who are working with binary data often express integer values in hexadecimal because 1 hexadecimal digit is equivalent to 4 binary digits]. Create a method that prints, one per line, all the three-digit hexadecimal numbers in ascending order:

```
public void printHex1()
{
    for( int i = 0; i < 4096; i++ )    // loop to 2^12 (12 bits binary)
        System.out.println( convertToHex(i) );
}

public String convertToHex( int n )
{
    char [] hexChar = new char[]{'0','1','2','3','4','5','6',
                                   '7','8','9','A','B','C','D','E','F'};
    String out = "";

    for( int i = 0; i < 3; i++ ) {    // always create 3 digits String
        out = hexChar[n%16] + out;
        n /= 16;                      // remove the last hex digit
    }
    return out;
}
```

----- OR -----

```
public void printHex2()
{
    char [] hexChar = new char[]{'0','1','2','3','4','5','6',
                                   '7','8','9','A','B','C','D','E','F'};

    for( int i = 0; i < 16; i++ )    // leftmost digit
        for( int j = 0; j < 16; j++ )    // middle digit
            for( int k = 0; k < 16; k++ )    // rightmost digit
                System.out.println( "" + hexChar[i] + hexChar[j] + hexChar[k] );
}
```

Problems 9, 10, and 11 are based upon programming a method that takes an array of **int** values and returns that array *after* sorting its contents in **descending** order. For example, if the original array stored: [-3, 11, 0, -6, 6], then the sorted result would be: [11, 6, 0, -3, -6].

9. [10 pts] Provide a good *analysis* for this problem. Show all the steps involved in the process of sorting the sample array. Hint: each step *might* swap two of the values.

Here's one way to do it. Search the whole array for the largest value and swap it with the first slot. Next, search the array for the largest number again, but this time, skip the first slot. After you found the largest number, swap it with the second slot. Continue doing this until you reach the last slot.

$[-3, 11, 0, -6, 6] \rightarrow [11, -3, 0, -6, 6] \rightarrow [11, 6, 0, -6, -3] \rightarrow [11, 6, 0, -6, -3] \rightarrow [11, 6, 0, -3, -6]$

10. [10 pts] *Design* a good solution to the problem.

We need a loop that run from the first slot (index zero) to the second last slot (The last slot is automatically the lowest value).

Inside the loop, we first need to reset the initial maximum index to the starting point which is the current slot number. The inner loop is used for searching a maximum value. This loop will start at current index and end at the last slot (e.g. we search the array excluding slots that we already sorted). Finally we swap the largest value we have found with the current slot.

Return the array after the outer loop finish.

11. [20 pts] *Implement* that solution in Java.

```
public int[] sort( int []array )
{
    for( int i = 0; i < array.length - 1; i++ ) {

        int max = i;        // reset max index to the current index

        // searching for the largest value
        for( int j = i; j < array.length; j++ ) {
            if( array[max] < array[j] )    // if we find a larger value,
                max = j;                    // mark the index as a new max
        }

        // swap max value with the current position
        int temp = array[i];                // we need a temp variable
        array[i] = array[max];
        array[max] = temp;
    }

    return a;
}
```