

Rekursif

Rekursif

- Proses yang memanggil dirinya sendiri.
- Merupakan suatu fungsi atau prosedur
- Terdapat suatu kondisi untuk berhenti.

Faktorial

- Konsep Faktorial
 $n! = n(n-1)(n-2)\dots 1$
- Dapat diselesaikan dengan
 - Cara Biasa
 - Rekursif

Iterative factorial:

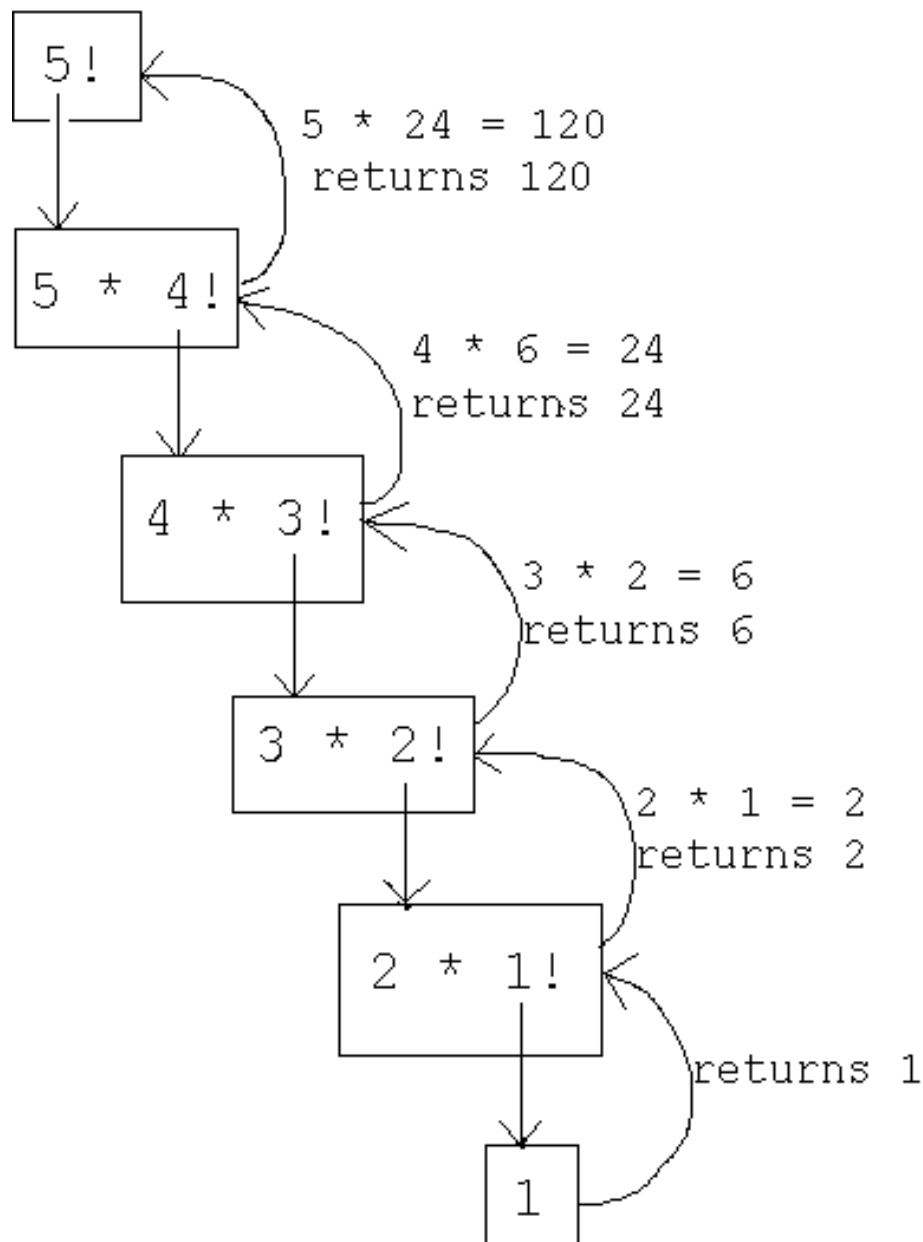
$$n! = \prod_{k=1}^n k$$

Recursive factorial:

