Αντικειμενοστρεφής Προγραμματισμός (Object Oriented Programming)

Α Ν Α Δ Ρ Ο Μ Η (R e c u r s i o n)

Παναγιώτης Σφέτσος, PhD
http://aetos.it.teithe.gr/~sfetsos/
sfetsos@it.teithe.gr

Το πρότυπο Αναδρομή *(Recursion)*

Αναδρομική μέθοδος: Όταν μία μέθοδος ***καλεί τον εαυτό της*** και έχει ***μια συνθήκη τερματισμού.***

- Ατέρμονη αναδρομή είναι προγραμματιστικό λάθος και οδηγεί σε τερματισμό του προγράμματος, λόγω υπερχείλισης της στοίβας *(stack overflow)*.
- Κάθε αναδρομική κλήση μιας μεθόδου δημιουργεί **νέες τοπικές μεταβλητές και παραμέτρους**. Πολλά πολύπλοκα προβλήματα μπορούν να λυθούν με αναδρομή.
- Κλασικό παράδειγμα – η μέθοδος υπολογισμού παραγοντικού: $\mathbf{n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1) \cdot n,}$ (για n>=0)

Το πρότυπο Αναδρομή *(Recursion)*

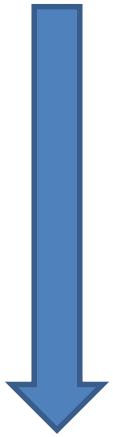
Όπου: **0! = 1**

n! = n(n-1)!

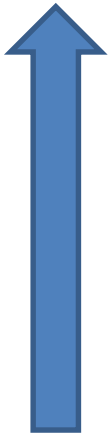
n!=1*2*3*…*(n-1) *n => n! = (n-1)! * n, π.χ. 24! = 23! * 24

Παραδείγματα:

4! = 1*2*3*4 = 3! * 4
3! = 1*2*3 = 2! * 3
2! = 1*2 = 1! *2
1! = 1=1!* 1
0! = 1



4! = 6 * 4 = 24
3! = 2 * 3 = 6
2! = 1 * 2 = 2
1! = 1
0! = 1

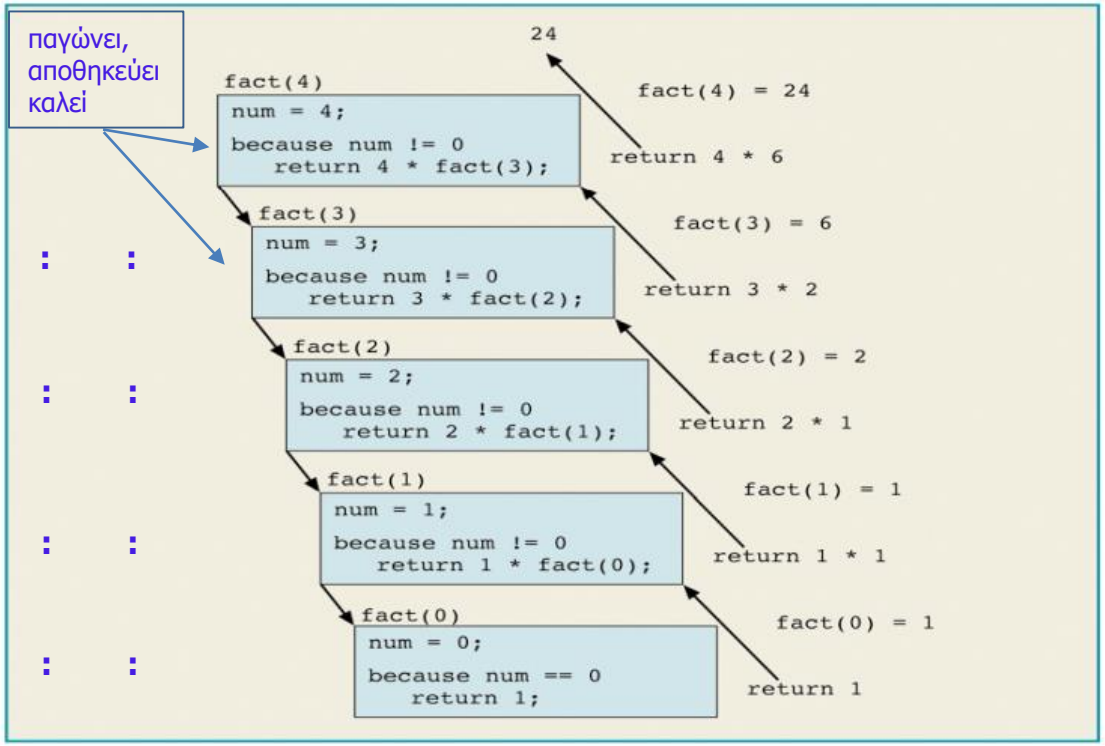


Το πρότυπο Αναδρομή *(Recursion)*

Recursive Factorial Method :

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

Με παράμετρο 4:



Χρησιμοποιώντας την τιμή 1 του 0! (βήμα διακοπής) είναι δυνατός ο υπολογισμός του 4! Επιστρέφοντας στον υπολογισμό του 1!, του 2!, 3! και τέλος 4! *(σχήμα)*

Το πρότυπο Αναδρομή *(Recursion)*

Μια άλλη απεικόνιση, για n=4:

factorial(4) = 4*factorial(3)

factorial(3) = 3*factorial(2)

factorial(2) = 2*factorial(1)

factorial(1) = 1*factorial(0)

