Northeastern University

# Lecture 21: Recursion - 1

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5 © 2019 Pearson Education, Inc.

# What Is Recursion?

# What Is Recursion?

- Consider hiring a contractor to build
  - He hires a subcontractor for a portion of the job
  - That subcontractor hires a sub-subcontractor to do a smaller portion of job
- The last sub-sub- … subcontractor finishes
  - Each one finishes and reports “done” up the line

# Example: The Countdown

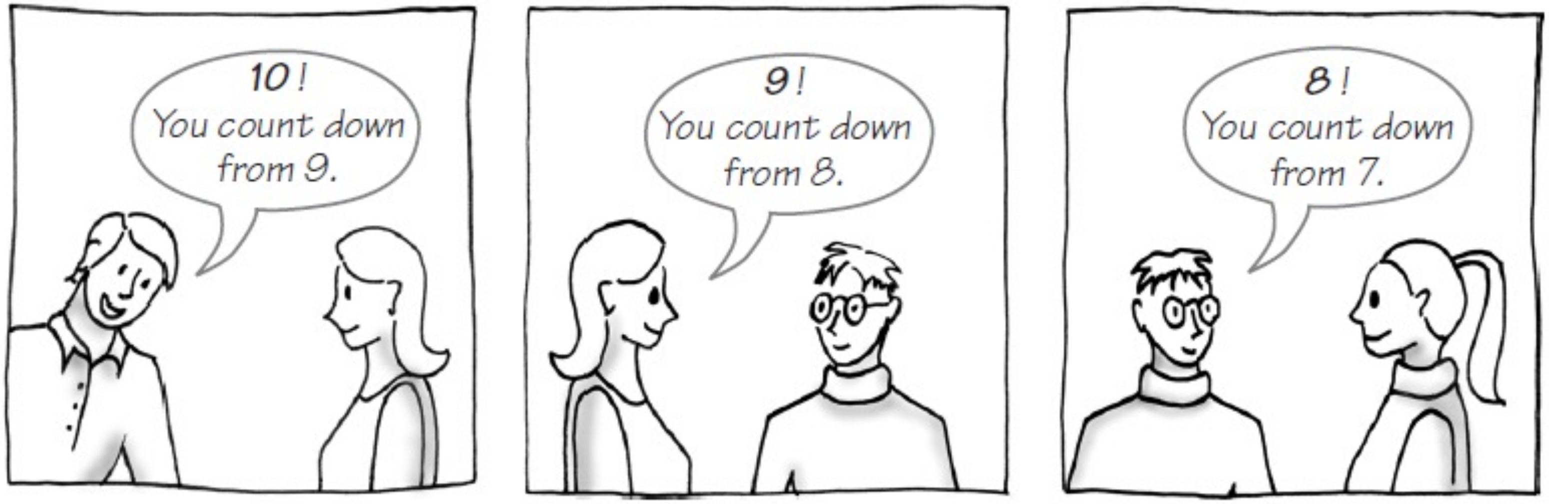

10! You count down from 9.
9! You count down from 8.
8! You count down from 7.

