Data Structures and Algorithms

Recursion

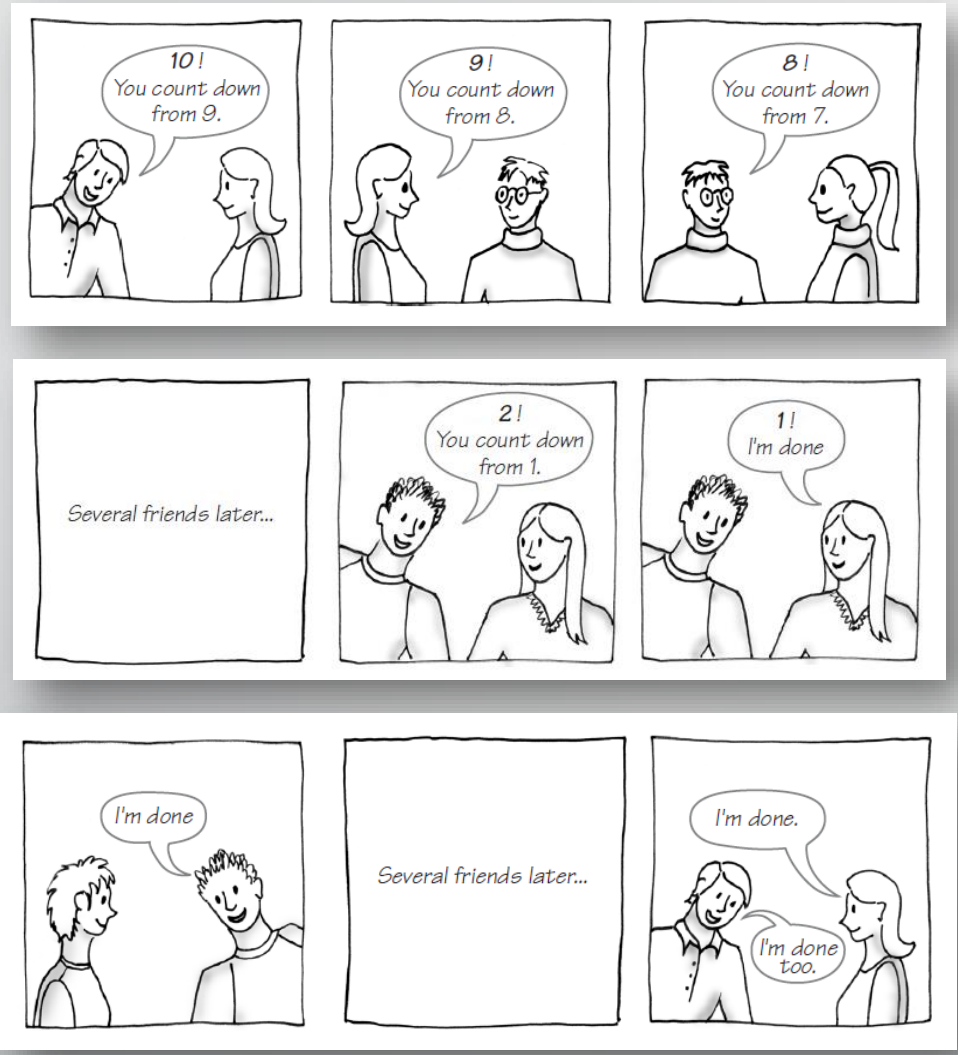
Amilcar Aponte

amilcar@cct.ie

What Is Recursion?

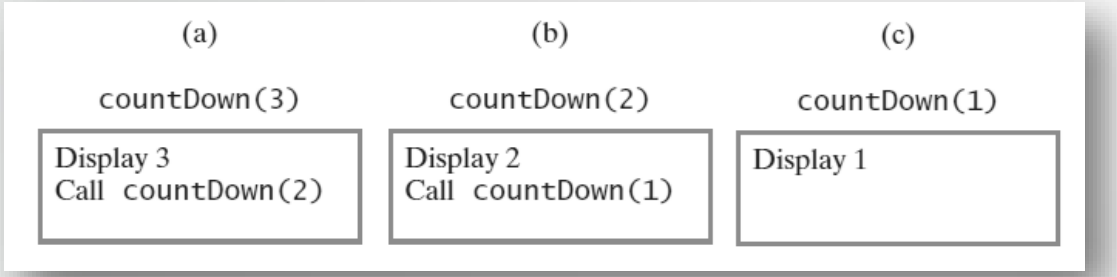
- The process in which a **method calls itself directly or indirectly** is called recursion and the corresponding method is called as recursive method.