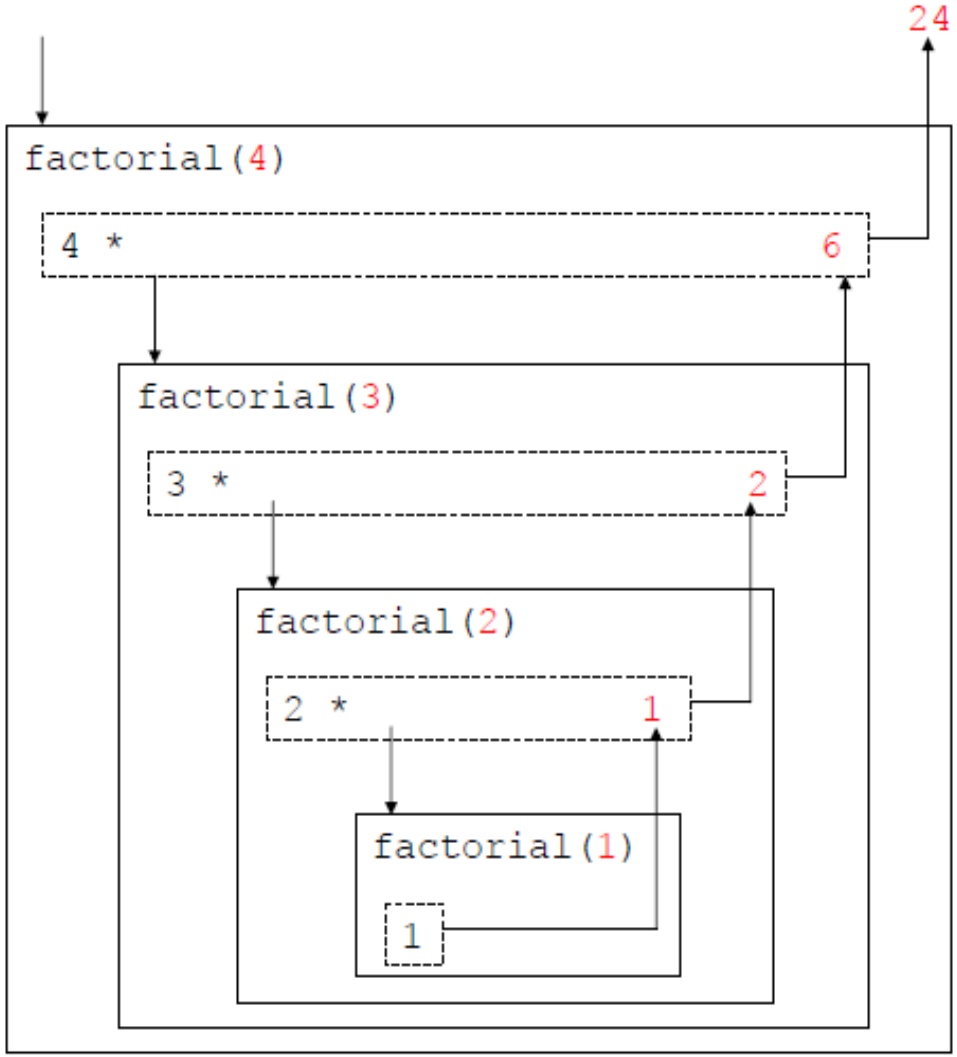
factorial(0) = 1

Η τελευταία τιμή 1 μεταβιβάζεται στην προηγούμενη κλήση και έτσι υπολογίζεται το factorial(1) = 1.

Ομοίως: factorial(2) = 2*1 = 2,

factorial(3) = 3*2 = 6

factorial(4) = 4*6 = 24

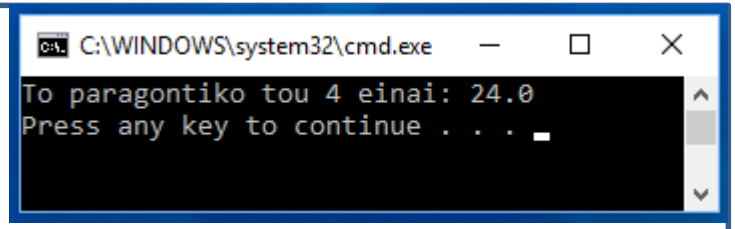


Το πρότυπο Αναδρομή *(Recursion)*

Μια λύση με χρήση αντικειμένου:

```
class Anadromi
 {
  public double paragontiko(int n)
        {
              if(n==0) return 1;   //συνθήκη τερματισμού
              if(n<0) return 0;
              return n*paragontiko(n-1); //αναδρομικό βήμα
 }}

class TestAnadromi
 {public static void main(String args[])
        {
          Anadromi Q =new Anadromi();
        System.out.println("To paragontiko tou 4 einai: "
                              + Q.paragontiko(4)); }}
```

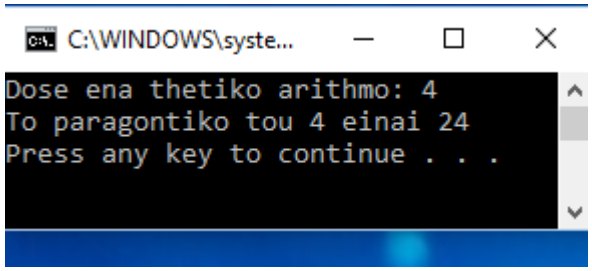


Το πρότυπο Αναδρομή *(Recursion)*

Μια λύση με χρήση στατικής μεθόδου:

```
import java.util.Scanner;
public class Anadromi {
 public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   System.out.print("Dose ena thetiko arithmo: ");
   int n = input.nextInt();
   System.out.println("To paragontiko tou " + n + " einai " +
     factorial(n));
  }

 public static long factorial(int n) {
   if (n == 0) // synthiki termatismou
     return 1;
   else
      return n * factorial(n - 1); // anadromiki klisi
 }}
```

