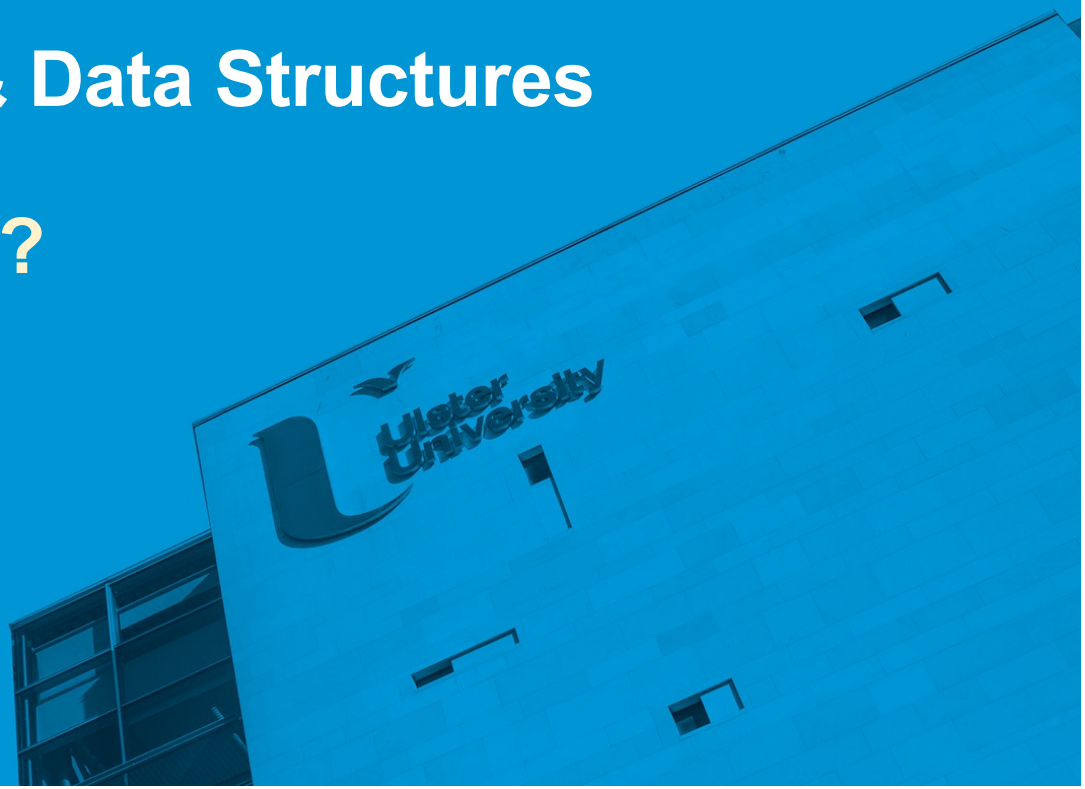
Ulster University

COM498 Algorithms & Data Structures

4.1 What is Recursion?



Recursion

What is recursion?

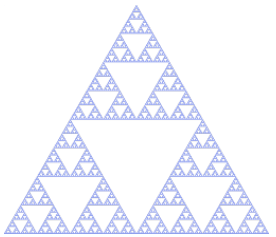
- When something is specified in terms of itself

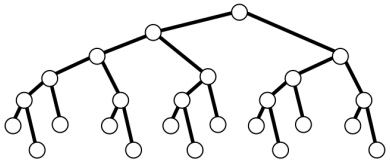
Why learn recursion?

- It represents a new mode of thinking
- Provides a powerful programming paradigm
- Can give insight into nature of computation