Figure 9-1: Counting down from 10

# Example: The Countdown

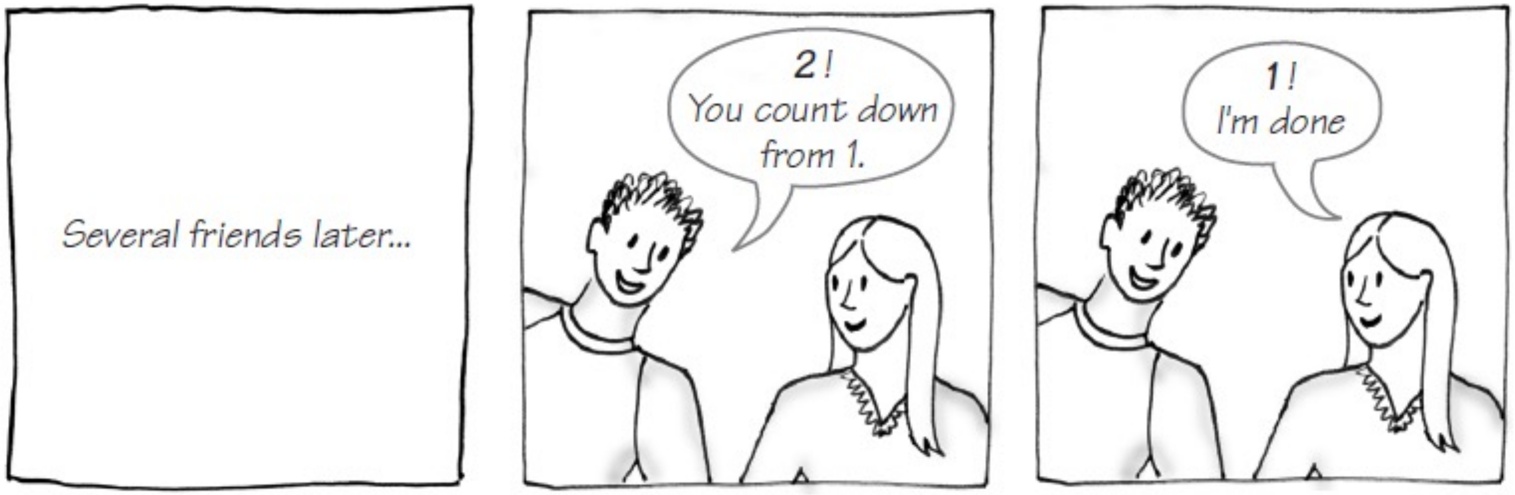


Figure 9-1: Counting down from 10

# Example: The Countdown

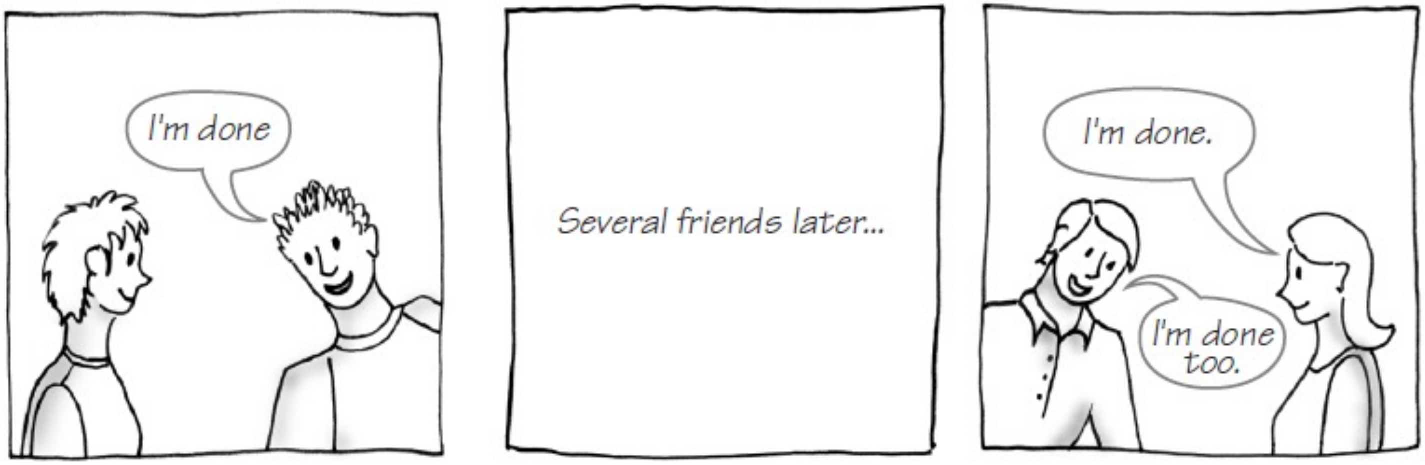


Figure 9-1: Counting down from 10

# 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
```

Recursive Java method to do countdown.

# Definition

- Recursion is a problem-solving process
  - » Breaks a problem into identical but smaller problems.
- A method that calls itself is a recursive method.