Example: The Countdown

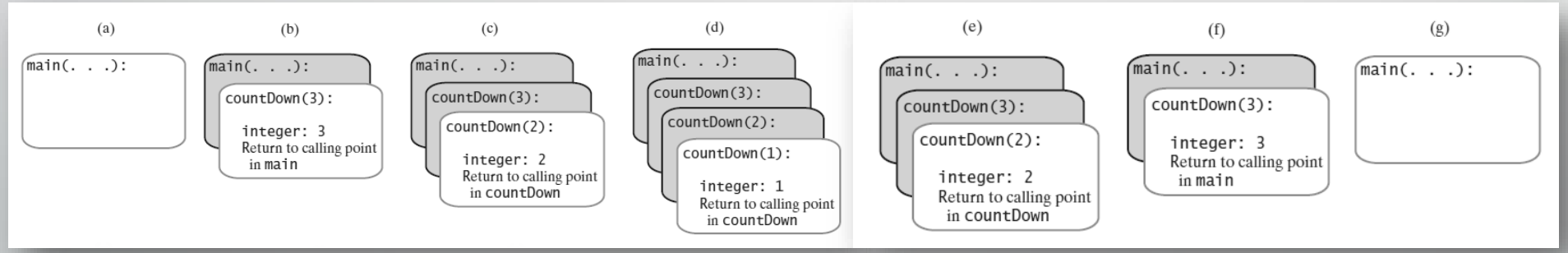



```java
/** Counts down from a given positive integer.
    @param integer  An integer > 0. */
public static void countDown(int integer)
{
   System.out.println(integer);
   if (integer > 1)
      countDown(integer - 1);
} // end countDown
```

Tracing a Recursive Method



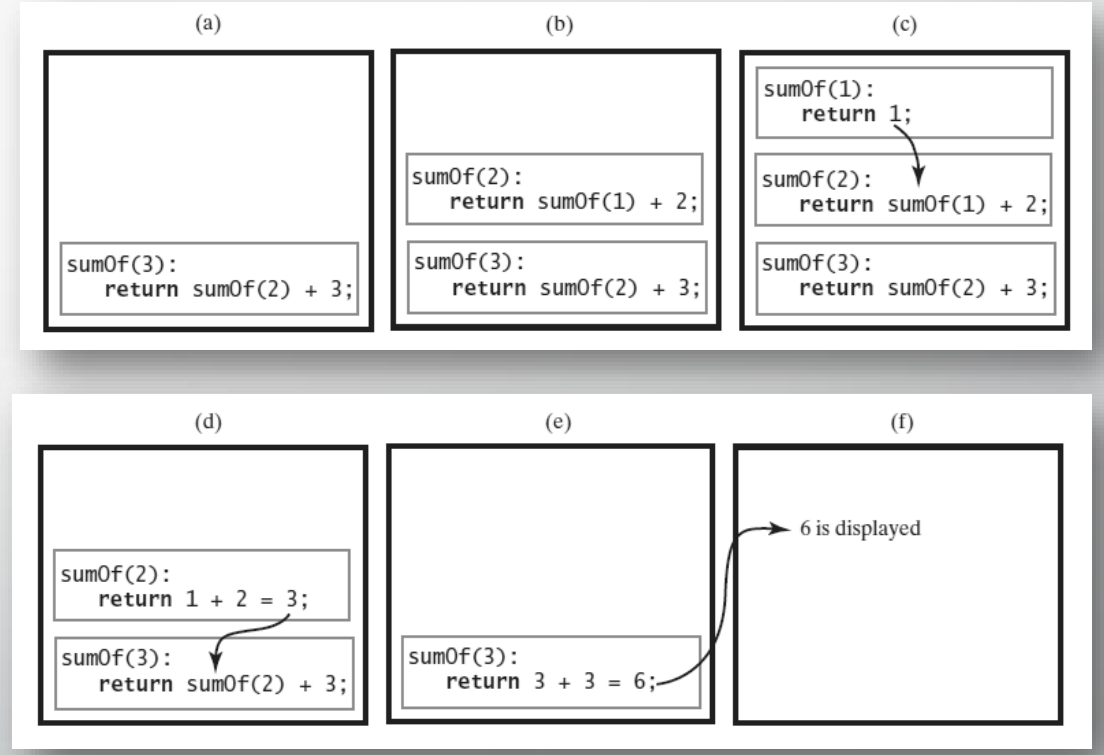
The effect of the method call **`countDown(3)`**



The stack of activation records during the execution of the call **`countDown(3)`**

Let’s see another example

$$\sum_{i=1}^{n} i$$



Code solution

```
/** @param n  An integer > 0.
    @return  The sum 1 + 2 + ... + n. */
public static int sumOf(int n)
{
   int sum;
   if (n == 1)
      sum = 1;                    // Base case
   else
      sum = sumOf(n - 1) + n; // Recursive call

   return sum;
} // end sumOf
```

Recursive Definitions

- A recursive method consists of two parts:
  - The **anchor** or **ground** or **base case**, the basic elements that are the building blocks of all other elements of the set
  - Rules that allow for the construction of new objects out of basic elements or objects that have already been constructed

Content of a Recursive Method

- **Base case**
  - Values of the input variables for which we perform no recursive calls are called base cases (there should be at least one base case).
  - Every possible chain of recursive calls must eventually reach a base case.
- **Recursive calls**
  - Calls to the current method.
  - Each recursive call should be defined so that it makes progress towards a base case.

A quick exercise

- Remember the binary search?
- Try to write it using recursion instead of a while loops

Binary Search

```
1  /**
2   * Returns true if the target value is found in the indicated portion of the data array.
3   * This search only considers the array portion from data[low] to data[high] inclusive.
4   */
5  public static boolean binarySearch(int[ ] data, int target, int low, int high) {
6    if (low > high)
7      return false;                                    // interval empty; no match
8    else {
9      int mid = (low + high) / 2;
10     if (target == data[mid])
11       return true;                                   // found a match
12     else if (target < data[mid])
13       return binarySearch(data, target, low, mid - 1);   // recur left of the middle
14     else
15       return binarySearch(data, target, mid + 1, high);  // recur right of the middle
16   }
17 }
```

That’s all folks

- Any questions?