Naturally self-referential computational artifacts

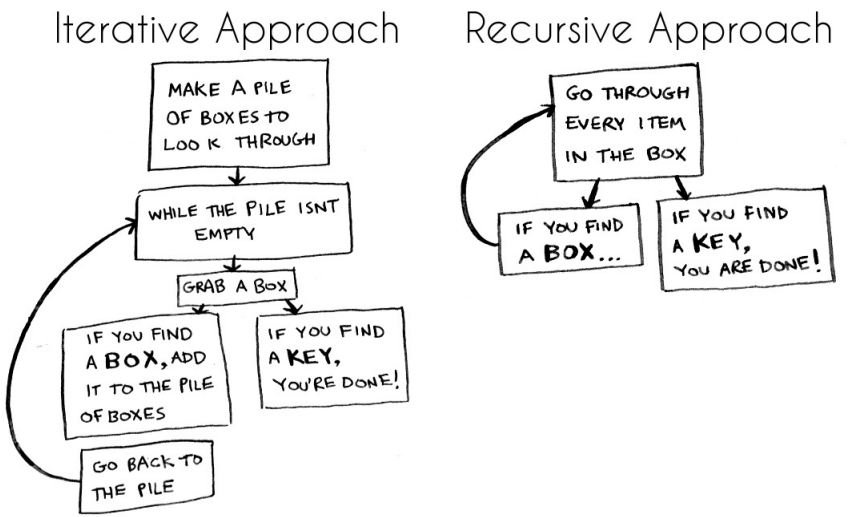
- File systems with folders containing folders
- Fractal graphics patterns
- Divide and conqueror algorithms





Iteration & Recursion

- Repetition is a major feature of many algorithms (repeating actions rapidly is a key ability of computers!)
- Two problem-solving processes involve repetition:
    1. Iteration
    2. Recursion
- At times, iterative solutions are elusive or extremely complex
- Recursive solutions can (sometimes) provide an elegant alternative



Example: The Countdown (10 to 1)

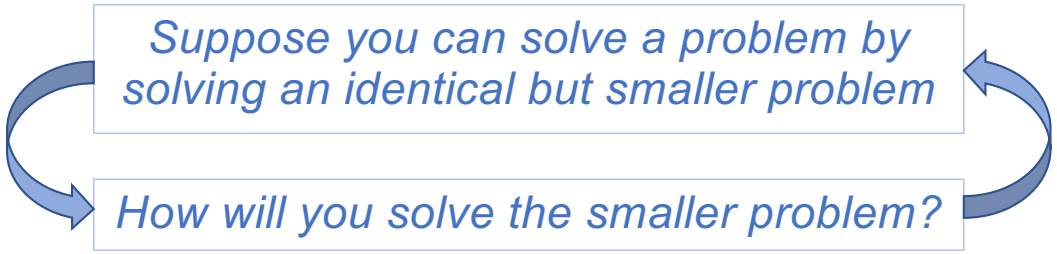
- First person counts down from 10 then asks next person to count down from 9, etc.
- Each person calls out the *N* in their countdown
- Each successive person is solving the smaller problem $(N - 1)$

Recursion is a problem-solving process that breaks a problem down into identical smaller problems

What is Recursion?

Ulster University

- Recursion is a problem-solving process that breaks a problem into identical but smaller problems



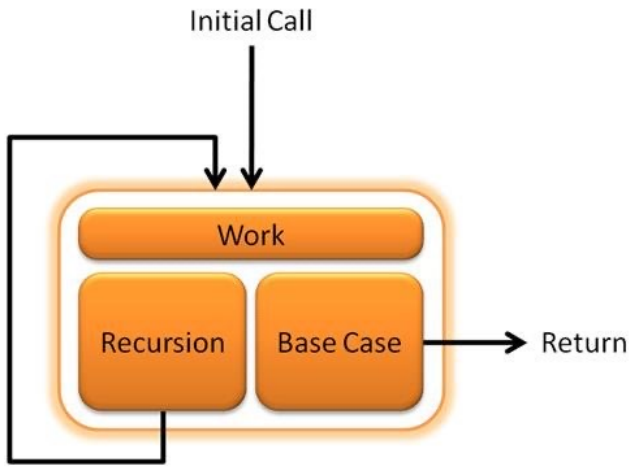
- If you use recursion you will need to solve an even smaller problem (that is just like the original problem in every respect)
- Eventually you will reach a smaller problem whose solution you know
- Solution to smallest problem contributes a portion of the solution to the larger problem

What is Recursion (in programming)?

