6. Recursion :: Palindromic Strings

Before proceeding to the solution, let's check the problem characteristics we discussed earlier:

1. The size of the problem must be reducible to a smaller, basically-equivalent subproblem.

   Check. Rule 3 states that we will reduce the size of the string s being tested from $s.length()$ characters to a smaller string, t of size $s.length()$-2 characters. We will determine if t is a palindrome using the same operations.

2. The smaller, basically-equivalent subproblem must be simpler to solve than the larger problem.

   Check. I can assure you that it is easier to determine if a string of length 5 is a palindrome than a string of length 60,000,000.

3. The solution to the original problem requires repetition.

   Check. We will repeatedly test smaller and smaller strings. In fact, we could solve this problem by writing a while loop rather than using recursion (and that is true of all problems for which we devise a recursive solution; remember this: if the solution to a problem does not require a loop of some sort, then the problem cannot be solved recursively).

4. There must be a base case at which point the size of the problem cannot be further reduced.

   Check. There is a reason I defined the empty string to be a palindrome.

5. The base case will generally have an easily-obtainable solution.

   Check. See 4.

6. Recursion :: Palindromic Strings (continued)

6. The solution to the smaller subproblem must be returned and used in solving the larger problem.

   Check. The solution to our smaller subproblem will be true (the smaller string is a palindrome) or false (it is not). Consequently, the solution to our larger problem is the solution to our smaller problem.

7. The solution to the original problem results from solving all of the subproblems.

   Check. This is implied by 6.

6. Recursion :: *Palindrome* Example Program

```
//********************************************************************************
// CLASS: Palindrome (Palindrome.java)
//********************************************************************************
import java.util.Scanner;

public class Palindrome {
  public static void main(String[] args) { new Palindrome().run(); }

  public void run() {
    Scanner in = new Scanner(System.in);
    System.out.print("Enter a string? ");
    String s = in.next();
    if (isPalindromic(s)) {
      System.out.println(s + " is a palindrome.");
    } else {
      System.out.println(s + " is not a palindrome.");
    }
  }

  public boolean isPalindromic(String pString) {
    // The base case is when the length of pString is less than or equal to 1.
    // In this case, pString is a palindrome, so we return true. This implements
    // Rule 1.
    if (pString.length() <= 1) {
      return true;
```

6. Recursion :: *Palindrome* Example Program (continued)

```
    } else {
      // Check Rule 2. If the first and last characters are not the same then
      // pString is not a palindrome so we return false.
      if (pString.charAt(0) != pString.charAt(pString.length() - 1)) {
        return false;
      } else {
        // Check Rule 3, the recursive rule. We got here because the first and
        // last characters are the same. We now create a new string t which
        // contains the substring of pString starting at index 1 up to and
        // including pString.length() - 2. The String.substring() method can
        // be used to extract a substring from a string. However, the method is
        // a bit peculiar: s.substring(i, j) would return a substring of s
        // starting at index i up to and including the character at j-1, i.e.,
        // it does not return the character at index j.
        String t = pString.substring(1, pString.length()-1);
        // By Rule 3, if t is a palindrome then so is pString, so if
        // isPalindromic(t) returns true, we return true. Otherwise, if
        // isPalindromic(t) returns false, we return false. To be more concise,
        // we can say that at this point we simply need to return what
        // isPalindromic(t) returns.
        return isPalindromic(t);
      }
    }
  }
}
```