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0. \end{cases}$$

Final value = 120

Faktorial



Faktorial : Cara Biasa

```
int Faktorial(int n)
{
    if (n<0) return -1 ;
    else if (n>1)
    {
        S = 1 ;
        for(i=2 ;i<=n;i++)    S = S * n ;
        return S ;
    }
    else return 1 ;
}
```

Faktorial dengan Rekursif

```
int Faktorial(int n)
{
    if (n<0) return -1;
    else if (n>1) return (n*Faktorial(n-1))
        else return 1 ;
}
```

Deret Fibonacci

- ❑ Leonardo Fibonacci berasal dari Italia 1170-1250
- ❑ Deret Fibonacci f_1, f_2, \dots didefinisikan secara rekursif sebagai berikut :

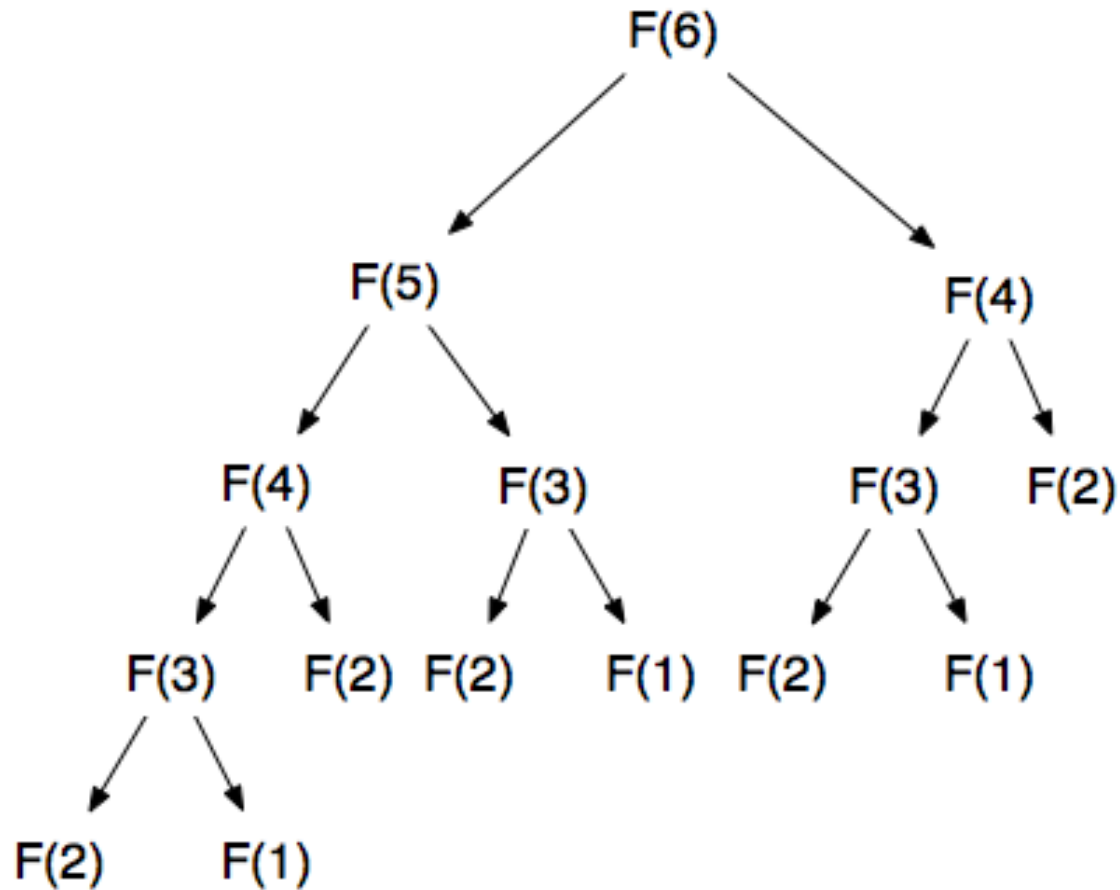
$$f_1 = 1$$

$$f_2 = 2$$

$$f_n = f_{n-1} + f_{n-2} \quad \text{for } n \geq 3$$

- ❑ Deret: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, ...