- Recursion happens when a method calls itself *within its own definition*
- A method that calls itself is a recursive method

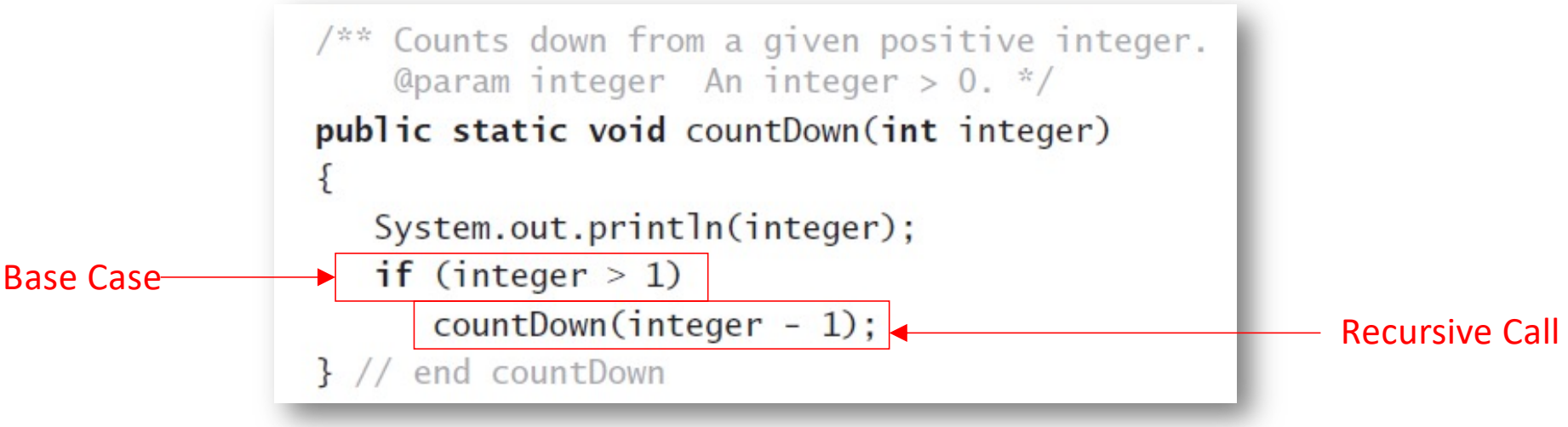
- Two main parts of a recursive method:

1. **Base Case**: important as without it the method would never stop (the smallest identical problem that has a known solution)
2. **Recursive Call**: the recursive invocation of a recursive method (solving the smaller identical problem)



Example: The Countdown

- Recursive method to perform the countdown (in this case, print out all integers from a given *N* counting down to 1):



Tracing a Recursive Method

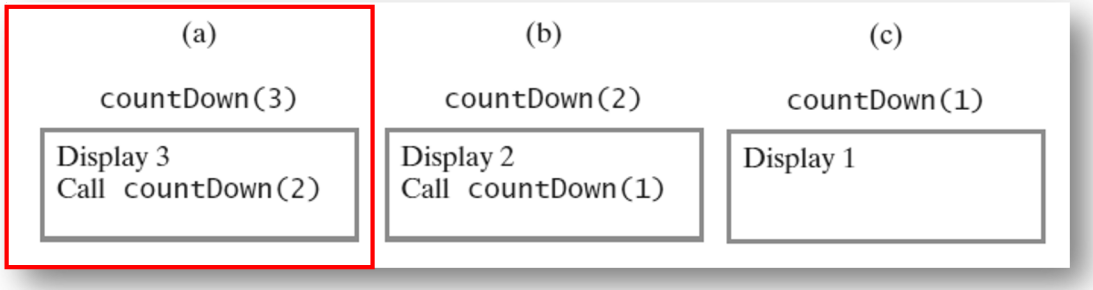
```
public static void countDown(int integer)
{
    System.out.println(integer);
    if (integer > 1) countDown(integer - 1);
}
```

- Suppose a client invokes method `countDown()` with the statement `countDown(3);`

Tracing a Recursive Method

- The argument 3 is copied into the parameter `integer` and the following statements of the method are executed:

```
System.out.println(3);
if (3 > 1)
  countDown(3 - 1);
```



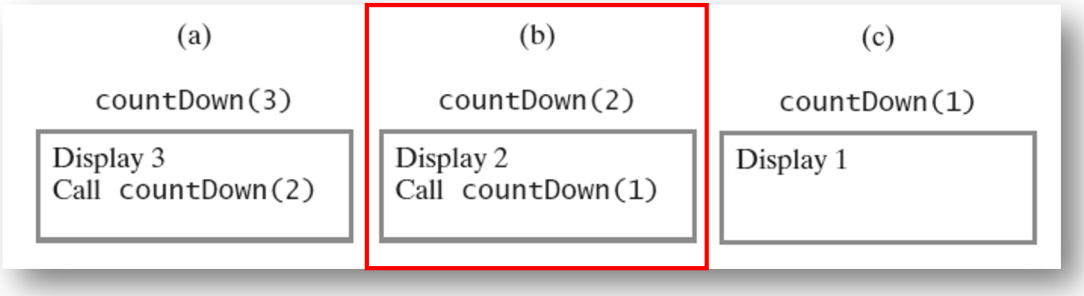
a) A line containing "3" is output and the recursive call to `countDown(2)` occurs

