Computing Powers

Computing a positive integer power of a number is easily seen as a recursive process. Consider a^n:

- If $n = 0$, a^n is 1 (by definition)
- If $n > 0$, a^n is $a * a^{n-1}$

File *Power.java* contains a main program that reads in integers *base* and *exp* and calls method *power* to compute $base^{exp}$. Fill in the code for *power* to make it a recursive method to do the power computation. The comments provide guidance.

```
// ****************************************************************
//   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
    }
}
```

Palindromes

A *palindrome* is a string that is the same forward and backward. In Chapter 5 you saw a program that uses a loop to determine whether a string is a palindrome. However, it is also easy to define a palindrome recursively as follows:

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

Write a program that prompts for and reads in a string, then prints a message saying whether it is a palindrome. Your main method should read the string and call a recursive (static) method *palindrome* that takes a string and returns true if the string is a palindrome, false otherwise. Recall that for a string *s* in Java,

- *s.length()* returns the number of charaters in *s*
- *s.charAt(i)* returns the i^{th} character of *s*, 0-based
- *s.substring(i,j)* returns the substring that starts with the i^{th} character of *s* and ends with the $j-1^{st}$ character of *s* (not the j^{th}), both 0-based.

So if *s*="happy", *s.length*=5, *s.charAt(1)*=a, and *s.substring(2,4)* = "pp".

Efficient Computation of Fibonacci Numbers

The *Fibonacci* sequence is a well-known mathematical sequence in which each term is the sum of the two previous terms. More specifically, if fib(n) is the nth term of the sequence, then the sequence can be defined as follows:

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

1. Because the Fibonacci sequence is defined recursively, it is natural to write a recursive method to determine the nth number in the sequence. File *Fib.java* contains the skeleton for a class containing a method to compute Fibonacci numbers. Save this file to your directory. Following the specification above, fill in the code for method *fib1* so that it recursively computes and returns the nth number in the sequence.

2. File *TestFib.java* contains a simple driver that asks the user for an integer and uses the *fib1* method to compute that element in the Fibonacci sequence. Save this file to your directory and use it to test your *fib1* method. First try small integers, then larger ones. You'll notice that the number doesn't have to get very big before the calculation takes a very long time. The problem is that the *fib1* method is making lots and lots of recursive calls. To see this, add a print statement at the beginning of your *fib1* method that indicates what call is being computed, e.g., "In fib 1(3)" if the parameter is 3. Now run TestFib again and enter 5—you should get a number of messages from your print statement. Examine these messages and figure out the sequence of calls that generated them. (This is easiest if you first draw the call tree on paper.) . Since fib(5) is fib(4) + fib(3), you should not be surprised to find calls to fib(4) and fib(3) in the printout. But why are there two calls to fib(3)? Because both fib(4) and fib(5) need fib(3), so they both compute it—very inefficient. Run the program again with a slightly larger number and again note the repetition in the calls.

3. The fundamental source of the inefficiency is not the fact that recursive calls are being made, but that values are being recomputed. One way around this is to compute the values from the beginning of the sequence instead of from the end, saving them in an array as you go. Although this could be done recursively, it is more natural to do it iteratively. Proceed as follows:

   a. Add a method *fib2* to your Fib class. Like *fib1*, *fib2* should be static and should take an integer and return an integer.

   b. Inside *fib2*, create an array of integers the size of the value passed in.

   c. Initialize the first two elements of the array to 0 and 1, corresponding to the first two elements of the Fibonacci sequence. Then loop through the integers up to the value passed in, computing each element of the array as the sum of the two previous elements. When the array is full, its last element is the element requested. Return this value.

   d. Modify your TestFib class so that it calls *fib2* (first) and prints the result, then calls *fib1* and prints that result. You should get the same answers, but very different computation times.

```java
// ******************************************************************
//   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
    }
}
```

```java
// ******************************************************************
//   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);
    }
}
```

Printing a String Backwards

Printing a string backwards can be done 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.print("\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

    }
}
```