CHAPTER _1

RECURSION

Recursive Definitions

- Recursion

عملية حل مشكلة عن طريق تصغيرها الي إصدارات تقل منها بحد ذاتها

  - Process of solving a problem by reducing it to smaller versions of itself
  - Or the method is calling itself.

أو الطريقة تستدعي نفسها

- Recursive definition
  - Definition in which a problem is expressed in terms of a smaller version of itself
  - Has one or more base cases

التعريف الذي يتم فيه التعبير عن مشكلة من حيث نسخة أصغر مها

لديه حالة أساسية واحدة أو أكثر

Recursive Definitions (Cont'd)

- Recursive algorithm
  - Algorithm that finds the solution to a given problem by reducing the problem to smaller versions of itself
  - Has one or more base cases
  - Implemented using recursive methods

الخوارزمية التي تجد الحل لمشكلة معينة عن طريق تقليل المشكلة إلى إصدارات أصغر من نفسها

لديه حالة أساسية واحدة أو أكثر

نُفّذت باستخدام طرق التكرار

- Recursive method
  - Method that calls itself
  - Using the stack data structure to store the self-calling
  - Functions.

تقوم باستدعاء نفسها

يتم استخدام الستاك لتخزين الاستدعاءات

عبارة عن دالة

- Base case
  - Case in recursive definition in which the solution is obtained directly
  - Stops the recursion

حالة في تعريف متكرر يتم فيها الحصول على الحل مباشرة

يوقف التكرار

The Idea

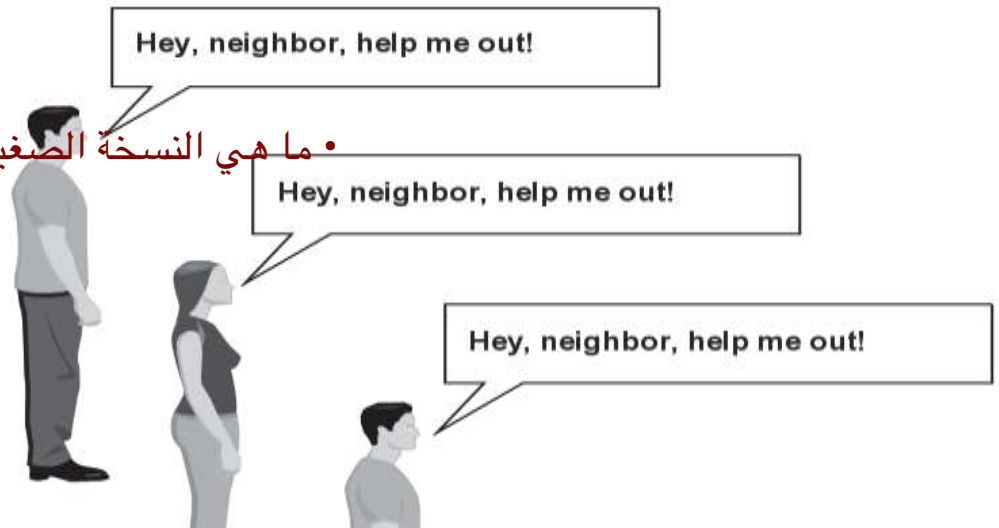
- Recursion is all about breaking a big problem into smaller occurrences of that same problem.
  - Each person can solve a small part of the problem.
    - What is a small version of the problem that would be easy to answer?
    - What information from a neighbor might help me?

يدور التكرار حول تقسيم مشكلة كبيرة إلى تكرارات أصغر لنفس المشكلة.

• يمكن لكل شخص حل جزء صغير من المشكلة.

• ما هي النسخة الصغيرة من المشكلة التي سيكون من السهل الإجابة عليها؟

ما هى المعلومات من الجار قد تساعدنى؟



Recursion Parts

- Every recursive method involves at least 2 cases:
  - **Base case**: A simple occurrence that can be answered directly.