Tracing a Recursive Method

- Execution of the original call to `countDown(3)` is suspended until the result of `countDown(2)` is known (even though nothing happens after the recursive call)

- Continuing the trace, `countDown(2)` executes the following statements:

```
System.out.println(2);
if (2 > 1)
  countDown(2 - 1);
```

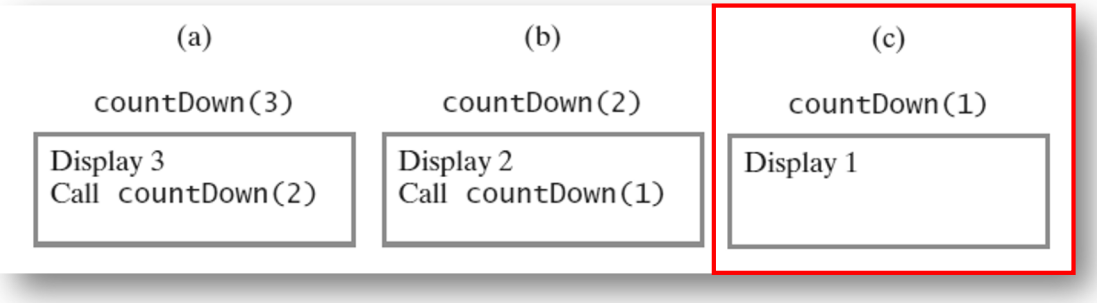


b) A line containing "2" is output and the recursive call to `countDown(1)` occurs

Tracing a Recursive Method

- Continuing the trace, `countDown(1)` executes the following statement:

```
System.out.println(1);
if (1 > 1)
```



c) A line containing "1" is output and no other recursive calls occur

Method Calls (Program Stack)

- For each call to a method Java records a snapshot of the current state of the method's execution, known as an activation record

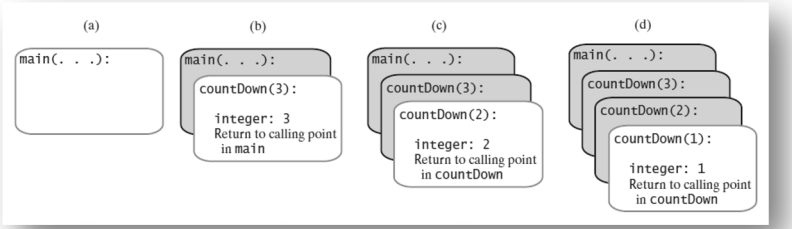
```
1     public static
      void main(string[] arg)
      {
         . . .
         int x = 5;
50       int y = methodA(x);
         . . .
      } // end main

100   public static
      int methodA(int a)
      {
         . . .
         int z = 2;
120      methodB(z);
         . . .
         return z;
      } // end methodA

150   public static
      void methodB(int b)
      {
         . . .
      } // end methodB
```

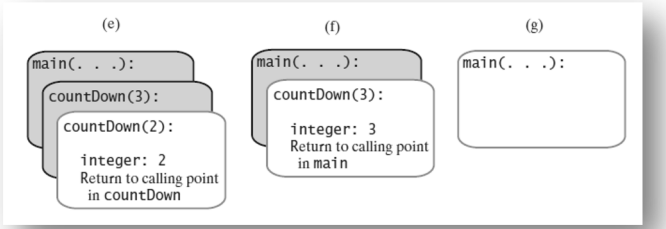
Program

- Activation record includes values of parameters, local variables, and location of current instruction
- Records are placed into the program stack (chronological organisation; currently executing method at the top)
- Program stack at three points:
  a) main begins execution
  b) methodA begins execution
  c) methodB begins execution

Program Stack and Recursive Methods

- Due to the program stack, Java can suspend the execution of a recursive method and invoke it again with new argument values
- Illustration of activation records during execution of `countDown(3)` from `main()`:





- A recursive method uses more memory than an iterative method as each recursive call generates an activation record
- Too many recursive calls can use all the memory available to the program stack, resulting in a *stack overflow* error!