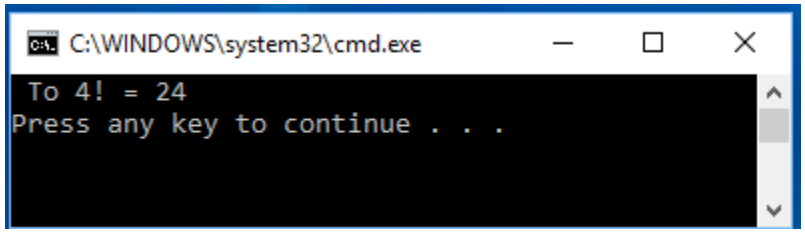


Το πρότυπο Αναδρομή *(Recursion)*

Επαναληπτική υλοποίηση χωρίς αναδρομή:

```
class EpanaliptikoFactorial  {
  public static void main(String args[]){
   int i=Factorial(4);
   System.out.println(" To 4! = "+i);
}

public static int Factorial(int n)
 {
  int i, result;
  result = 1;
  for (i = 2; i<= n; i++)
  {
   result = result * i;
  }
 return result;
}
```



Το πρότυπο Αναδρομή *(Recursion)*

Να λυθεί η άσκηση – 1:

Υπολογισμός δύναμης: power(int a, int n), (2- παράμετροι, βάση και δύναμη)

Είναι:

$a^0=1$

$a^n=a^{n-1}.a \quad (n>=1)$

Μια προτεινόμενη αναδρομική σχέση: p = a*power(a, n-1);

```
int power(int a, int n)
{
  int p;
  if (n==0)
     p = 1;
  else
     p = a*power(a,n-1);
 return p;
 }
```

Παράδειγμα:

power(2,3) => 2* power(2,2)
= 2*2* power(2,1)
= 2*2*2* power(2,0)
= 2*2*2*1= 8

Το πρότυπο Αναδρομή *(Recursion)*

Να λυθεί η άσκηση - 2:

Εμφάνιση των ψηφίων ενός ακέραιου αριθμού αντίστροφα. Χρησιμοποιείστε μια στατική αναδρομική μέθοδο, π.χ.:

```
public static void reverseDisplay(int value)
```

Δηλ. αν **reverseDisplay(87654)**, τότε θα εμφανίσει **45678**.

Ακολουθία αριθμών Fibonacci *(1/5)*

Η ακολουθία αριθμών:
`1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...`
είναι η σειρά *Fibonacci*.

- Χαρακτηριστικό: κάθε αριθμός της σειράς προκύπτει από το άθροισμα των δύο προηγούμενων αριθμών.

- Η ακολουθία Fibonacci εμφανίζεται στην φύση, στα ζώα (κουνέλια) και στα φυτά, αλλά και στην μουσική και την τέχνη γενικότερα.

- Ένας μαθηματικός τύπος για την εύρεση οποιουδήποτε όρου n της σειράς είναι:

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

Προγραμματιστικά;

Ακολουθία αριθμών Fibonacci *(2/5)*

Προγραμματιστικά:

Η ακολουθία ορίζεται ως:

fib(0) = 0;

fib(1) = 1;

fib(index) = fib(index - 2) + fib(index - 1); index >= 2

Fib(n) = fib(n-1) + fib(n-2)

f(1),f(2),f(3),f(4),f(5),f(6),f(7),f(8),...

1, 1, 2, 3, 5, 8, 13, 21,...

1+1 1+2 2+3 3+5 5+8 8+13

- Συνθήκη τερματισμού: οι δύο πρώτοι αριθμοί είναι ίσοι με την μονάδα (1)

Ακολουθία αριθμών Fibonacci *(3/5)*

Δηλαδή,

- Μπορούμε να βρούμε εύκολα την θέση - index ενός αριθμού, π.χ.

  fib(2), επειδή γνωρίζουμε τα fib(0) και fib(1),

  άρα για οποιαδήποτε θέση:

  fib(index) = fib(index - 2) + fib(index - 1).

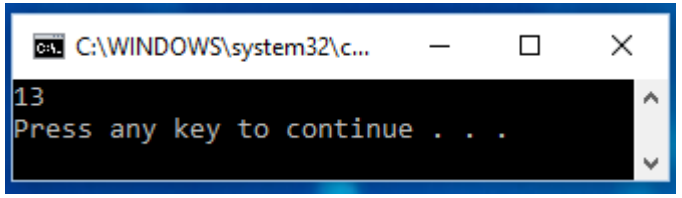
- Αναδρομική κλήση μέχρι ο index να γίνει 0 ή 1.