  - » The invocation is a recursive call or recursive invocation.

# Design Guidelines

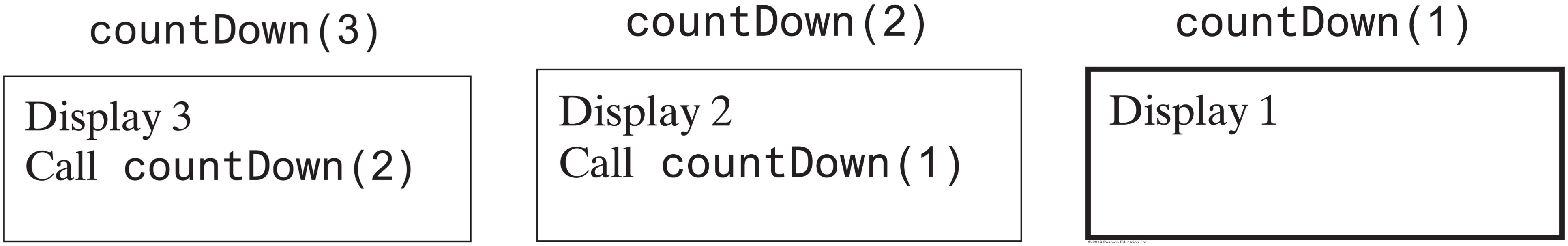
- Method must be given an input value.
- Method definition must contain logic that involves this input, leads to different cases.
- One or more cases should provide solution that does not require recursion.
  - » Otherwise, it is an infinite recursion
- One or more cases must include a recursive invocation.

# Programming Tip

- While iterative method contains a loop, recursive method calls itself.
