CSC115 Lecture 8

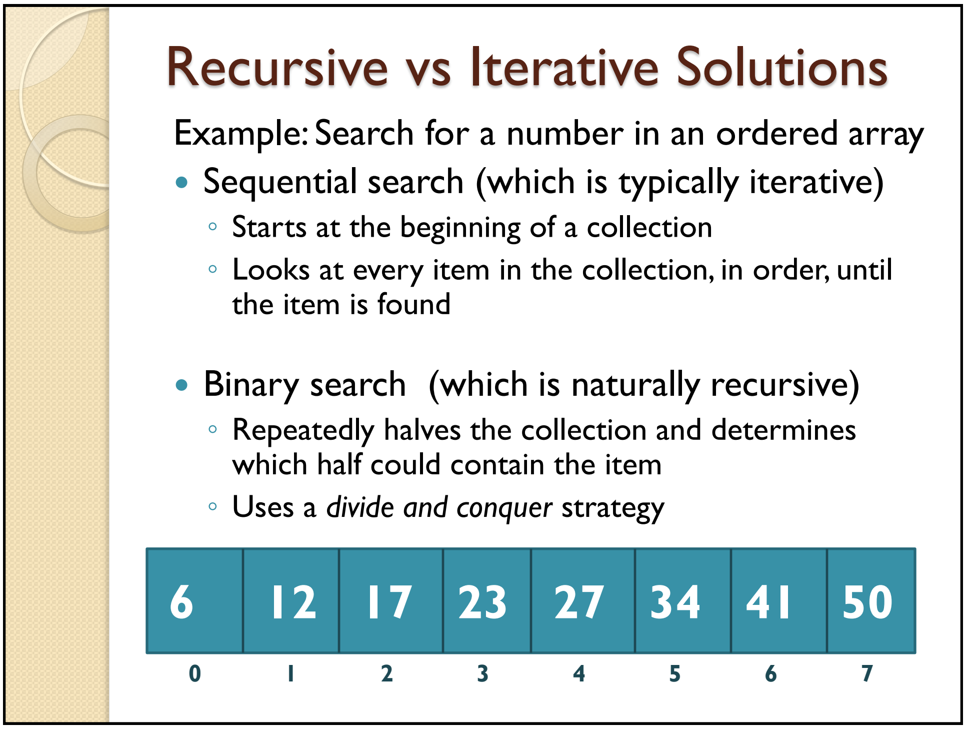
Chapter 3

Recursion:The Mirrors

Recursive Solutions

- Recursion
  - An extremely powerful problem-solving technique
  - Breaks a problem in smaller identical problems
- Two things to watch for in today's class:

	Iterative	Recursive
Static Methods	Iterative Static Methods	Recursive Static Methods
Instance Methods	Iterative Instance Methods	Recursive Instance Methods