    الحالة الأساسية. حدث بسيط يمكن ايجاد الحل بشكل مباشر.
  - **Recursive case**: A more complex occurrence of the problem that cannot be directly answered, but can instead be described in terms of smaller occurrences of the same problem.

    حالة تكرارية
    حدوث أكثر تعقيداً للمشكلة لا يمكن الإجابة عليه مباشرة ، ولكن
    يمكن بدلاً من ذلك وصفه من حيث التكرارات الأصغر لنفس المشكلة.

Recursion Implementation

- Consider the following method to print a line of **(*)** characters:

طباعة سطر من حرف ال *

```
// Prints a line containing the given number of stars.
// Precondition: n >= 0
public static void printStars(int n) {
    for (int i = 0; i < n; i++) {
        System.out.print("*");
    }
    System.out.println();   // end the line of output
}
```

شرط اساسي n>=0

- We must write a recursive version of this method (that calls itself).

يجب عمل نسخةبها استدعاء الذات

  - To solve the problem without using any loops.
  - Hint: Your solution should print just one star at a time.

لحل هذة المشكلة بدون حلقات تكرارية
الحل يطبع نجمة واحدة كل مرة

Base Case

الحالات لاخذها بالاعتبار

- What are the cases to be considered?
  - What is a very easy number of stars to print without a loop?

اسهل الحالات اذا n تساوي ١ تطبعها بدون تكرار

```
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else {
        ...
    }
}
```

Adding more cases

التعامل مع الحالات الاخري بدون تكرار حل سئ

- Handling additional cases, with no loops (**in a bad way):**

```
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else if (n == 2) {
        System.out.print("*");
        System.out.println("*");
    } else if (n == 3) {
        System.out.print("*");
        System.out.print("*");
        System.out.println("*");
    } else if (n == 4) {
        System.out.print("*");
        System.out.print("*");
        System.out.print("*");
        System.out.println("*");
    } else ...
}
```

Adding more cases (Part 2)

- Taking advantage of the repeated pattern (somewhat better): تحسين الحل مع الاستفادة من التكرار

```
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else if (n == 2) {
        System.out.print("*");
        printStars(1);    // prints "*"
    } else if (n == 3) {
        System.out.print("*");
        printStars(2);    // prints "**"
    } else if (n == 4) {
        System.out.print("*");
        printStars(3);    // prints "***"
    } else ...
}
```

Using the recursion properly

- Condensing the recursive cases into a single case:

```
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else {
        // recursive case; print one more star
        System.out.print("*");
        printStars(n - 1);
    }
}
```

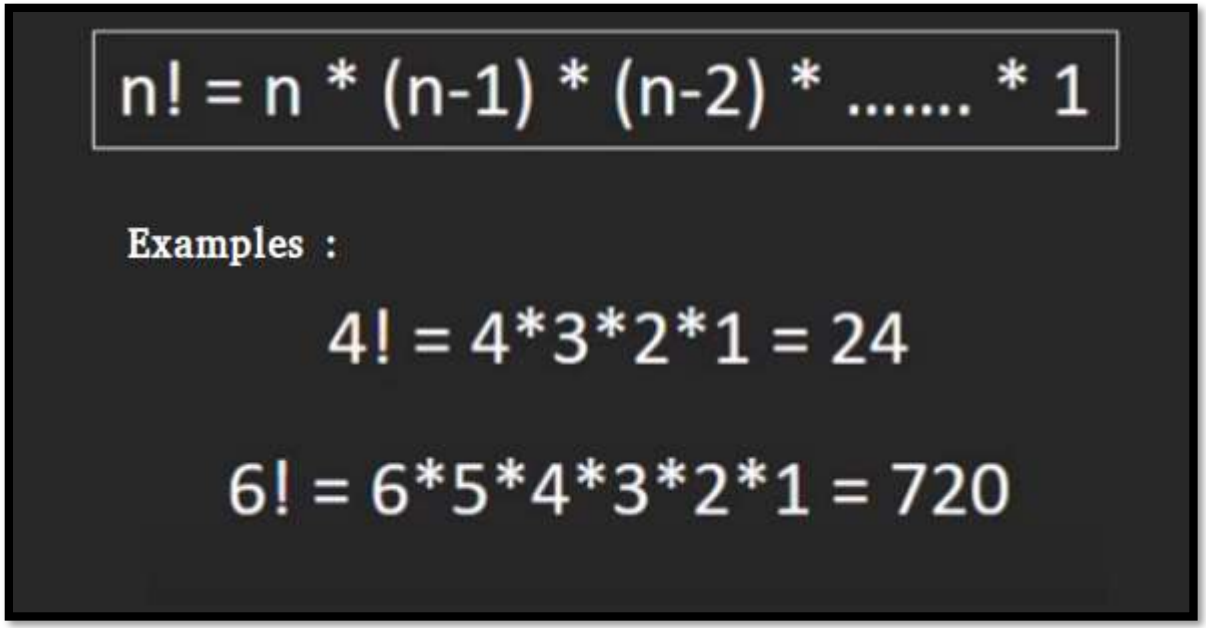
االستخدام الصحيح للتكرار وحاالت االساسية و التكرارية

