Tutorial #4

SOFE 2710U Object Oriented Programming and Design

Problem1: Computing Powers

In this problem, you will implement a recursive method to compute the power of a number. The mathematical process of calculating a^n (where 'a' is the base and 'n' is the exponent) can be described as follows:

- If $n = 0,\ a^n$ is 1 (by definition)

- If $n > 0,\ a^n$ is $a * a^{n-1}$

You are provided with a file named Power.java, which contains a main program that reads in two integers, base and exp, and calls the power method to compute $base^{exp}$. However, the recursive code for the power method is missing.

Task: Please complete the missing code in the power method, <u>marked in red</u>, using the provided comments as guidance, so that it correctly performs the recursive calculation.

```
// ************************************************************
// Power.java
// Reads in two integers and uses a recursive power method
// to compute the first raised to the second power.
// ************************************************************
import java.util.Scanner;
public class Power
{
   public static void main(String[] args)
   {
      int base, exp;
      int answer;
      Scanner scan = new Scanner(System.in);
      System.out.print("Welcome to the power program! ");
```

```
        System.out.println("Please use integers only.");

        //get base
        System.out.print("Enter the base you would like raised to a power: ");
        base = scan.nextInt();
        //get exponent
        System.out.print("Enter the power you would like it raised to: ");
        exp = scan.nextInt();
        answer = power (base,exp);
        System.out.println(base + " raised to the " + exp + " is " + answer);

    // ----------------------------------------------
    // Computes and returns base^exp
    // ----------------------------------------------
    public static int power(int base, int exp)
    {
        int pow;
        //if the exponent is 0, set pow to 1
        //otherwise set pow to base*base^(exp-1)
        //return pow
    }
}
```

Problem2:

A **palindrome** is a string that reads the same forward and backward. In this problem, you are required to write a recursive function to check whether a string is a palindrome.

A string is considered a palindrome based on the following rules:

- A string with fewer than 2 letters is always a palindrome.
- A string with 2 or more letters is a palindrome if:
    - The first and last letters are the same.
    - The rest of the string (without the first and last letters) is also a palindrome.

Task: Write a program that prompts the user to enter a string. Your program should then print whether the string is a palindrome. You should implement a recursive method called palindrome that takes a string and returns true if the string is a palindrome, or false otherwise.

Helpful information about Java string methods:

- s.length() returns the number of characters in the string s.
- s.charAt(i) returns the character at position i in the string s. (Note: i is 0-based).
- s.substring(i, j) returns the substring starting at the i^{th} character and ending at the $j\text{-}1^{th}$ character. Both i and j are 0-based.

Example:

If s = "happy", then:
s.length() = 5
s.charAt(1) = 'a'
s.substring(2,4) = "pp"

Problem3: Efficient Computation of Fibonacci Numbers

The Fibonacci sequence is a series where each number is the sum of the two preceding ones. It can be defined as:

fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2) for n > 1

Task 1: Write a Recursive Fibonacci Method
You will create a method to compute the Fibonacci sequence using recursion. You are provided with a file named Fib.java, which contains a class skeleton. Your job is to fill in the fib1 method to compute the nth Fibonacci number using recursion.

****

Task 2: Test the Recursive Method
You are also given a file called TestFib.java, which will help you test the fib1 method. This file asks the user for an integer and calls the fib1 method to compute the corresponding Fibonacci number.

Test with small numbers, then larger ones. You'll notice the method becomes slow for larger numbers because it makes many repeated recursive calls.
To understand this, add a print statement in fib1 to display when a call is being made (e.g., "In fib(3)" if calculating fib(3)). Run the program again to see how often values are recalculated.
Notice how fib(5) needs fib(4) and fib(3), and fib(4) also needs fib(3). This repetition makes it inefficient.

Task 3: Make the Fibonacci Method More Efficient
To fix the inefficiency, you will modify the Fibonacci method to store the values as you calculate them, so they don't have to be recalculated repeatedly. Here's how to do it:

Add another method, fib2, which will calculate Fibonacci numbers iteratively (instead of using recursion).
In fib2, create an array to store the Fibonacci sequence up to the nth value.

Initialize the first two elements of the array as 0 and 1, then fill in the rest by summing the previous two values in the sequence.
Modify your TestFib file to call fib2 first and then compare it with the results of fib1.
With this new method, you will get the same result but with much faster computation times, especially for larger numbers.

```
          ************************************************************
// Fib.java
// A utility class that provide methods to compute elements of the
// Fibonacci sequence.
// ************************************************************

public class Fib
{
   // -------------------------------------------
   // Recursively computes fib(n)
   // -------------------------------------------
   public static int fib1(int n)
   {
      //Fill in code -- this should look very much like the
      //mathematical specification
   }
}

// ************************************************************
// TestFib.java
// A simple driver that uses the Fib class to compute the
// nth element of the Fibonacci sequence.
// ************************************************************

import java.util.Scanner;

public class TestFib
{
   public static void main(String[] args)
   {
      int n, fib;

      Scanner scan = new Scanner(System.in);
```

```
        System.out.print("Enter an integer: ");
        n = scan.nextInt();

        fib = Fib.fib1(n);
        System.out.println("Fib(" + n + ") is " + fib);
    }
}
```

Problem 4: Printing a String Backwards

Printing a string backwards can be done either iteratively or recursively. To do it recursively, think of the following specification:

If S contains any characters (i.e., is not the empty string)

- print the last character in S
- print S' backwards, where S' is S without its last character

File **Backwards.java** contains a program that prompts the user for a string, then calls method **printBackwards** to print the string backwards. Save this file to your directory and fill in the code for **printBackwards** using the recursive strategy outlined above.

```
// ************************************************************
//  Backwards.java
//
//  Uses a recursive method to print a string backwards.
// ************************************************************
import java.util.Scanner;

public class Backwards
{
    // --------------------------------------------------------
    //  Reads a string from the user and prints it backwards.
    // --------------------------------------------------------
    public static void main(String[] args)
    {
```

```
        String msg;
        Scanner scan = new Scanner(System.in);

        System.out.print("Enter a string: ");
        msg = scan.nextLine();

        System.out.println("\nThe string backwards: ");
        printBackwards(msg);
        System.out.println();
    }

    // -----------------------------------------------------
    //  Takes a string and recursively prints it backwards.
    // -----------------------------------------------------
    public static void printBackwards(String s)
    {
        // Fill in code
    }

}
```