Ακολουθία αριθμών Fibonacci *(4/5)*

Αναδρομική αναζήτηση όρου στην ακολουθία Fibonacci (με αντικείμενο):

```
class Fibonacci {
  public int Fib(int b)
       {
            if(b==1 || b==2)return 1;
            return (Fib(b-1) + Fib(b-2)); }}

class TestFibonacci {
  public static void main(String args[])
       {
            Fibonacci f = new Fibonacci();
            System.out.println(f.Fib(7)); }}
```

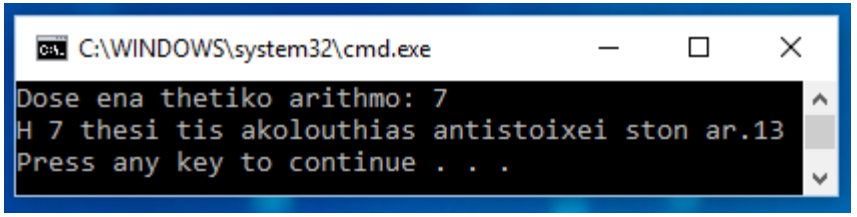


Ακολουθία αριθμών Fibonacci *(5/5)*

Αναδρομική αναζήτηση όρου στην ακολουθία Fibonacci (με στατική μέθοδο):

```
import java.util.Scanner;
public class Fibonacci1 {
 public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Dose ena thetiko arithmo: ");
    int n = input.nextInt();
    System.out.println("H " + n + " thesi tis akolouthias
      antistoixei ston ar." + fibo(n));
 }

public static int fibo (int num) {
 if (num == 1 || num == 2) return 1;
 else return fibo(num-1) + fibo(num-2);
}}
```



Παλίνδρομοι αριθμοί *(1/3)*

Ένας αριθμός λέγεται **παλίνδρομος**, όταν διαβάζεται το ίδιο από αριστερά προς τα δεξιά και από τα δεξιά προς τα αριστερά, δηλ. τα στοιχεία του είναι 'συμμετρικά' ως προς το κέντρο, π.χ. οι αριθμοί 353, 42624, κλπ.

- Μια τεχνική είναι να **αντιστρέψουμε τον αριθμό** και μετά να τον συγκρίνουμε με τον αρχικό αριθμό. Αν είναι ίσοι τότε είναι παλινδρομικοί. Ο αριθμός χωρίζεται στα ψηφία του διαιρούμενος με το 10 ή με δυνάμεις του 10. Χρησιμοποιούμε τα σύμβολα της ακέραιας διαίρεσης (/) και του υπολοίπου (%).

Έστω ο αρ. n=22342:

- εντοπίζω το τελευταίο ψηφίο το 2, από το υπόλοιπο της διαίρεσης του αριθμού με το 10 και αναδρομικά
- επαναλαμβάνω το ίδιο (εντοπισμός του τελευταίου ψηφίου) αλλά με τον νέο αριθμό (2234), που είναι το πηλίκο του αριθμού με το 10, έτσι

Παλινδρομικοί αριθμοί *(2/3)*

- κάθε φορά που εντοπίζω το τελευταίο ψηφίο θα δημιουργώ ένα νέο αριθμό (τον αντίστροφο) που τελικά θα τον συγκρίνουμε με τον αρχικό.
- συνθήκη τερματισμού το πηλίκο να ισούται με 0.

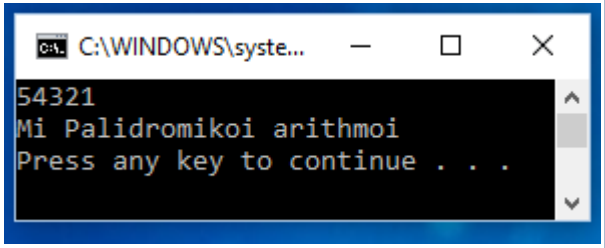
n	**Πηλίκο**	**Υπόλοιπο**	**Αντίστροφος**
22342 / 10 =	2234	22342 % 10 = 2	0*10 + 2 = 2
2234 / 10 =	223	2234 % 10 = 4	2*10 +4 = 24
223 / 10 =	22	223 % 10 = 3	24*10+3 = 243
22 / 10 =	2	22 % 10 = 2	243*10+2 = 2432
2 / 10 =	0	2 % 10 = 2	2432*10+2 = 24322

- **Αλγόριθμος:** να καλούμε αναδρομικά την μέθοδο αντιστροφής με τον ακέραιο διαιρούμενο (ακέραια διαίρεση) δια του 10. Αν ο αριθμός π.χ. είναι ο 22342, τότε η κλήση κάθε φορά θα είναι: 2234, 223, 22, 2. Πρέπει όμως να τοποθετούμε σε μια μεταβλητή τον εκάστοτε αριθμό δηλ.: 0+2, 10*2+4, 10*24+3, κλπ.).

Παλινδρομικοί αριθμοί *(3/3)*

```
class palidromes {
   static int rev=0;
   public static int REV(int num)
        {
        if(num>0) {
          rev=(rev*10)+ (num %10);
           //System.out.println(rev*10+" "+num%10+ "  "+rev );
          REV(num/10);   //klisi tis methodoy me anadromi
          } return rev; } }

class Testpalidromes {
 public static void main(String args[]) {
   int x=12345;
   int y=palidromes.REV(x);
   System.out.println(y);
   if (x==y) System.out.println("Palidromikoi arithmoi");
     else System.out.println("Mi Palidromikoi arithmoi"); } }
```



Παλινδρομικά Strings *(1/2)*

Όπως με τους αριθμούς έτσι και ένα String λέγεται **παλινδρομικό**, όταν διαβάζεται το ίδιο από αριστερά προς τα δεξιά και από τα δεξιά προς τα αριστερά, π.χ. anna, anana, radar, κλπ.

