UCL DEPARTMENT OF ELECTRONIC & ELECTRICAL ENGINEERING

ELEC0021 - PROGRAMMING II
OBJECT-ORIENTED PROGRAMMING

RECURSION

Prof. George Pavlou
Communication and Information Systems Group
Dept. of Electronic and Electrical Engineering
http://www.ee.ucl.ac.uk/~gpavlou/
g.pavlou@ucl.ac.uk

Recursion

- In most programs object methods call each other in a hierarchical manner
- There is a category of programs though in which a method calls itself in a recursive manner
- Most tasks that can be done through recursion can also be done through iteration
  - For a category of problems though an iterative solution can be complex and difficult to engineer while a recursive solution can be concise and elegant
- Recursive solutions are generally slower in terms of performance because of the incurred penalty of successive method calls

Recursion Example: Factorials

- The factorial of a positive integer n denoted *n!* is defined as n(n-1)…1, with 1!=1 and 0!=1
- This can be easily calculated in an iterative manner through a for loop
- But it can be also observed that *n!=n*(n-1)!* and this can result in a recursive approach to calculate the factorial
- The class FactorialCalculator provides methods to calculate a number's factorial in both iterative and recursive fashion
  - Note that both methods are static as the methods of this class are used without any class object

Factorial Calculator

```
public class FactorialCalculator {
    public static long factorialIteration (int number) {
        if (number < 0)
            return -1; // method call failed
        long factorial = 1;
        for (int i = 1; i <= number; i++)
            factorial = factorial * i;
        return factorial;
    }
    public static long factorialRecursion (int number) {
        if (number < 0)
            return -1; //method call failed
        if (number <= 1)  // 0 or 1
            return 1;
        // note: no else required here because of the return above
        return number * factorialRecursion(number-1);
    }
} // end class FactorialCalculator
```

Recursion Example: Fibonacci Series

- The Fibonacci series starts with 0 and 1 and each subsequent number is the sum of the previous two
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, …
- The Fibonacci series can be defined recursively as follows:
  - fibonacci(0) = 0, fibonacci(1)=1
  - fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
- We have also added a static instance variable to keep the record the number of times the fibonacci method is called
  - Relevant numbers can be very big: for 30 the number is 2692537, for 31 it is 4356617, for 32 it is 7049155, etc.
  - You may write a program to calculate the Fibonacci number in an iterative manner and time the difference

Fibonacci Calculator

```
public class FibonacciCalculator {

    static private int fibonacciCalls;

    public static int   getFibonacciCalls ()    { return fibonacciCalls; }
    public static void resetFibonacciCalls () { fibonacciCalls = 0; }

    public static long fibonacciRecursion (int number) {
        if (number < 0)
            return -1; // method call failed

        fibonacciCalls++;
        if (number <= 1)  // 0 or 1
            return number;
        return fibonacciRecursion(number-1) + fibonacciRecursion(number-2);
    }

} // end class FibonacciCalculator
```

Timing a Program Segment

- The static currentTimeMillis method of class System returns the current time in msecs since 1 January 1970 GMT
  - static long currentTimeMillis()
- Taking the time before and after a particular program segment and calculating the difference can give the computation time a particular task takes
- We show an Fibonacci calculator test program that also shows the number of recursive calls needed and the time taken to calculate a Fibonacci number
  - You may comment out the fibonacciCalls++ line in the recursive fibonacci method in order to see the impact it has on the calculation time – you will be surprised

Fibonacci Calculator Test

```
public class FibonacciCalculatorTest {
    public static void main (String[] args) {
        if (args.length != 1) {
            System.err.println("usage: FibonacciCalculatorTest number");
            System.exit(1);
        }
        // here we need exception handling but we omit it for brevity
        int number = Integer.parseInt(args[0]);
        if (number < 0) {
            System.err.println("usage: number should be non-negative");
            System.exit(1);
        }
        long time0 = System.currentTimeMillis();
        long fibonacci = FibonacciCalculator.fibonacciRecursion(number);
        long time1 = System.currentTimeMillis();

        System.out.println("fibonacci of " + number + " is " + fibonacci);
        System.out.println("the fibonacci method was called recursively " +
                                        FibonacciCalculator.getFibonacciCalls() + " times");
        System.out.println("and the time it took to calculate the number was " +
                                        (time1-time0) + " msecs");
    }
} // end class FibonacciCalculatorTest
```

Fibonacci Iteration Method

```
// continuing FibonacciCalculator

    public static long fibonacciIteration (int number) {
        if (number < 0)
            return -1; // method call failed
        if (number <= 1) // 0 or 1
            return number;

        long fib = 1;           // initially fib(2)=1
        long prevFib = 1;  // initially fib(1)=1
        for (int i = 2; i < number; i++) {
            long tmp = fib;       // storing temporarily last/previous value
            fib = fib + prevFib; // calculating new value
            prevFib = tmp;      // “advancing” previous value
        }
        return fib;
    }

} // end class FibonacciCalculator
```