Deret Fibonacci



Deret Fibonacci

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

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 1; i <= N; i++)
            System.out.println(i + ": " + fib(i));
    }
}
```

Tower Hanoi

- Tower Hanoi adalah permainan puzzle dengan tiga tiang dan sejumlah disk/cakram yang tersusun pada tiang.
- Disk memiliki ukuran yang bervariasi. Disk-disk tertumpuk rapi berurutan berdasarkan ukurannya dalam salah satu tiang, disk terkecil diletakkan teratas, sehingga membentuk kerucut.
- Tujuan adalah untuk memasukkan semua disk dari tiang awal ke tiang tujuan dengan menggunakan tiang bantuan dengan aturan :
 - Hanya satu disk dapat dipindahkan pada suatu waktu
 - Sebuah disk tidak dapat ditempatkan di atas disk yang lebih kecil

The Towers of Hanoi puzzle



FIGURE 7.6 The Towers of Hanoi puzzle

A solution to the three-disk Towers of Hanoi puzzle

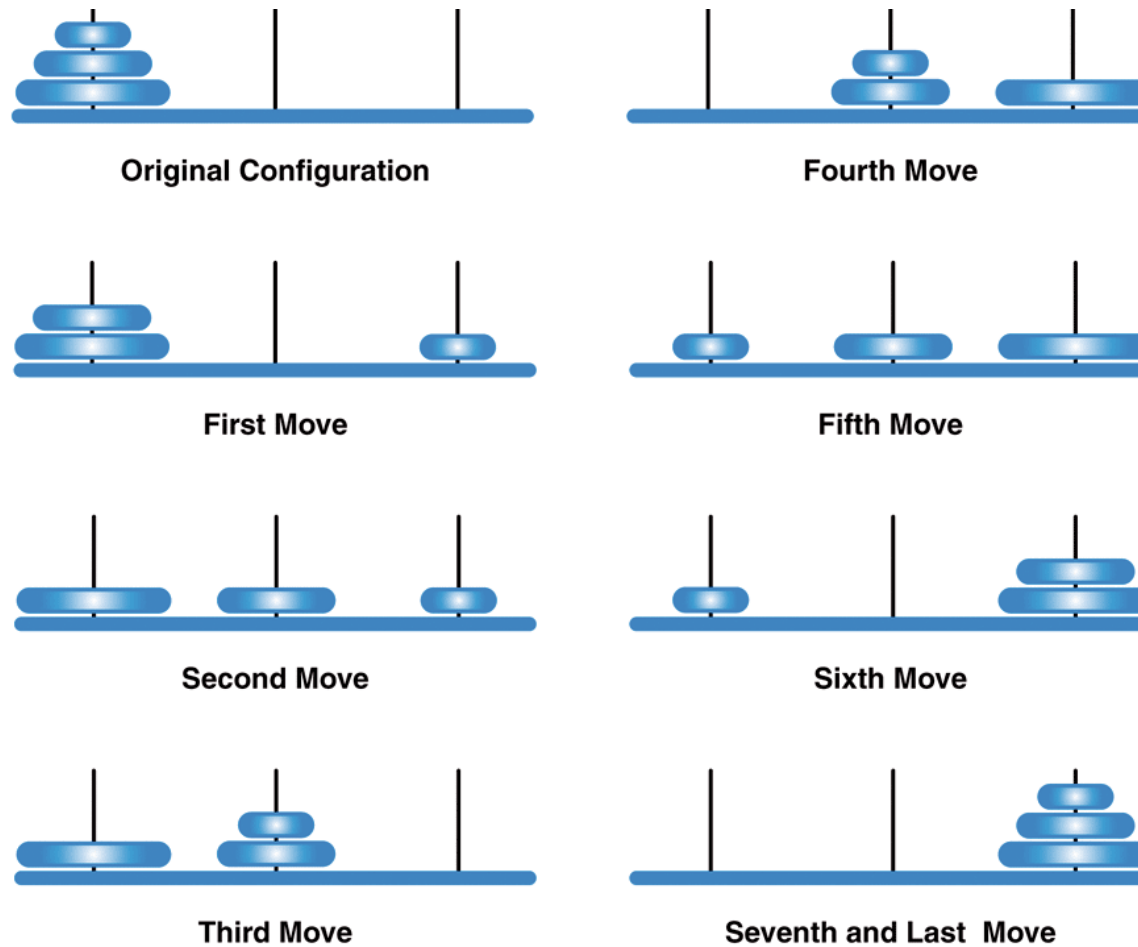


FIGURE 7.7 A solution to the three-disk Towers of Hanoi puzzle

Towers of Hanoi

- Untuk memindahkan N disk dari tiang asal ke tiang tujuan :
 - Pindahkan $N-1$ disk yang paling atas dari tiang asal ke tiang bantuan.
 - Pindahkan 1 disk terbesar dari tiang asal ke tiang tujuan
 - Pindahkan $N-2$ disk dari tiang bantuan ke tiang tujuan

Towers of Hanoi

- Jumlah perpindahkan disk bertambah secara exponential dengan bertambahnya jumlah disk
- Solusi secara rekursif lebih sederhana dibandingkan dengan solusi iteratif

UML description of the SolveTowers and TowersofHanoi classes

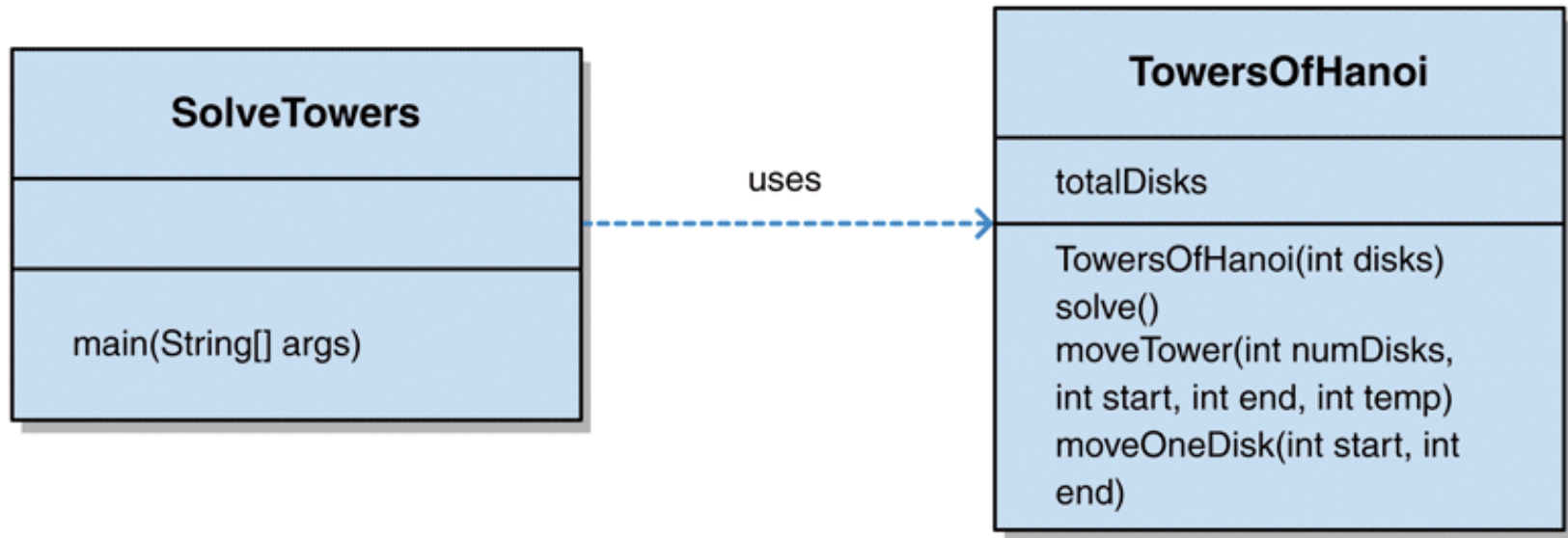


FIGURE 7.8 UML description of the `SolveTowers` and `TowersOfHanoi` classes

The Solve Towers class