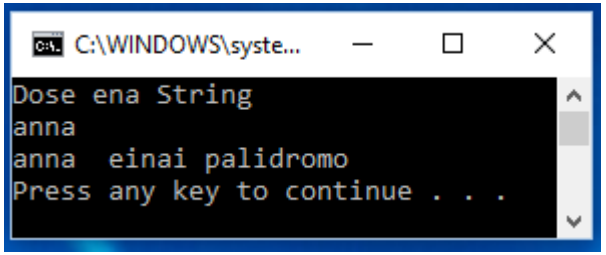
- Μια τεχνική είναι να συγκρίνουμε αναδρομικά τον 1ο χαρακτήρα με τον τελευταίο, τον 2ο με τον προτελευταίο κλπ. Αν σε κάποια αναδρομική κλήση βρεθούν άνισοι χαρακτήρες, τότε τελειώνει η αναδρομή και τα Strings δεν είναι παλινδρομικά.

- Ο αλγόριθμος είναι: αν ο 1ος χαρ. είναι ίδιος με τον τελευταίο, τότε συνεχίζουμε με το δεύτερο ζεύγος (2ος με προτελευταίο κλπ.). Κάθε φορά η αναδρομή καλείται με το substring κομμένο από τον 1ο και τελευταίο χαρ. του String.

Παλινδρομικά Strings *(2/2)*

```
import java.util.Scanner;
public class PalindromeString {
    public static boolean isPal(String s){
        if(s.length() == 0 || s.length() == 1)
            return true; //einai palidromo
        //an o 1os einai idios me ton teleytaio, tote synexise gia
        // to epomeno substring me 1o kai teleytaio komena
      if(s.charAt(0) == s.charAt(s.length()-1))
       return isPal(s.substring(1, s.length()-1));
        // an ohi tote o eleghos peftei se lathos.
        return false;}

  public static void main(String[]args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Dose ena String ");
        String x = sc.nextLine();
        if(isPal(x))System.out.println(x + "  einai palidromo");
        else System.out.println(x + "  den einai palidromo");  } }}
```



Πρώτοι Αριθμοί *(1/2)*

Στα μαθηματικά **πρώτος αριθμός** (ή απλά πρώτος) είναι ένας φυσικός αριθμός μεγαλύτερος της μονάδας με την ιδιότητα οι μόνοι φυσικοί διαιρέτες του να είναι η **μονάδα και ο εαυτός του** *(wiki)*, π.χ. 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, κλπ.

- Ο αρ. 5 είναι πρώτος, γιατί διαιρείται δια του 1 και του 5, ενώ ο αρ. 8 είναι σύνθετος, γιατί διαιρείται δια του 1, του 8 και του 2 (8 % 2 = 0).

- Αναδρομικός τρόπος (αλγόριθμος του WILSON για όχι πολύ μεγάλο n)

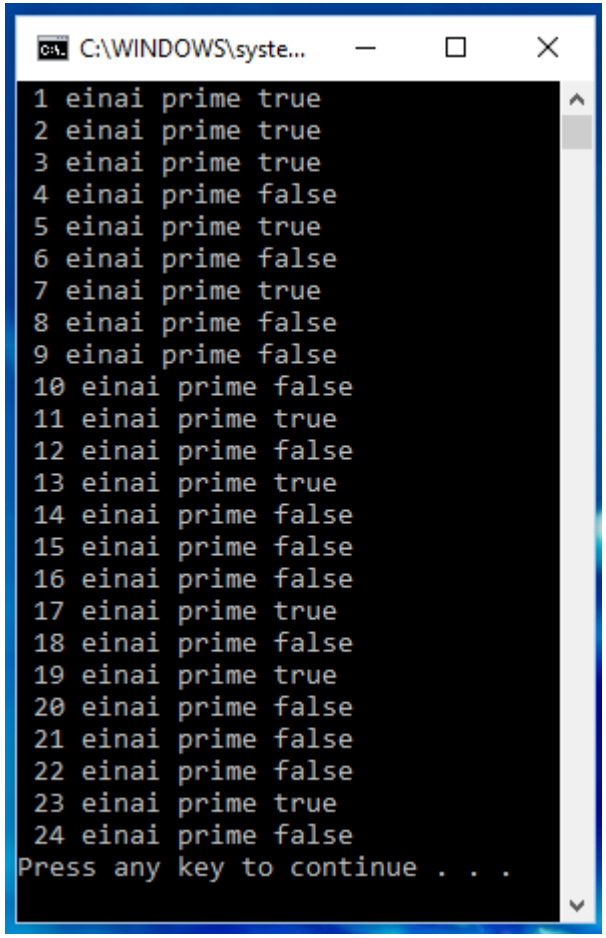
Αν το:

(1+factorial (n-1)) % n ισούται με 0, τότε το n είναι πρώτος αρ.

