

The State University of New York, Korea

Computer Science

CSE 114

Handout 4: PS 3

March 23, 2019

This problem set is due at **11:55pm on Sunday, March 31, 2019, KST**. Don't go by the due date that you see on Blackboard because it is in EST. Go by the one given in this handout.

This problem set is due **11:50pm on Thursday, October 3**.

- Please read carefully and follow the directions exactly for each problem. Files and classes should be named exactly as directed in the problem (including capitalization!) as this will help with grading.
- You should create your programs using emacs.
- Your programs should be formatted in a way that's readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.
- This problem set assumes you have installed Java and emacs on your computer. Please see the course web for installation instructions.
- You must use the command-line interface to run your programs. That is, you must use the `javac` and `java` commands to do this problem set. Do not use Eclipse yet.
- **Be sure to include** a comment at the top of each file submitted that gives **your name** and **email address**.

Submit the following files on **Blackboard**. Multiple submissions are allowed before the due date. Please do **not** submit `.class` files or any that I did not ask for.

```
Digits.java
Series.java
Diamond.java
Slot.java
```

What Java Features to Use

For this assignment, you are not allowed to use more advanced features than what we have covered in Lectures 1 through 7 (March 18 lecture) [Chapters 1 through 6 of our textbook]. You may use for loops although it was covered in Lecture 8.

Partial vs. Complete Solutions

Please read what I said in PS 1 on this issue.

Naming Conventions In Java And Programming Style In General

Refer to the ones given in PS 1.

Problem 1

Before we begin. Starting with this problem set, problems get much more challenging than the previous ones. It is because we now start combining concepts that we have studied so far as we solve these problems.

Now, Problem 1. Implement a class named `Digits` in a file named `Digits.java`. Your program is to read a positive integer from keyboard and print the number of even digits in the number, the number of odd digits in the number, and the number of zeros in the number that you read in. As you design your program, implement the following functions in the class:

```
public static boolean isEven (int n);
public static int numEvens (int n);
public static int numOdds (int n);
public static int numZeros (int n);
```

`isEven` returns true if `n` is even; false otherwise. `numEvens` counts the number of even digits in `n` and returns the count. Similarly for the number of odd digits and zeros with the other two functions.

Your main function should read a positive integer from keyboard and use these four functions to produce the desired output. For example, if the number that your program reads in from keyboard is 1073740804, your program should generate an output that looks like the following:

```
1073740804 contains 3 evens, 4 odds, and 3 zeros.
```

Notes: As you design your program, you may feel that the whole thing could easily be done in one function, namely `main` without introducing any of the functions that are mentioned above. Well, that may be the case, but more importantly I would like you to learn how to use functions to manage the complexity of your program. Another point: How large is too large a number to read into an `int` variable? Well, how many bits are used to represent an `int` value in Java? Also note that there is a sign bit.

Hand in your `Digits.java`.

Problem 2

In this problem you will solve the problem that you solved in Problem 1 once more but differently.

Again your program is to read a positive integer from keyboard and print the number of even digits in the number, the number of odd digits in the number, and the number of zeros in the number that you read in. In this new design though you will implement the following function:

```
// numDigits will return the number of even digits in n if
// which is 2, the number of zero digits in n if which is 0,
// and the number of odd digits in n if which is 1.
//
public static int numDigits (int n, int which);
```

You will add this function to `Digits` from Problem 1. I can think of at least two different ways of implementing `numDigits` as follows: (1) Use the other three functions (i.e., `numEvens`, `numOdds`, and `numZeros`) from `numDigits`; or (2) Don't call any of the three functions (i.e., `numEvens`, `numOdds`, and `numZeros`) explicitly from `numDigits` although use the same logic used inside the bodies of those three functions in the body of `numDigits`. You may choose either one in your solution.

The main function from Problem 1 can now be extended to call `numDigits` as many times as you need to generate the desired output using the same number that you read in in Problem 1. Using the same input number 1073740804, your program should generate the same output that your program generated in Problem 1, namely, i.e.:

```
1073740804 contains 3 evens, 4 odds, and 3 zeros.
```

Hand in your `Digits.java` which now contain your solutions for both Problems 1 and 2.

Problem 3

Implement a class named `Series` in a file named `Series.java`. Implement the following function in the class:

```
public static double series (double x, int n);
```

where $-1.0 < x < 1.0$ and n as a positive integer is the number of terms to be added. This function computes the sum of specified number of terms in the following series.

$$1 - 2x + 3x^2 - 4x^3 + 5x^4 - \dots$$

For example,

```
series(0.5, 1) = 1
series(0.5, 2) = 0
series(0.5, 3) = 0.75
```

If you have to use any loop in this problem, you may only use `while` loops. This method (`series`) should contain no output statement. That is, output would be generated by a function other than `series`. You may assume that you only get valid actual arguments.

Here is a sample main that you may use. Include at least the three tests given above and add a few more of your own.

```
public static void main (String[] args) {
    System.out.print("series(0.5, 1) should be 1.0");
    System.out.println(" : " + series(0.5, 1));
}
```

Note: You should not try to simplify the series mathematically. We are not asking you to do math. You are simply using the formula to write a program to compute.

Hand in `Series.java`.