```
/**
 * SolveTowers demonstrates recursion.
 *
 * @author Dr. Lewis
 * @author Dr. Chase
 * @version 1.0, 8/18/08
 */

public class SolveTowers
{
    /**
     * Creates a TowersOfHanoi puzzle and solves it.
     */
    public static void main (String[] args)
    {
        TowersOfHanoi towers = new TowersOfHanoi (4);
        towers.solve();
    }
}
```


The Towers of Hanoi class

```
/**
 * TowersOfHanoi represents the classic Towers of Hanoi puzzle.
 *
 * @author Dr. Lewis
 * @author Dr. Chase
 * @version 1.0, 8/18/08
 */

public class TowersOfHanoi
{
    private int totalDisks;

    /**
     * Sets up the puzzle with the specified number of disks.
     *
     * @param disks the number of disks to start the towers puzzle with
     */
    public TowersOfHanoi (int disks)
    {
        totalDisks = disks;
    }
}
```

The Towers of Hanoi class (continued)

```
/**
 * Performs the initial call to moveTower to solve the puzzle.
 * Moves the disks from tower 1 to tower 3 using tower 2.
 */
public void solve ()
{
    moveTower (totalDisks, 1, 3, 2);
}

/**
 * Moves the specified number of disks from one tower to another
 * by moving a subtower of n-1 disks out of the way, moving one
 * disk, then moving the subtower back. Base case of 1 disk.
 *
 * @param numDisks the number of disks to move
 * @param start    the starting tower
 * @param end      the ending tower
 * @param temp     the temporary tower
 */
```

The Towers of Hanoi class (continued)

```
private void moveTower (int numDisks, int start, int end, int temp)
{
    if (numDisks == 1)
        moveOneDisk (start, end);
    else
    {
        moveTower (numDisks-1, start, temp, end);
        moveOneDisk (start, end);
        moveTower (numDisks-1, temp, end, start);
    }
}