Πρώτοι Αριθμοί *(2/2)*

Έλεγχος για τους πρώτους 25 αριθμούς

```
class Prime {
 public static void main(String[] arg) {
    for(int i = 1; i < 25; i++) {
     System.out.println(" " + i + " einai prime"
        + isPrime(i, i-1));
 }}
 static boolean isPrime(int num, int div) {
   if(div <= 1) {return true;}
   if(num % div == 0)    {return false;}
   return isPrime(num, div-1); }}
```



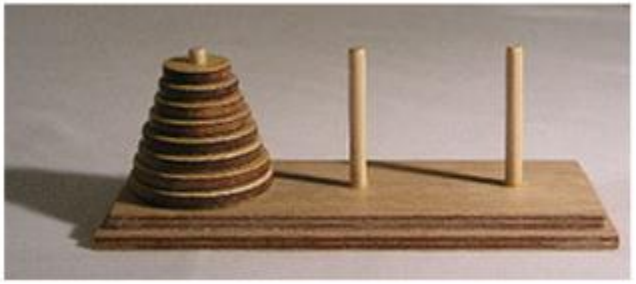
Οι Πύργοι του Ανόι (towers of Hanoi) *(1/6)*

Σύμφωνα με τον μύθο σε έναν ναό της Άπω Ανατολής, οι ιερείς προσπαθούσαν να μεταφέρουν μια στοίβα χρυσών δίσκων από ένα στύλο σε ένα άλλο. Ο αρχικός στύλος έχει 64 δίσκους τοποθετημένους σε φθίνουσα σειρά μεγέθους από κάτω προς τα πάνω. Η μεταφορά των δίσκων από τον πρώτο στύλο στον τρίτο πρέπει να γίνει με τους εξής κανόνες:

- Κάθε φορά μεταφέρεται μόνο ένας δίσκος.
- Σε καμιά περίπτωση δεν μπορεί ένας μεγαλύτερος δίσκος να τοποθετηθεί πάνω σε ένα μικρότερο
- Ο δεύτερος στύλος μπορεί να χρησιμοποιηθεί για προσωρινή τοποθέτηση.

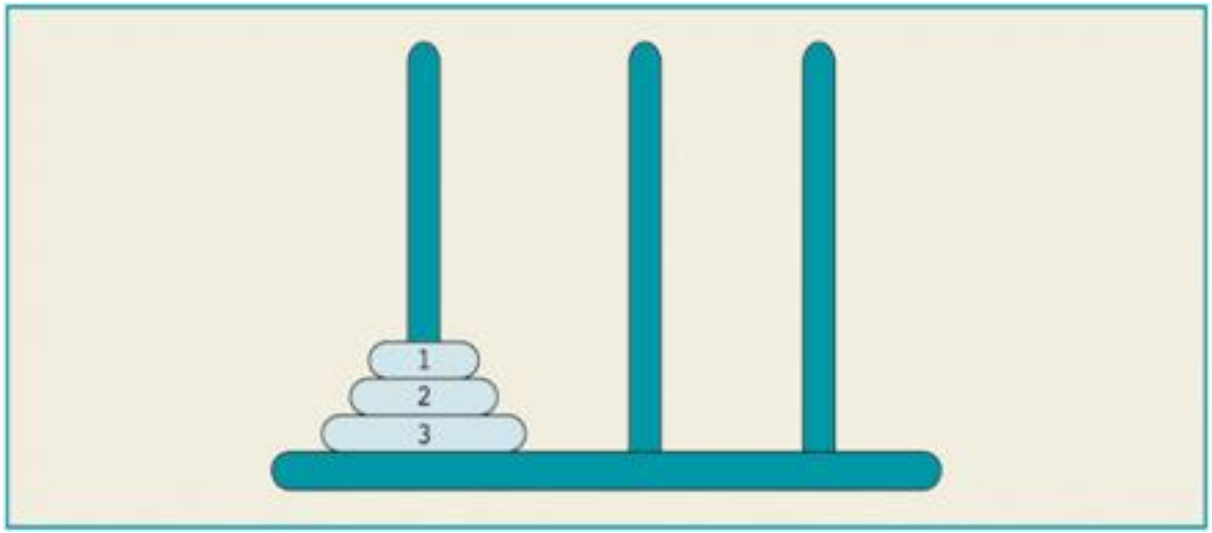
Για τη μεταφορά :

3 δίσκων απαιτούνται 7 κινήσεις δηλ 2^3- 1 κινήσεις.
4 δίσκων απαιτούνται 15 κινήσεις δηλ 2^4- 1 κινήσεις.
5 δίσκων απαιτούνται 31 κινήσεις δηλ 2^{5-} 1 κινήσεις.



