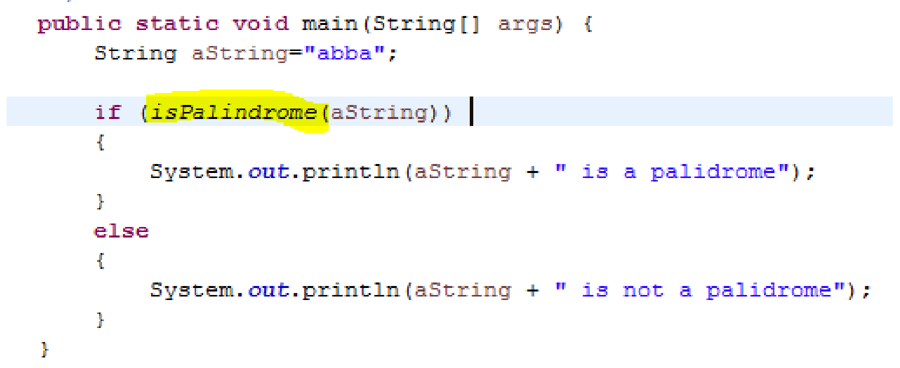
Recursion (continued)

What is a recursive helper?

- Sometimes, a task can be solved by handing it off to a recursive helper method.
- Sometimes it is easier to find a recursive solution if you make a slight change to the original problem.
- Consider the palindrome test of previous slide.
  - It is a bit inefficient to construct new Sentence objects in every step

We look at the following main program, related to palindrome



```java
public static void main(String[] args) {
    String aString="abba";

    if (isPalindrome(aString))
    {
        System.out.println(aString + " is a palidrome");
    }
    else
    {
        System.out.println(aString + " is not a palidrome");
    }
}
```

We create a recursive helper method to assist the original method of isPalindrome, using overloading.

```
/**
 * Verify if a string is a palindrome
 * @param s: the string that we examine
 * @return true if the string is a palindrome
 */
public static boolean isPalindrome(String s) {
    return isPalindrome(s,0, s.length()-1);
}

/**
 * this is the recursive helper for the above function
 * @param s the string that we verify as a palindrome
 * @param i the first character of the string or substring
 * @param f the last character of the string or substring
 * @return true if the string is a palindrome
 */
public static boolean isPalindrome(String s, int i, int f)  {
    if (i < f) {
        if (s.charAt(i) == s.charAt(f)) {
         return isPalindrome(s, i+1, f-1);
        } else {
         return false;
        }
    } else {
        return true;
    }
}
```

What is mutual recursion?

In a mutual recursion, a set of cooperating methods calls each other repeatedly. Here is an example of mutually recursive method

```
public static void main(String[] args) {
    int i = 455;

    if (isOdd(i))
        System.out.println(i + " is an odd number");
    System.out.println("It is an even number");

}
```

The implement of isOdd and isEven is mutually recursive

```
static boolean isEven( int number )
{
    if( number == 0 )
        return true;
    else
        return isOdd(Math.abs(number)-1);
}

static boolean isOdd( int number )
{
    if( number == 0 )
        return false;
    else
        return isEven(Math.abs(number)-1);
}
```

This is a simple example to show mutually recursive methods. Practical mutual recursion is used in parsing math expression.

The above example also shows that not all recursive should be use. The two methods can be written as follows

```
static boolean isEven( int number )
{
    return ((number % 2) ==0);
}

static boolean isOdd( int number )
{
    return !((number % 2) ==0);
}
```

Is recursive efficient?

- In most cases, the iterative and recursive approaches have comparable efficiency.
- Occasionally, a recursive solution runs much slower than its iterative counterpart.
- In most cases, the recursive solution is only slightly slower.
- The iterative `isPalindrome` performs only slightly better than recursive solution.
    - Each recursive method call takes a certain amount of processor time

One example of inefficient recursive algorithm is Fibonacci number.

```
import java.util.Scanner;

/**
  This program prints trace messages that show how often the
  recursive method for computing Fibonacci numbers calls itself.
*/
public class RecursiveFibTracer
{
  public static void main(String[] args)
   {
     Scanner in = new Scanner(System.in);
     System.out.print("Enter n: ");
     int n = in.nextInt();

     long f = fib(n);

     System.out.println("fib(" + n + ") = " + f);
   }

  /**
    Computes a Fibonacci number.
    @param n an integer
    @return the nth Fibonacci number
  */
  public static long fib(int n)
  {
    System.out.println("Entering fib: n = " + n);
    long f;
    if (n <= 2) { f = 1; }
    else { f = fib(n - 1) + fib(n - 2); }
    System.out.println("Exiting fib: n = " + n
        + " return value = " + f);
    return f;
  }
}
```

After put in tracing code, you can draw an execution tree. You will see that the inefficiency reveals at the fact that fib(3) are execute multiple times when fib(6) is called. It will get worse for fib(n) for n > 6.

Call Tree for Computing fib(6)

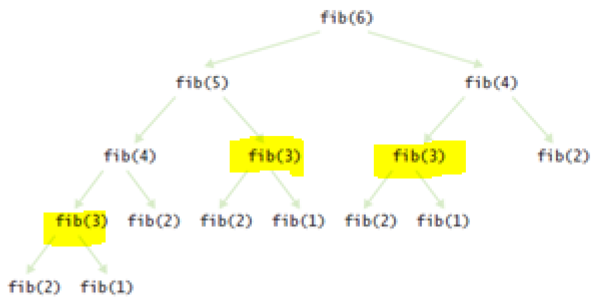


Figure 2 Call Pattern of the Recursive `fib` Method

Fibonacci method is more efficient when written in non-recursive approach. Please refer to your book at page