- Some recursive methods contain a loop and call themselves.
  - » If the recursive method with loop uses `while`, make sure you did not mean to use an `if` statement.

# Tracing a Recursive Method

# Tracing a Recursive Method


countDown(3)

Display 3
Call countDown(2)

countDown(2)

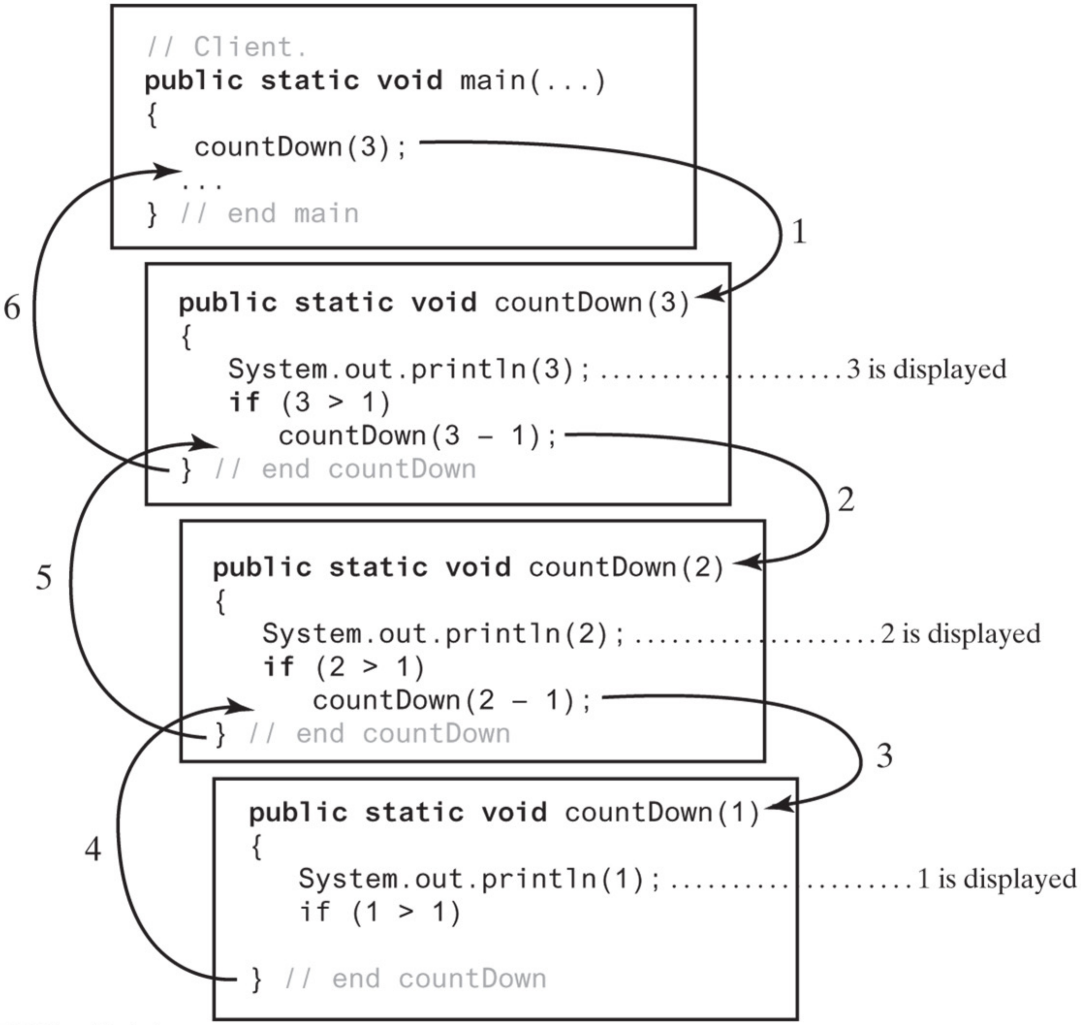
Display 2
Call countDown(1)

countDown(1)

Display 1

Figure 9-2: The effect of the method call `countDown(3)`

# Tracing a Recursive Method