Designing the recursive method

- Understand problem requirements — فهم متطلبات المشكلة
- Determine limiting conditions — تحديد الشروط المقيدة
- Identify base cases — تحديد الحالات الأساسية
- Provide direct solution to each base case — قدم حلاً مباشرًا لكل حالة أساسية
- Identify general case(s) — تحديد الحالة (الحالات) العامة
- Provide solutions to general cases in terms of smaller versions of general cases — تقديم حلول للحالات العامة من حيث الإصدارات الأصغر للحالات العامة

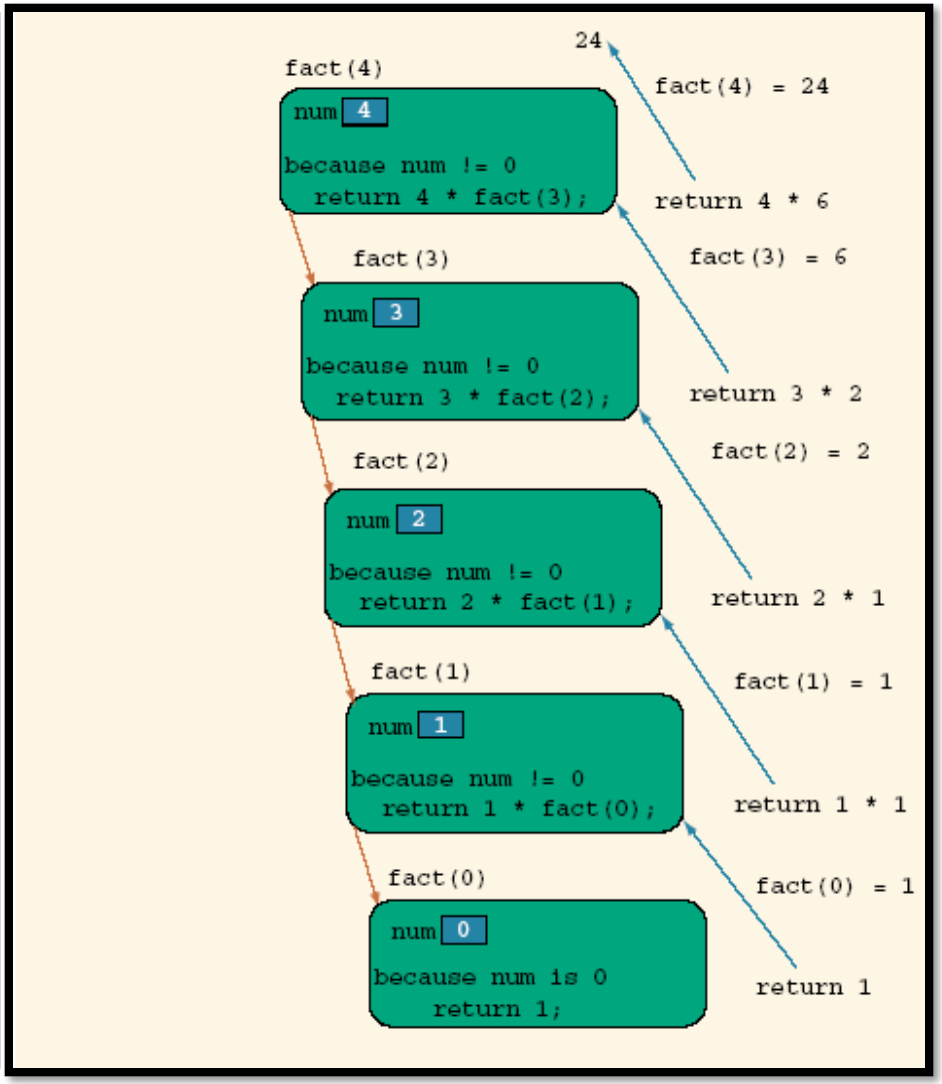
Recursive factorial method

- Factorial of a non-negative integer, is multiplication of all integers smaller than or equal to n. For example factorial of 6 is 6*5*4*3*2*1 which is 720.



Factorial : Recursion version

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



Fibonacci Series: Recursion version

- In Fibonacci series, next number is the sum of previous two numbers. The first two numbers of Fibonacci series are 0 and 1.
- The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...
- مضروب عدد صحيح غير سالب هو ضرب كل الأعداد الصحيحة الأصغر من أو تساوي n.

$$F_0=0 \text{ and } F_1=1$$
$$F_n=F(n-1)+F(n-2)$$

على سبيل المثال ، مضروب 6 هو 6 * 5 * 4 * 3 * 2 * 1 وهو 720

Fibonacci series : Recursion version

```
class fibonacci
{
    static int fib(int n)
    {
    if (n <= 1)
       return n;
    return fib(n-1) + fib(n-2);
    }
