Recursion

and Java

Recursion is an extremely powerful problem-solving technique. Problems that at first appear difficult often have simple recursive solutions. Recursion breaks a problems into several smaller problems of *exactly the same type* as the original problem.

The typical example is the factorial of n, $n!$*. Although you could iteratively calculate $1 \bullet 2 \bullet \ldots \bullet n$, you could also claim that $n! = n \bullet (n-1)!$. Therefore, if you knew $(n-1)!$, you could easily calculate $n!$.

Java supports recursion by letting the programmer make an activation call inside a method on the method itself. However, be aware that programming recursively is not always more efficient or more elegant, the two main reasons why recursivity is an attracting programming technique, then an iterative approach.

*i.e.: $n! = n \bullet (n-1) \bullet \ldots \bullet 2 \bullet 1$

Example :

```
public class iter_fact
{
    public static void main(String args[])
        {
            long n = Long.parseLong(args[0]), result = 1;

            if (n == 0)
                System.out.println(1);
            else
            {
                for (long i = 1; i <= n; i++)
                    result *= i;
                System.out.println(result);
            }
        }
}
```

In which the program starts by reading an argument *n* and initiates a *result* variable to 1. It then multiples this variable *result* by all the values between 1 and n.

Example :

```
public class recur_fact
{

    public static long fact(long n)
        {
            if (n == 0)
                return 1;
            else return n * fact(n-1);
        }

    public static void main(String args[])
        {
            long n = Long.parseLong(args[0]);

            System.out.println(fact(n));
        }
}
```

We have now defined a recursive method called *fact* that takes as parameter the value whose factorial we are looking for. The recursive call is in the `return` statement where we multiply n by $fact(n-1)$.Note that we have included a *base case* to stop the recursivity when n is equal to 0.

Cautions:

- A recursive algorithm must have a base case, whose solution you know directly without making any recursive calls. Without a base case, a recursive function will generate an infinite sequence of calls. Note that you may have more then one base case.

- A recursive solution must involve one or more smaller problems that are each closer to a base case than is the original problem. You must be sure that these smaller problems eventually reach the base case. Failure to do so could result in an algorithm that does not terminate.

- You must be sure that the solutions to the smaller problems may be used to find a solution to the original problem.

- A recursion that recomputes certain values frequently can be quite inefficient.

The Fibonacci sequence:

(also known as the multiplying rabbits)

> Suppose that every month a breeding pair of rabbits produce a pair of offsprings. The offsprings will in their turn start breeding two months later, and so on.
>
> Thus if you buy a pair of baby rabbits in month 1, you will still have just one pair in month 2. In month 3 they will start breeding, so you now have two pairs; in month 4 they will produce a second pair of offsprings, so you now have three pairs; in month 5 both they and their first pair of offsprings will produce baby rabbits, so you now have five pairs; and so on.

If no rabbits ever die, this can be written as :

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

Although this could be programmed in an iterative manner, programming it recursively is much more obvious :

```
public class fibo
{
    public static long  fib(long n)
        {
            if (n<2)
                return n;
            else return fib(n-1) + fib(n-2);
        }

    public static void main(String args[])
        {
            long n = Long.parseLong(args[0]);

            System.out.println(fib(n));
        }
}
```

Note that De Moivre proved the following formula :

$$f_n = \frac{1}{\sqrt{5}} \left(\phi^n - (-\phi)^{-n} \right)$$

where ϕ is named the *golden ratio* and is worth $\frac{(1+\sqrt{5})}{2}$.
This function can easily be precisely estimated.