```
// Client.
public static void main(...)
{
   countDown(3);
   ...
} // end main
```

```
public static void countDown(3)
{
   System.out.println(3);
   if (3 > 1)
      countDown(3 – 1);
} // end countDown
```

```
public static void countDown(2)
{
   System.out.println(2);
   if (2 > 1)
      countDown(2 – 1);
} // end countDown
```

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

} // end countDown
```

Figure 9-3: Tracing the execution of `countdown(3)`

# Tracing a Recursive Method

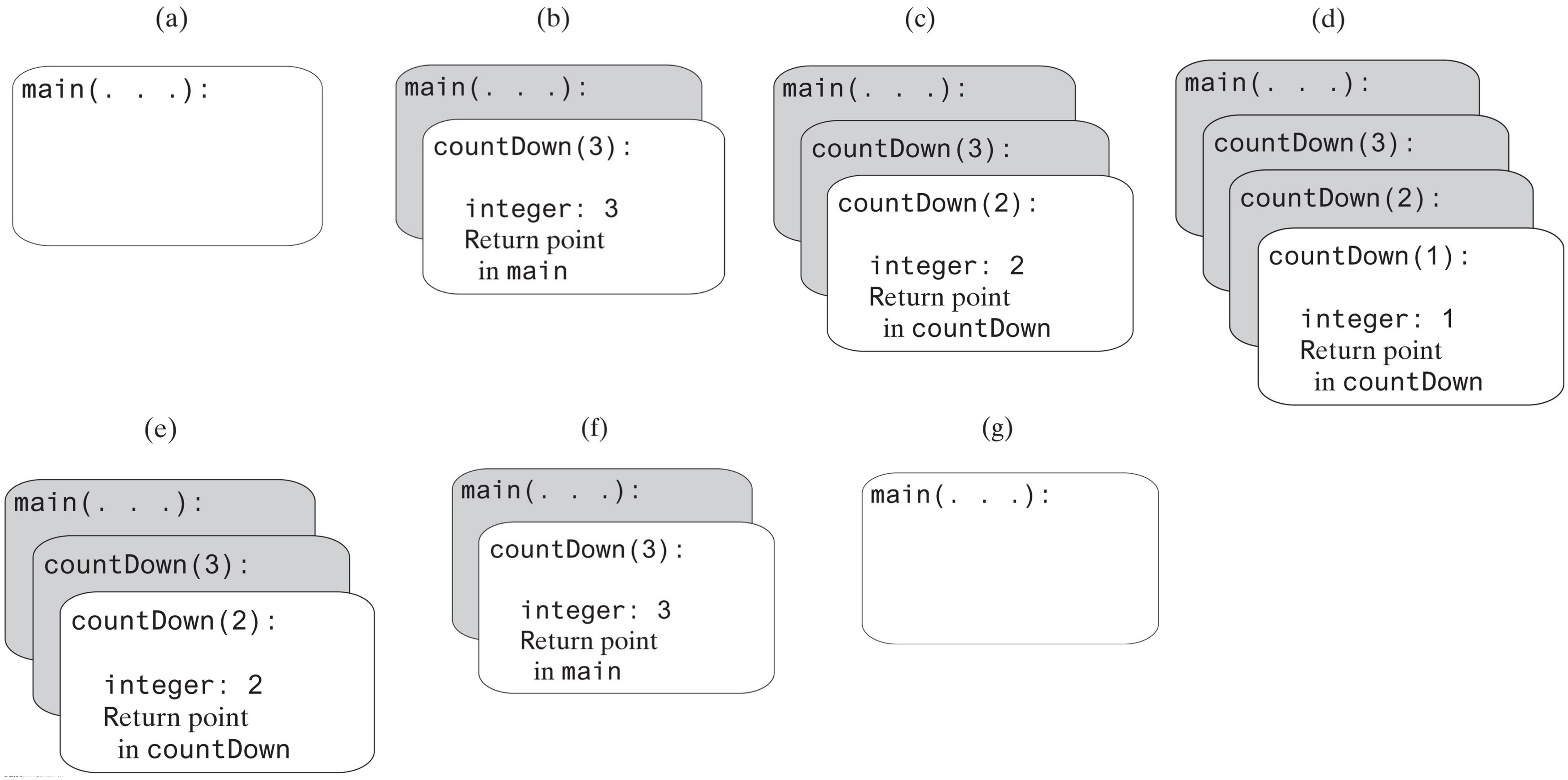


Figure 9-4: The stack of activation records during the execution of the call `countDown(3)`

# Stack of Activation Records

- Each call to a method generates an activation record.
- Recursive method uses more memory than an iterative method.
  - Each recursive call generates an activation record.
- If recursive call generates too many activation records, it could cause stack overflow.

