

Review Exercises

Here is a list of review exercises. There are times when an exercise will be open ended, allowing you to pursue multiple aspects of the material. While encompassing some areas we have studied, you should further supplement your studying by reviewing: your notes, your project assignments, exercise questions given in lecture, exercise questions given in discussion sessions, and problems from the chapters in our textbook.

1. Create the *analysis*, *design*, and *implementation* of a method *diamond*. The method will print a diamond figure to the screen. The method will receive an int for the *length* of the diagonals to print. The method will not return anything. If five were passed to the method, the output would be:

```
    1
   2 2
  3  3
 4  4
 5
```

Analysis

We only need one integer to determine both height and width of the diamond figure. This method doesn't return anything.

Design

We can use nested loop to solve this problem by having an outer loop going one row at a time and an inner loop printing the content of each row. For the inner loop, we will distinguish between 3 row types; rows with only one number to be printed (the first and the last row), the upper half rows and the lower half rows, using row number. The positions of numbers on each row can be determined by the diamond height and current row number.

```
public void diamond(int number) {
    int row, col;

    System.out.println();

    for (row = 0; row < number; row++) {
        for (col = 0; col < number; col++) {
            if (row == 0 || row == number) {
                if (col == (number / 2)) {
                    System.out.print(row + 1);
                }
            }
            else {
                System.out.print(" ");
            }
        }
        else if (row <= number / 2) {
            if ((col == (number / 2) + row) || (col == (number / 2) - row)) {
                System.out.print(row + 1);
            }
            else {
                System.out.print(" ");
            }
        }
    }
}
```

```

    }
    else {
        if ((col == row - (number / 2)) ||
            (col == (number / 2) + (number - (row + 1)))) {
            System.out.print(row + 1);
        }
        else {
            System.out.print(" ");
        }
    }
}
System.out.println();
}
System.out.println();
}

```

2. Create the *analysis*, *design*, and *implementation* of a method *diamond*. The method will print a series of diamond figures to the screen. The method will receive an int for the *length* of the diagonals to print and an int for the *number* of diamonds to be printed. Similar to Question #1, if the user entered five and the three, the output is shown here. The method will not return anything.

```

    1          1          1
  2  2      2  2      2  2
 3     3 3      3 3      3
 4  4      4  4      4  4
 5     5          5

```

Analysis

We will need two integer inputs: the height of diamond figures and the number of diamonds to be printed. This method doesn't return anything.

Design

We can solve this problem by inserting one additional loop between the outer loop and the inner loop of question #1's code. This new loop will repeatedly print the same row *number* times.

```

public void diamond(int number, int diamonds) {
    int row, col, repeat;

    System.out.println();

    for (row = 0; row < number; row++) {
        for (repeat = 0; repeat < diamonds; repeat++) {
            for (col = 0; col < number; col++) {
                if (row == 0 || row == number) {
                    if (col == (number / 2)) {
                        System.out.print(row + 1);
                    }
                    else {
                        System.out.print(" ");
                    }
                }
                else if (row <= number / 2) {
                    if ((col == (number / 2) + row) || (col == (number / 2) - row)) {
                        System.out.print(row + 1);
                    }
                }
            }
        }
    }
}

```

```

        else {
            System.out.print(" ");
        }
    }
    else {
        if ((col == row - (number / 2)) ||
            (col == (number / 2) + (number - (row + 1)))) {
            System.out.print(row + 1);
        }
        else {
            System.out.print(" ");
        }
    }
}
}
}
System.out.println();
}
System.out.println();
}
}

```

3. Create the array *sentence*, type `char`, and ask the user for the number of characters to be entered. Then receive these characters, placing them within *sentence*.

Analysis

We do not need any input and output for this method. We also assume that users always enter correct inputs.

Design

We declare a new array using user input length. Then we can use a loop that goes over every element of that array and asks user for character input on each of them.

```

public static void initializeSentence()
{
    Scanner scanner = new Scanner(System.in);

    System.out.print("How many characters will you enter? ");
    int input = scanner.readInt();

    sentence = new char[input];

    for (int i = 0; i < sentence.length; i++) {
        System.out.print("char #" + (i + 1) + ": ");
        sentence[i] = scanner.next().charAt(0);
    }

    // echo characters entered
    printSentence();
}

public static void printSentence()
{
    System.out.print("The sentence entered is, \"");
    for (int i = 0; i < sentence.length; i++) {
        System.out.print(sentence[i]);
    }
    System.out.println("\"");
    System.out.println();
}
}

```

4. Create the *analysis*, *design*, and *implementation* of the method *copy*. The method will copy the contents of an array into a new array. The method will receive the array to be copied, the size of the new array, and return the newly created array.

Analysis

This method receives an array and an integer as input. This method will return an array which is a sub-array of the input array. We also assume *number* is a non-negative integer.

