Recursion

• **Example 1:** Fibonacci Numbers
  1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

  ```java
  public static long fib(int n)
  {
      if (n <= 1)
          return n;
      else
          return fib(n-1) + fib(n-2);
  }
  ```

• **Example 2:** Towers of Hanoi
Recursion

• **Example 1:** Fibonacci Numbers
  1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

  ```java
  public static long fib(int n)
  {
    if (n <= 1)
      return n;
    else
      return fib(n-1) + fib(n-2);
  }
  ```

• **Example 2:** Towers of Hanoi
Figure 2.11
Recursive calls that \texttt{rabbit(7)} generates
Figure 2.19a and b

a) The initial state; b) move $n - 1$ disks from $A$ to $C$
Figure 2.19c and d

c) move one disk from A to B; d) move $n - 1$ disks from C to B
Sample output

Move top disk from pole A to pole B
Move top disk from pole A to pole C
Move top disk from pole B to pole C
Move top disk from pole A to pole B
Move top disk from pole C to pole A
Move top disk from pole C to pole B
Move top disk from pole A to pole B
public static void solveTowers(int count, char source, char destination, char spare) {
    if (count == 1) {
        System.out.println("Move top disk from pole " + source + " to pole " + destination);
    }
    else {
        solveTowers(count-1, source, spare, destination); // X
        solveTowers(1, source, destination, spare);       // Y
        solveTowers(count-1, spare, destination, source); // Z
    }  // end if
}  // end solveTowers
Figure 2.20
The order of recursive calls that results from $\text{solveTowers}(3, A, B, C)$
Figure 2.21a
Box trace of `solveTowers(3, 'A', 'B', 'C')`

The initial call 1 is made, and `solveTowers` begins execution:

```
count = 3
source = A
dest = B
spare = C
```

At point X, recursive call 2 is made, and the new invocation of the method begins execution:

```
count = 3
source = A
dest = B
spare = C
```

```
count = 2
source = A
dest = C
spare = B
```

At point X, recursive call 3 is made, and the new invocation of the method begins execution:

```
count = 3
source = A
dest = B
spare = C
```

```
count = 2
source = A
dest = C
spare = B
```

```
count = 1
source = A
dest = B
spare = C
```

This is the base case, so a disk is moved, the return is made, and the method continues execution.
Figure 2.21b
Box trace of $solveTowers(3, 'A', 'B', 'C')$

At point Y, recursive call 4 is made, and the new invocation of the method begins execution:

\[
\begin{array}{c}
\text{count} = 3 \\
\text{source} = A \\
\text{dest} = B \\
\text{spare} = C
\end{array} \xrightarrow{X} \begin{array}{c}
\text{count} = 2 \\
\text{source} = A \\
\text{dest} = C \\
\text{spare} = D
\end{array} \xrightarrow{Y} \begin{array}{c}
\text{count} = 1 \\
\text{source} = A \\
\text{dest} = C \\
\text{spare} = D
\end{array}
\]

This is the base case, so a disk is moved, the return is made, and the method continues execution.

At point Z, recursive call 5 is made, and the new invocation of the method begins execution:

\[
\begin{array}{c}
\text{count} = 3 \\
\text{source} = A \\
\text{dest} = B \\
\text{spare} = C
\end{array} \xrightarrow{X} \begin{array}{c}
\text{count} = 2 \\
\text{source} = A \\
\text{dest} = C \\
\text{spare} = B
\end{array} \xrightarrow{Z} \begin{array}{c}
\text{count} = 1 \\
\text{source} = B \\
\text{dest} = C \\
\text{spare} = A
\end{array}
\]

This is the base case, so a disk is moved, the return is made, and the method continues execution.
Figure 2.21c
Box trace of $solveTowers(3, \text{`A'}, \text{`B'}, \text{`C'})$

This invocation completes, the return is made, and the method continues execution.

\begin{verbatim}
count = 3
source = A
dest = B
spare = C
\end{verbatim} \hspace{0.5cm}
\begin{verbatim}
count = 2
source = A
dest = C
spare = B
\end{verbatim} \hspace{0.5cm}
\begin{verbatim}
count = 1
source = B
dest = C
spare = A
\end{verbatim}

At point Y, recursive call 6 is made, and the new invocation of the method begins execution:

\begin{verbatim}
count = 3
source = A
dest = B
spare = C
\end{verbatim} \hspace{0.5cm} \begin{verbatim}
count = 1
source = A
dest = B
spare = C
\end{verbatim}

This is the base case, so a disk is moved, the return is made, and the method continues execution.

\begin{verbatim}
count = 3
source = A
dest = B
spare = C
\end{verbatim} \hspace{0.5cm} \begin{verbatim}
count = 1
source = A
dest = B
spare = C
\end{verbatim}

At point Z, recursive call 7 is made, and the new invocation of the method begins execution:

\begin{verbatim}
count = 3
source = A
dest = R
spare = C
\end{verbatim} \hspace{0.5cm} \begin{verbatim}
count = 2
source = C
dest = R
spare = A
\end{verbatim}
Figure 2.21d
Box trace of \texttt{solveTowers(3, 'A', 'B', 'C')}.

At point X, recursive call 8 is made, and the new invocation of the method begins execution:

At point Y, recursive call 9 is made, and the new invocation of the method begins execution:

This is the base case, so a disk is moved, the return is made, and the method continues execution.
Figure 2.21e
Box trace of $\text{solveTowers}(3, \ 'A', \ 'B', \ 'C')$

At point $Z$, recursive call 10 is made, and the new invocation of the method begins execution:

```
<table>
<thead>
<tr>
<th>count</th>
<th>source</th>
<th>dest</th>
<th>spare</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>
```

This is the base case, so a disk is moved, the return is made, and the method continues execution.

```
<table>
<thead>
<tr>
<th>count</th>
<th>source</th>
<th>dest</th>
<th>spare</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>
```

This invocation completes, the return is made, and the method continues execution.

```
<table>
<thead>
<tr>
<th>count</th>
<th>source</th>
<th>dest</th>
<th>spare</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>count</th>
<th>source</th>
<th>dest</th>
<th>spare</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>C</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>count</th>
<th>source</th>
<th>dest</th>
<th>spare</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>count</th>
<th>source</th>
<th>dest</th>
<th>spare</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>count</th>
<th>source</th>
<th>dest</th>
<th>spare</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>C</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>count</th>
<th>source</th>
<th>dest</th>
<th>spare</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>
```

02/05/04
Lecture 8