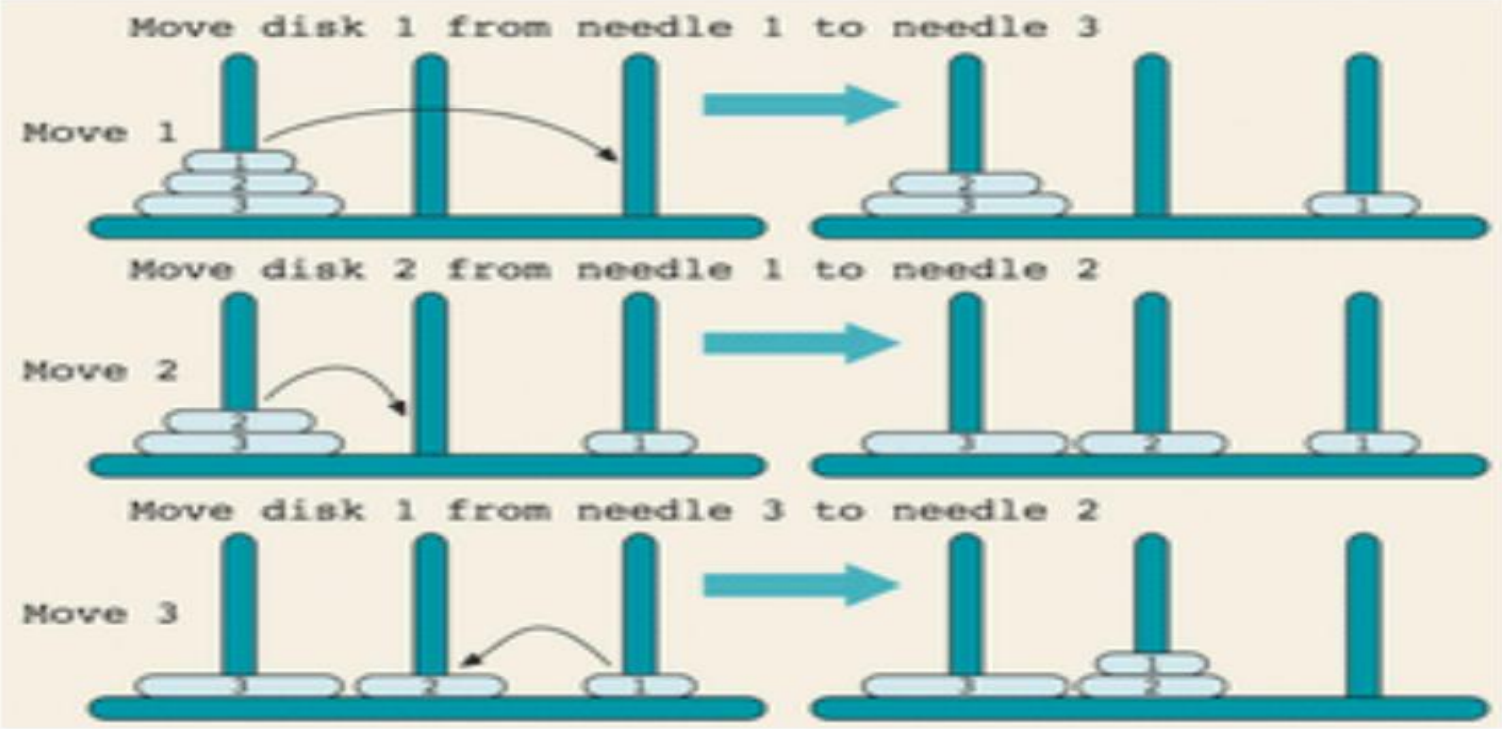
Οι Πύργοι του Hanoi (towers of Hanoi) *(2/6)*

Το πρόβλημα των 3-δίσκων μπορεί να λυθεί και χωρίς πρόγραμμα.