Problem 4

Implement a class named `Diamond` in a file named `Diamond.java`. In this class we will build a diamond. We will start with something very simple and incrementally build up to arrive at the ultimate result—incremental development will help us here. In each of the six parts in this problem, you are free to add additional functions beyond the ones I ask you to define if you think defining additional auxiliary functions would be helpful.


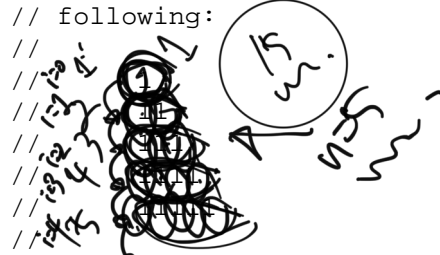
(1) Add a function with the following signature to `Diamond`:

```
// Prints a 1 in each of the n lines. So, if n is 5, it will
// print the following:
//
//      1
//      1
//      1
//      1
//      1
//
// without the leading blank spaces.
public static void printLines (int n);
```

Now add a call to `printLines` in your main to make sure that it works correctly.

(2) Add a function with the following signature to Diamond:

```
// Prints varying number of 1's in each of the n lines, increasing
// by one more 1 in each line. So, if n is 5, it will print the
// following:
//
// 1
// 11
// 111
// 1111
// 11111
// without the leading blank spaces.
public static void printRightHalf (int n);
```



Now add a call to `printRightHalf` in your main to make sure that it works correctly too.

(3) Add a function with the following signature to Diamond:

```
// Prints varying number of 1's and 2's in each of the n lines,
// increasing by one more 1's and 2's in each line generating a
// triangular shape as shown below. So, if n is 5, it will print
// the following:
//
// 21
// 2211
// 222111
// 22221111
// 2222211111
// without the leading blank spaces.
public static void printTop1 (int n);
```

Now add a call to `printTop1` in your main to make sure that it works correctly as well.

(4) Add a function with the following signature to Diamond:

```
// Prints varying number of 1's and 2's in each of the n lines,
// increasing by one more 1's and 2's in each line generating a
// triangular shape as shown below. So, if n is 5, it will print
// the following:
//
//      21
//     2211
//    222111
//   22221111
//  2222211111
// with the leading blank spaces this time.
public static void printTop2 (int n);
```

Now add a call to `printTop2` in your main to make sure that it works correctly as well.

(5) Add a function with the following signature to Diamond:

```
// Prints varying number of 1's and 2's in each of the n lines
```

```

// by one fewer 1's and 2's in each line generating a triangular
// shape as shown below. So, if n is 5, it will print the
// following:
//
// 2222211111
//  22221111
//   222111
//    2211
//     21
//
// with the leading blank spaces.
public static void printBottom (int n);

```

Now add a call to `printBottom` in your main to make sure that it works correctly as well.

(6) Add a function with the following signature to `Diamond`:

```

// Prints a diamond of size n. If n = 5, it would generate the
// following:
//
//     21
//    2211
//   222111
//  22221111
// 2222211111
// 2222211111
//  22221111
//   222111
//    2211
//     21
//
// with the leading blank spaces.
public static void printDiamond (int n);

```

Now add a call to `printDiamond` in your main to make sure that it works correctly as well.

(6) Add a function with the following signature to `Diamond`:

```

// Prints m diamonds of size n. If m = 3 and n = 5, it would
// generate the following:
//
//     21
//    2211
//   222111
//  22221111
// 2222211111
// 2222211111
//  22221111
//   222111
//    2211
//     21
//     21
//    2211
//   222111
//  22221111
// 2222211111
// 2222211111
//  22221111

```

```

//      222111
//      2211
//      21
//      21
//      2211
//      222111
//      22221111
//      2222211111
//      2222211111
//      22221111
//      222111
//      2211
//      21
//
// with the leading blank spaces.
public static void printDiamonds (int m, int n);

```

Now add a call to `printDiamonds` in your main to make sure that it works correctly as well.

Hand in your `Diamond.java`.

Problem 5

Implement a class named `Slot` in a file named `Slot.java`. In this problem your program will simulate a simple slot machine in which three numbers between 0 and 9 (inclusive) are randomly selected and printed side by side separated by a reasonable number of blank spaces. It will also print an outcome telling the user whether it was a *triple*, *double*, or *single*. Continue playing until the user chooses to stop. When the user chooses to stop, print out some statistics much like my solution does (see below). Try to introduce a function whenever a specific task can be isolated as a separate function that can be called by another function. Hand in `Slot.java`.

Here is a sample interaction (generated by my solution):

```

alee$ java Slot
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 4, 9, 1
    Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 2, 3, 4
    Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 1, 6, 1
    wow, double luck!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 7, 3, 2
    Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 3, 4, 4
    wow, double luck!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 0, 6, 4
    Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 2, 6, 8
    Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 6, 5, 4
    Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1

```

The numbers you drew are: 9, 5, 6
Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 0, 5, 8
Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 2, 2, 2
wow, triple luck!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 8, 8, 0
wow, double luck!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 5, 8, 0
Better luck next time!
Continue? (enter 1 to continue 2 to stop): 1
The numbers you drew are: 8, 7, 2
Better luck next time!
Continue? (enter 1 to continue 2 to stop): 2
Out of 14 hands, you had
1 triple jack pots (7%),
3 doubles (21%), and
10 singles (71%) today!!!
Come again...