/**
 * Prints instructions to move one disk from the specified start
 * tower to the specified end tower.
 *
 * @param start the starting tower
 * @param end the ending tower
 */
private void moveOneDisk (int start, int end)
{
    System.out.println ("Move one disk from " + start + " to " +
        end);
}
}
```

Multibase Representations

- Decimal is only one representation for numbers. Other bases include 2 (binary), 8 (octal), and 16 (hexadecimal).
 - Hexadecimal uses the digits 0-9 and a=10, b=11, c=12, d=13, e=14, f=16.

$$\begin{aligned} 95 &= 1011111_2 & // & 95 &= 1(2^6)+0(2^5)+0(2^4)+0(2^3)+1(2^2)+1(2^1)+1(2^0) \\ & & & &= 1(64)+0(32)+1(16)+1(8)+1(4)+1(2)+1 \\ 95 &= 340_5 & // & 95 &= 3(5^2) + 4(5^1) + 0(5^0) \\ & & & &= 3(25) + 4(5) + 0 \\ 95 &= 137_8 & // & 95 &= 1(8^2) + 3(8^1) + 7(8^0) \\ & & & &= 1(64) + 3(8) + 7 \\ 748 &= 2ec_{16} & // & 748 &= 2(16^2) + 14(16^1) + 12(16^0) \\ & & & &= 2(256) + 14(16) + 12 \\ & & & &= 512 + 224 + 12 \end{aligned}$$

Multibase Representations (continued)

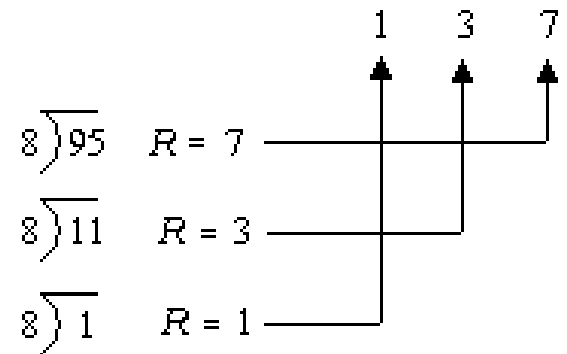
- An integer $n > 0$ can be represented in different bases using repeated division.
 - Generate the digits of n from right to left using operators '%' and '/'. The remainder is the next digit and the quotient identifies the remaining digits.

Multibase Representations (continued)

$$\begin{array}{r} \text{Step 1: } 8 \overline{)95} \\ \underline{88} \\ 7 \end{array}$$

$$\begin{array}{r} \text{Step 2: } 8 \overline{)11} \\ \underline{8} \\ 3 \end{array}$$

$$\begin{array}{r} \text{Step 3: } 8 \overline{)1} \\ \underline{0} \\ 1 \end{array}$$



```
85 % 8 = 7      // remaining digits: 85/8 = 11
11 % 8 = 3      // remaining digits: 11/8 = 1
 1 % 8 = 1      // remaining digits: 1/8 = 0
 0              // stopping condition
```

Multibase Representations (continued)

- Convert n to base b by converting the smaller number n/b to base b (recursive step) and adding the digit $n\%b$.

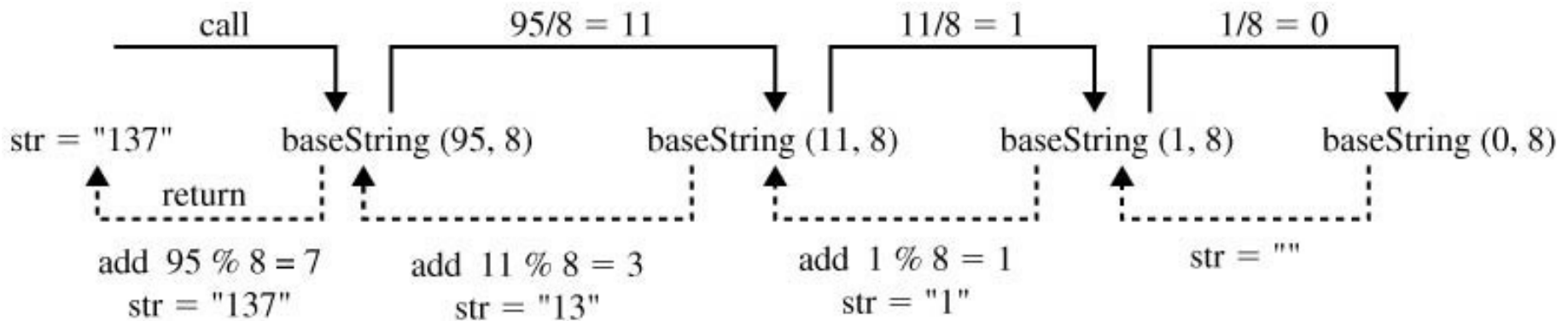
Multibase Representations (continued)