Design

First, we have to make sure that *number* is less than or equal to the array size. Then we will create a new array with that size. Finally, we use loop to copy the content of our input array over to the new array and return.

```
public char[] copy(char array[], int number)
{
    if (number > array.length) {
        number = array.length;
    }
    char temp[] = new char[number];

    for (int i = 0; i < number; i++) {
        temp[i] = array[i];
    }
    return temp;
}
```

5. Create the *analysis*, *design*, and *implementation* of the method *shrink*. The method will format the size of the array *sentence* to only be the size of the number of spaces actually used within the array. Suppose you modify your input questions to the user, allowing another character to be entered or the stopping of character input. Recall a position in a char array that has not be initialized is equal to "", that is two single quote with nothing in between. A *null* or empty character, opposed to a blank space.

Analysis

This method will not have any input because we only work with *sentence*. This method also returns the number of removed spaces. We assume that array *sentence* exists and it is visible inside our method.

Design

We need to know how many spaces to be removed before we can create a new array therefore the array need to be scanned and counts the number of empty characters. Next, we use another loop to copy over the original data without empty characters.

```
public int shrink() {
    char temp[], empty = (char) 0;
    int remove = 0;

    for (int i = 0; i < sentence.length; i++) {
        if (sentence[i] == empty) {
            remove++;
        }
    }
}
```

```

    }
    temp = new char[sentence.length - remove];

    for (int i = 0, j = 0; i < sentence.length; i++) {
        if (sentence[i] != empty) {
            temp[j] = sentence[i];
            j++;
        }
    }

    return remove;
}

```

6. Create the *analysis, design, and implementation* of the method *duplicates*. The method will receive a String and return a HashMap. The *keys* in the HashMap will be Strings. The *values* in the HashMap will be Integers. The HashMap will contain each substring and a count for the number of times the substring is found in the String given to the method. For example, given “two one two four one three one four one”, the HashMap will contain (“two”, 2), (“one”, 4), and so on.

Analysis

This method receives a string as an input and return HashMap of String and Integer as an output.

Design

We will use Scanner object to tokenize the String into word and add them into a newly declared HashMap object. If the word is already existed, we update the String count by first retrieving the count from the HashMap and then increment it before putting it back.

```

public HashMap<String, Integer> duplicates(String input)
{
    Scanner scanner = new Scanner(input);
    HashMap<String, Integer> out = new HashMap<String, Integer>();

    while( scanner.hasNext() ) {

        String s = scanner.next();

        if( out.containsKey(s) == false ) {
            out.put( s, 1 );
        }
        else {
            out.put( s, out.get(s) + 1 );
        }
    } // end while loop

    return out;
}

```

7. Create the *analysis, design, and implementation* of the method *commaSeparate*. The method will be receive an int value and modify the numeric representation to include comma’s. Use the standard mathematical definition for comma separating a numeric value, i.e. every third digit is separated from the other digits by a comma. If 7654321 were given to the method, “7,654,321” would be returned.

Analysis

We need to create a method that convert an integer value into a String that contains one or more commas. Our method will have one integer input and a String output. We assume that the input is a non-negative number.

Design

We will extract digits from the input integer one by one and append them to an output String. We also count how many digits we have extracted so far and add a comma character for every three characters extracted.

```
public String commaSeparate( int n )
{
    String out = "";    // output string

    for( int count = 0; n > 0; n /= 10 ) {

        // Extract the last digit from 'n' and
        // add it to the front of the string
        out = (n%10) + out;

        if( ++count == 3 ) {    // if we've extracted 3 digits,
            out = "," + out;    // add comma to the string
            count = 0;         // reset the counter
        }
    }
    return out;
}
```

8. Create the *analysis*, *design*, and *implementation* of the method *printJustifiedList*. The method will receive an ArrayList containing Integers. Each Integer will be printed to the screen justified to the right of the page. If the ArrayList contained: 14, -238, 3, 123456, and 1661 the output would be:

```
      14
     -238
        3
123456
     1661
```

Analysis

The method will receive an ArrayList of Integer object as an input. It will print the result on the screen thus returns nothing.

Design

If we want to print a right justified number we need to know length of the longest number (including sign prefix). The easiest way to determine the length is to convert Integer objects into Strings and use a length method. Then we can print all objects with preceding spaces calculated from the maximum length.

```
public void printJustifiedList( ArrayList<Integer> a )
{
    // find maximum length
    int maxlen = 0;
    for( int i = 0; i < a.size(); i++ ) {
        int len = a.get(i).toString().length();
        if( maxlen < len )
            maxlen = len;
    }

    // Printing loop
    for( int i = 0; i < a.size(); i++ ) {

        // convert Integer obj to String
        String s = a.get(i).toString();

        // print preceding spaces
        for( int j = 0; j < maxlen - s.length(); j++ ) {
            System.out.print(" ");
        }

        System.out.println( s );
    }
}
```