```

Recursion disadvantages

- Fairly slower than its iterative solution.
- For each step we make a **recursive** call to a function.
- May cause stack-overflow If the **recursion** goes too deep to solve the problem.
- Difficult to debug and trace the values with each step of **recursion**.

أبطأ إلى حد ما من الحل الحلقات التكرارية.

لكل خطوة نقوم بإجراء استدعاء متكرر للدالة

قد يتسبب في تجاوز حجم الستاك إذا توغلت الريكيرشن في حل المشكلة.

من الصعب تصحيح وتتبع القيم مع كل خطوة من العودية

Important Definitions:

- **Iterative:** A method or algorithm that repeats steps using one or more loops.
- **Recursive:** A method or algorithm that invokes itself one or more times with
- dierent arguments.
- **Base case:** A condition that causes a recursive method not to make another recursive call.
- **Factorial:** The product of all the integers up to and including a given integer.
- **Binary:** A system that uses only zeros and ones to represent numbers. Also known as \base 2".

تعريفات مهمة:

تكراري
طريقة أو خوارزمية تكرر الخطوات باستخدام حلقة واحدة أو أكثر.

الاستدعاء الذاتي
طريقة أو خوارزمية تستدعي نفسها مرة أو أكثر باستخدام وسيطات مختلفة.

حالة الاساسية:
شرط يتسبب في عدم قيام أسلوب تكراري بإجراء استدعاء ذاتي اخر.

المضروب
حاصل ضرب جميع الأعداد الصحيحة حتى عدد صحيح معطى

الثنائي
نظام يستخدم فقط الأصفار والآحاد لتمثيل الأرقام.
يُعرف أيضًا باسم \ base 2 ".