Hands-on Experiment # 10 : Worksheet

Section_____ Date_____

No more than 3 students per one submission of this worksheet.

Student ID _____ Name_____

Student ID _____ Name_____

Student ID _____ Name_____

Part A: Getting Familiar with Writing Recursive Methods

1) Consider the following recursive definition.

$$f(n) = \begin{cases} f(n-1) + f(n-2) + f(n-3) & n = 3,4,5,\ldots \\ 1 & n = 0,1,2 \end{cases}$$

Find the value of *f(n)* for all values of *n* listed in the table below.

n	*f(n)*
0	1
1	1
2	1
3	3
4	5
5	9
6	17

2) Write a Java method called *computeF(int n)* which returns the value of *f(n)*. Assume that *n* is a non-negative integer. Test your method in a program in which the values of f(n) according to the following table are computed. Complete the table.

n	*f(n)*	*n*	*f(n)*
0	1	7	31
1	1	8	57
2	1	9	105
3	3	10	193
4	5	100	1.2707161788700277E26
5	9	200	3.7067466851909835E52
6	17	500	9.20080768385554E131

List all your source code below.

```
import java.util.Scanner;

public class Recursive {
   static double[] num = new double[501];
   static int called = 0;
```

```
  public static void main(String[] args) {
    Scanner kb = new Scanner(System.in);
    int n = kb.nextInt();
    System.out.println("f(" + n + ") :" + computeF(n));
    System.out.println("Called: " + called + " times");
  }

  public static double computeF(int n) {
    called++;
    if (n < 3)
      return 1;
    if (num[n] != 0)
      return num[n];
    num[n] = computeF(n - 1) + computeF(n - 2) + computeF(n - 3);
    return num[n];
  }
}
```

3) (Optional) Based on your code, how many times *computeF()* is called in order to compute *computeF(500)*? You may modify the signature of the method so that you have a way to track the number of times it is called.

The number of times is :

1495

Explain how you obtained the answer. Show all relevant code.

Use called variable to collect round of function called.

```
import java.util.Scanner;

public class Recursive {
  static double[] num = new double[501];
  static int called = 0;

  public static void main(String[] args) {
    Scanner kb = new Scanner(System.in);
    int n = kb.nextInt();
    System.out.println("f(" + n + ") :" + computeF(n));
```

```
    System.out.println("Called: " + called + " times");
  }

  public static double computeF(int n) {
    called++;
    if (n < 3)
      return 1;
    if (num[n] != 0)
      return num[n];
    num[n] = computeF(n - 1) + computeF(n - 2) + computeF(n - 3);
    return num[n];
  }
}
```

Part B: Thinking Recursively

1) Write a recursive method that checks whether the input *String* is a palindrome (http://en.wikipedia.org/wiki/Palindrome). Assume that the input *String* only contains English alphabets. You can design the method signature by yourself.

List your source code here.

```
import java.util.Scanner;

public class Palindrome {
  public static void main(String[] args) {
    Scanner kb = new Scanner(System.in);
    String str = kb.nextLine();
    System.out.println(check(str.toLowerCase()));
  }

  public static boolean check(String str) {
    if (str.length() < 2)
      return true;
    return (str.substring(0, 1).equals(str.substring(str.length() - 1)))
        && check(str.substring(1, str.length() - 1));
  }
}
```

Submit this worksheet (by only one member of the group) via http://www.myCourseVille.com (Assignments > Hands-on Experiment # 10) before noon of the day after your lecture.