```
// returns string representation
// of n as a base b number
public static String baseString(int n, int b)
{
    String str = "", digitChar = "0123456789abcdef";

    // if n is 0, return empty string
    if (n == 0)
        return "";
    else
    {
        // get string for digits in n/b
        str = baseString(n/b, b);    // recursive step

        // return str with next digit appended
        return str + digitChar.charAt(n % b);
    }
}
```


Multibase Representations (concluded)



Series of recursive method calls for `baseString(95,8)`.

Rekursif Tail

- Jika hasil akhir yang akan dieksekusi berada dalam tubuh fungsi
- Tidak memiliki aktivitas selama fase balik.

What is Tail Recursion?

- Metode rekursif
 - Tail recursive
 - Nontail recursive
- Method tail recursive memiliki pemanggilan rekursif di akhir method.
- Recursive methods yang bukan tail recursive disebut non-tail recursive

Is Factorial Tail Recursive?

- Apakah method faktorial adalah tail recursive ?

```
int fact(int x){  
    if (x==0)  
        return 1;  
    else  
        return x*fact(x-1);  
}
```

- Bukan tail recursive karena pada saat kembali dari recursive call `x*fact(x-1)`, masih terdapat operasi perkalian.

Another Example

- Apakah method tail() adalah tail recursive?

```
void tail(int i) {  
    if (i>0) {  
        system.out.print(i+"")  
        tail(i-1)  
    }  
}
```

- Method tail() merupakan tail recursive

Third Example

- Apakah method prog() adalah tail recursive?

```
void non prog(int i) {  
    if (i>0) {  
        prog(i-1);  
        System.out.print(i+"");  
        prog(i-1);  
    }  
}
```

- Tidak, karena dibaris awal terdapat recursive call
- Pada tail recursive, recursive call menjadi statement akhir, dan tidak ada recursive call di atasnya.

Advantage of Tail Recursive Method

- Method dengan Tail Recursive, mudah dirubah menjadi iteratif

```
void tail(int i){  
    if (i>0) {  
        system.out.println(i+"");  
        tail(i-1)  
    }  
}
```



```
void iterative(int i){  
    for (;i>0;i--)  
        System.out.println(i+"");  
}
```

- Smart compilers can detect tail recursion and convert it to iterative to optimize code
- Used to implement loops in languages that do not support loop structures explicitly (e.g. prolog)

Converting Non-tail to Tail Recursive

- Method non-tail recursive dapat diubah menjadi tail-recursive method dengan menambahkan parameter tambahan untuk menampung hasil.

`method fact(int n) → fact_aux(int n, int result)`

- Teknik yang digunakan biasanya membuat fungsi tambahan (method `fact(int n)`) yang memanggil method tail recursive.

```
int fact_aux(int n, int result) {
    if (n == 1)
        return result;
    return fact_aux(n - 1, n * result)
}
int fact(n) {
    return fact_aux(n, 1);
}
```


Rekursif Tail : Faktorial()

$$F(n,a) = \begin{cases} a & \text{jika } n=0, n=1 \\ F(n-1,na) & \text{jika } n>1 \end{cases}$$

$$F(4,1) = F(3,4)$$

Fase awal

$$F(3,4) = F(2,12)$$

$$F(2,12) = F(1,24)$$

$$F(1,24) = 24$$

Kondisi Terminal

24

Fase Balik
Rekursif Lengkap

Converting Non-tail to Tail Recursive

- Method tail-recursive Fibonacci diimplementasikan menggunakan dua parameter bantuan untuk menampung hasil.

```
int fib_aux ( int n , int next, int result)
{
    auxiliary parameters!
    if (n == 0)
        return result;
    return fib_aux(n - 1, next + result, next);
}
```

To calculate fib(n) , call fib_aux(n,1,0)