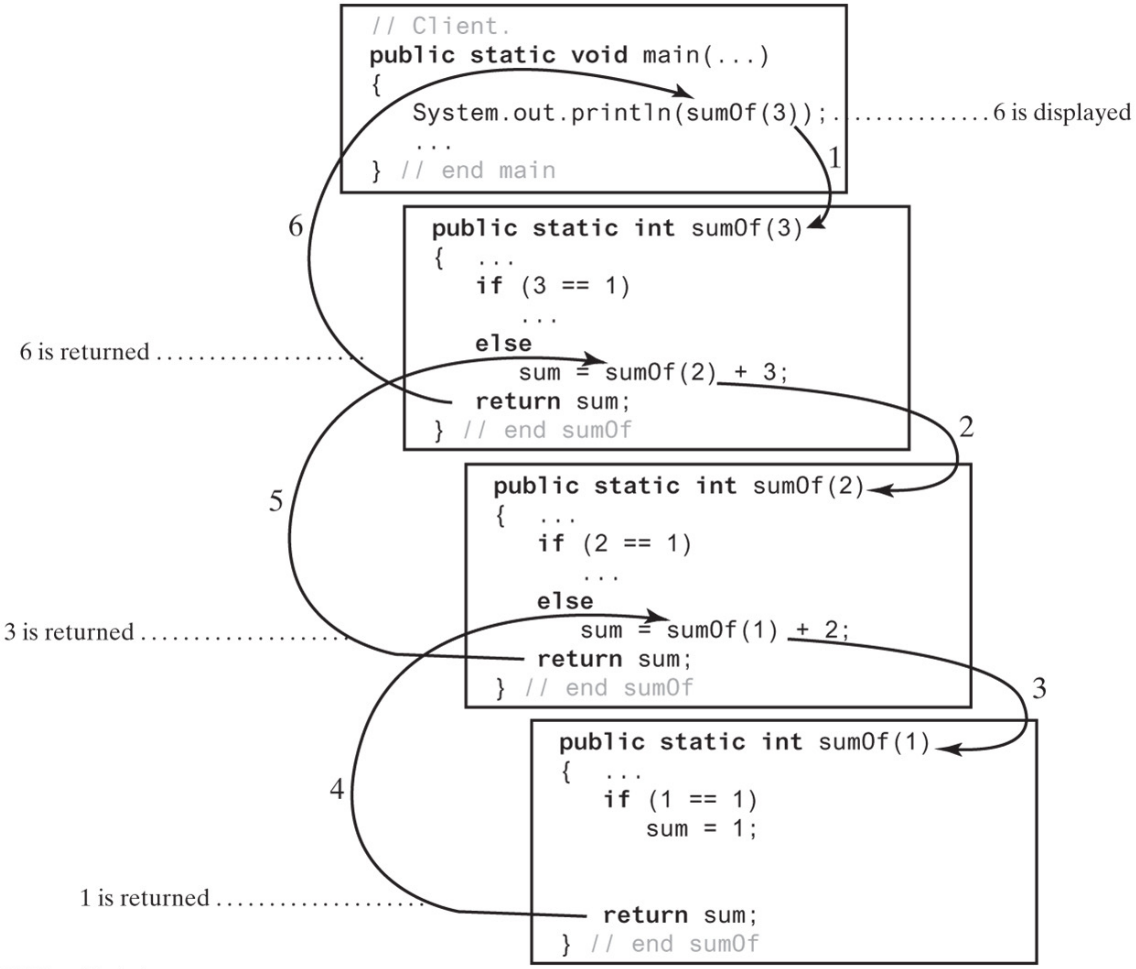
# Recursive Methods That Return a Value

```
/** @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 method to calculate $\sum_{i=1}^{n} i$

# Tracing a Recursive Method



```
// Client.
public static void main(...)
{
   System.out.println(sumOf(3));
   ...
} // end main
```

```
public static int sumOf(3)
{  ...
   if (3 == 1)
      ...
   else
      sum = sumOf(2) + 3;
   return sum;
} // end sumOf
```

```
public static int sumOf(2)
{  ...
   if (2 == 1)
      ...
   else
      sum = sumOf(1) + 2;
   return sum;
} // end sumOf
```

```
public static int sumOf(1)
{  ...
   if (1 == 1)
      sum = 1;


   return sum;
} // end sumOf
```

Figure 9-5: Tracing the execution of `sumOf(3)`

# Exercise (L21_E1)

- A child is running up a staircase with *n* steps, and can hop either 1 step, 2 steps, or 3 steps at a time.
- Implement a method to count how many possible ways the child can run up the stairs.

# Answer

```java
public class L21_E1 {

  public static void main(String[] args){
    //Assume the staircase has 20 steps
    int result = countWays(20);
    System.out.println("There are " + result + " ways");
  }

  private static int countWays(int n){
    if(n<0){
      return 0;
    }else if(n == 0){
      return 1;
    }else{
      return countWays(n-1)+countWays(n-2)+countWays(n-3);
    }
  }

}
```