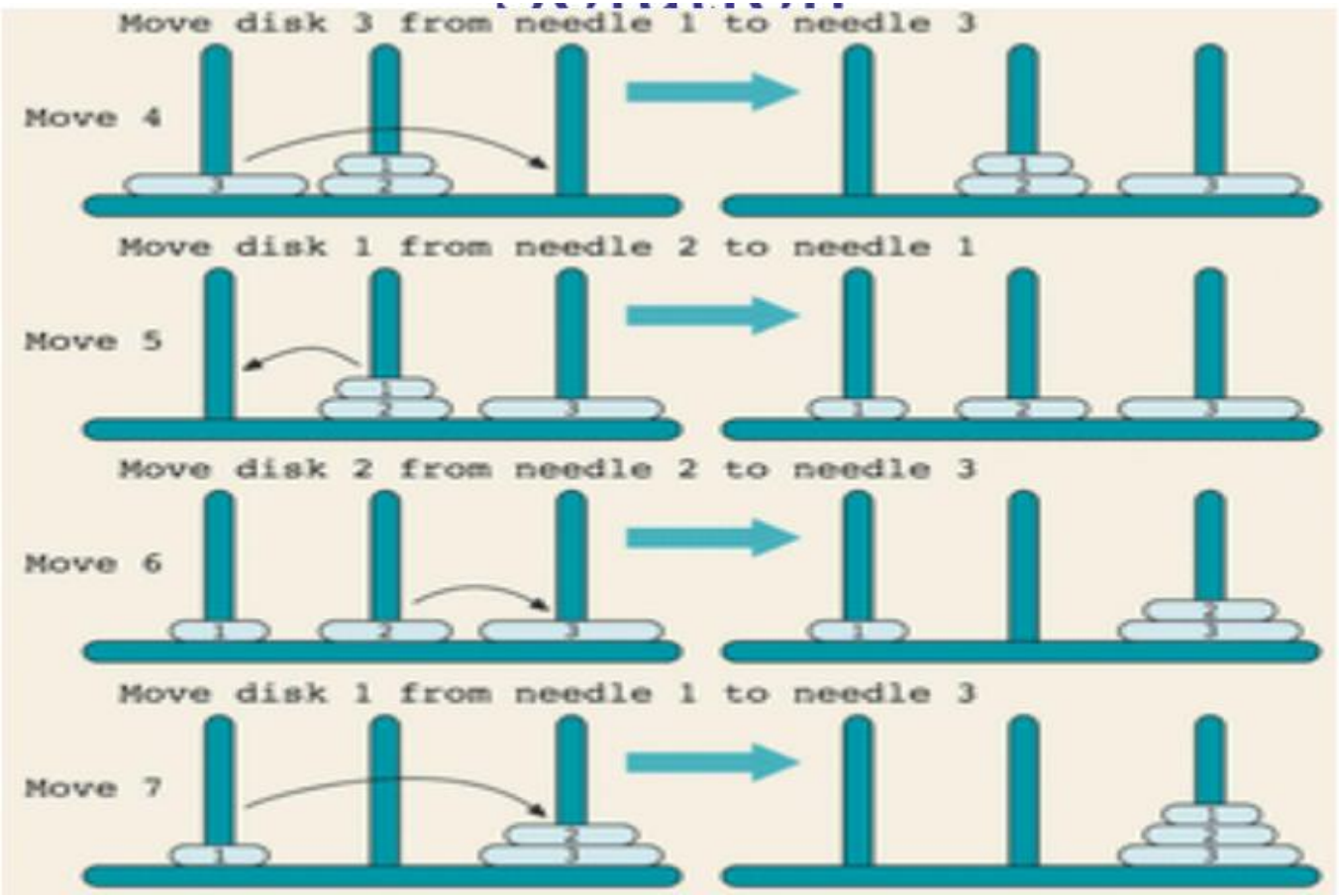


- Σε κάθε μεταφορά (transfer) n δίσκων, υπάρχει ένας στύλος αφετηρία *(source)*, ένας στύλος προορισμός *(destination)*, και ένας βοηθητικός / αποθηκευτικός *(storage)*. Η μέθοδος μετακίνηση μπορεί να περιγραφεί ως εξής: **moveDisks(int n, char source, char destination, char storage)**

Οι Πύργοι του Hanoi (towers of Hanoi) *(3/6)*



Οι Πύργοι του Hanoi (towers of Hanoi) *(4/6)*

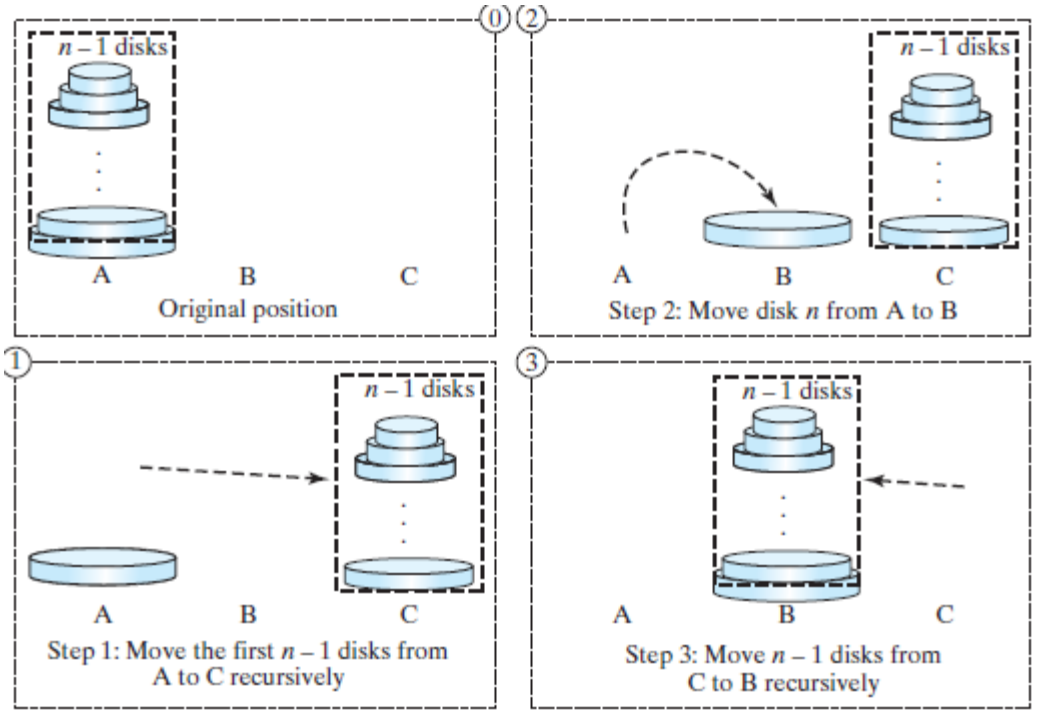


Οι Πύργοι του Hanoi (towers of Hanoi) *(5/6)*

Για περισσότερους δίσκους; Δύσκολο πρόβλημα.

'Σπάει' σε 3 υπό-προβλήματα :

(1) Μετακινούμε αναδρομικά τους πρώτους n-1 δίσκους από το A στο C με την βοήθεια του B (βήμα 1).



(2) Μετακινούμε τον δίσκο n από το A στο B (βήμα 2)

(3) Μετακινούμε αναδρομικά τους n-1 δίσκους από το C στο B, με την βοήθεια του A (βήμα 3).

(fig. από το βιβλίο του D.Liang)

Οι Πύργοι του Hanoi (towers of Hanoi) *(6/6)*

Να λυθεί η άσκηση – 3:

Να γραφεί ΄το πρόγραμμα που μετακινεί οποιοδήποτε πλήθος δίσκων (το πλήθος θα εισάγεται από το πληκτρολόγιο) από μία θέση (π.χ. A) σε μια άλλη (π.χ. C) με την βοήθεια μιας τρίτης (π.χ. B). Αν λάβετε υπόψιν τον προηγούμενο αλγόριθμο θα χρειαστείτε τις παρακάτω μεταβλητές:

- το πλήθος των δίσκων n
- τρείς μεταβλητές τύπου char τις ‘A’, ‘B’, και ‘C’

Μία αναδρομική μέθοδος (σύμφωνα με τον προηγούμενο αλγόριθμο) θα ήταν της μορφής:

```
void moveDisks(int n, char A, char C, char B)
```