Programming with Recursion

The Recursion Pattern (§ 2.5)
- **Recursion**: when a method calls itself
- Classic example - the factorial function:
  \[ n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n \]
- Recursive definition:
  \[
  f(n) = \begin{cases} 
  1 & \text{if } n = 0 \\
  n \cdot f(n-1) & \text{else}
  \end{cases}
  \]
- As a Java method:
  ```java
  // recursive factorial function
  public static int recursiveFactorial(int n) {
      if (n == 0) return 1; // basis case
      else return n * recursiveFactorial(n - 1); // recursive case
  }
  ```

Content of a Recursive Method
- **Base case(s)**.
  - 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.

Visualizing Recursion
- **Recursion trace**
  - A box for each recursive call
  - An arrow from each caller to callee
  - An arrow from each callee to caller showing return value

Example recursion trace:
```
recursiveFactorial(4)
| call |
recursiveFactorial(3)
| call |
recursiveFactorial(2)
| call |
recursiveFactorial(1)
| call |
recursiveFactorial(0)
| return 1 |
return 4 * 1 = 4
return 3 * 2 = 6
return 2 * 3 = 6
return 1 * 1 = 1
final answer 4 * 1 = 4
```
**Example – English Rulers**

Define a recursive way to print the ticks and numbers like an English ruler:

```java
// draw a tick with no label
public static void drawOneTick(int tickLength)
{
    // recursively draw left ticks
    drawOneTick(tickLength - 1);
    // draw center tick
    System.out.print(tickLength + " ");
    // recursively draw right ticks
    drawOneTick(tickLength - 1);
}

// draw one tick
public static void drawOneTick(int tickLength, int tickLabel)
{
    for (int i = 0; i < tickLength; i++)
        System.out.print("-");
    if (tickLabel > 0)
        System.out.print(tickLabel);
    System.out.print("\n");
}

public static void drawTicks(int tickLength)
{
    // draw ticks of given length
    if (tickLength > 0)
    {
        // stop when length drops to 0
        drawTicks(tickLength - 1);
        // recursively draw left ticks
        drawOneTick(tickLength);
        // draw center tick
        drawOneTick(tickLength - 1);
        // recursively draw right ticks
        drawTicks(tickLength - 1);
    }
}

public static void drawRuler(int nInches, int majorLength)
{
    // draw tick 0 and its label
    drawOneTick(majorLength, 0);
    // draw ticks for this inch
    for (int i = 1; i <= nInches; i++)
    {
        drawTicks(majorLength - 1);
        // draw tick i and its label
        drawOneTick(majorLength - 1, i);
    }
}
```

**Visualizing the DrawTicks Method**

An interval with a central tick length \( L \geq 1 \) is composed of the following:

- an interval with a central tick length \( L-1 \),
- a single tick of length \( L \),
- an interval with a central tick length \( L-1 \).