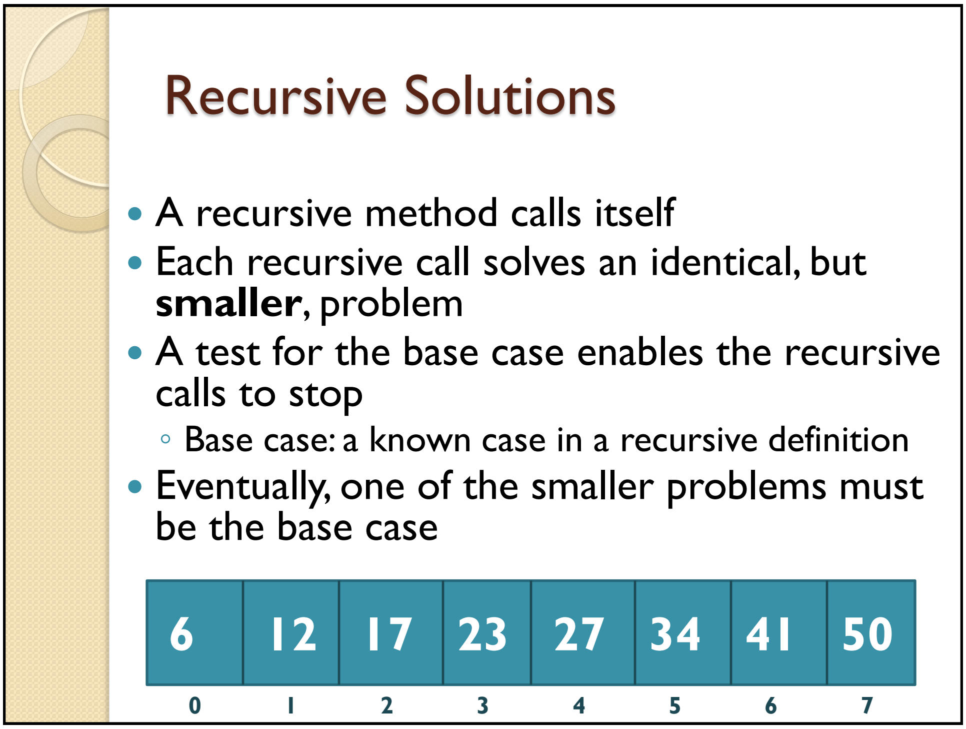
Recursive vs Iterative Solutions

Example: Search for a number in an ordered array

- Sequential search (which is typically iterative)
  - Starts at the beginning of a collection
  - Looks at every item in the collection, in order, until the item is found

- Binary search (which is naturally recursive)
  - Repeatedly halves the collection and determines which half could contain the item
  - Uses a *divide and conquer* strategy

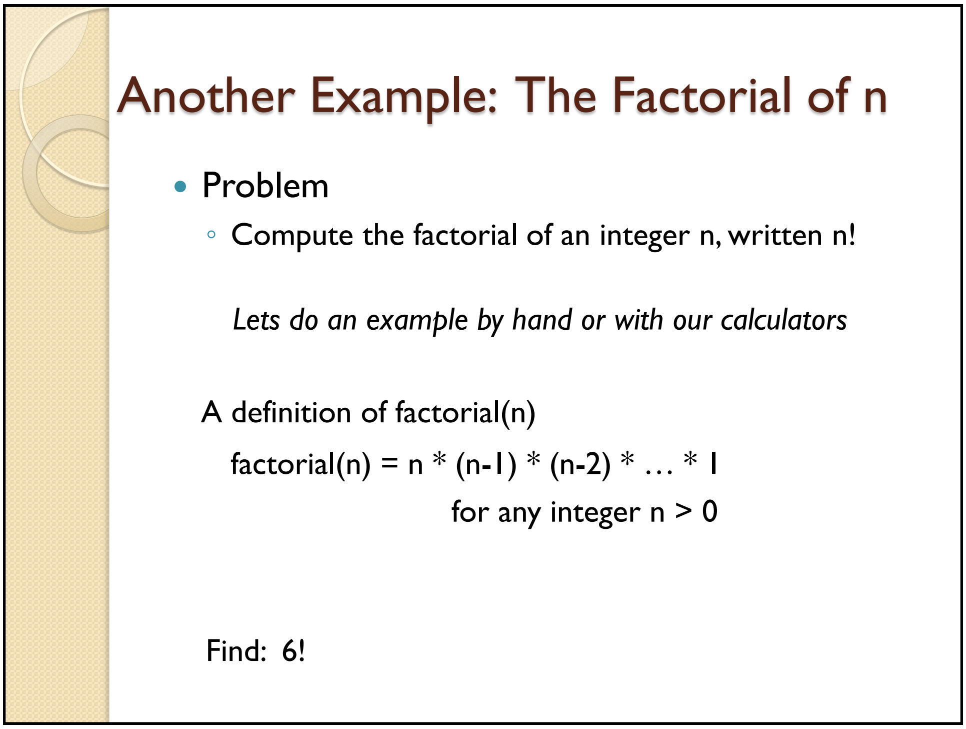
6	12	17	23	27	34	41	50
0	1	2	3	4	5	6	7



Recursive Solutions

- A recursive method calls itself
- Each recursive call solves an identical, but **smaller**, problem
- A test for the base case enables the recursive calls to stop
  - Base case: a known case in a recursive definition
- Eventually, one of the smaller problems must be the base case

6	12	17	23	27	34	41	50
0	1	2	3	4	5	6	7



Another Example: The Factorial of n

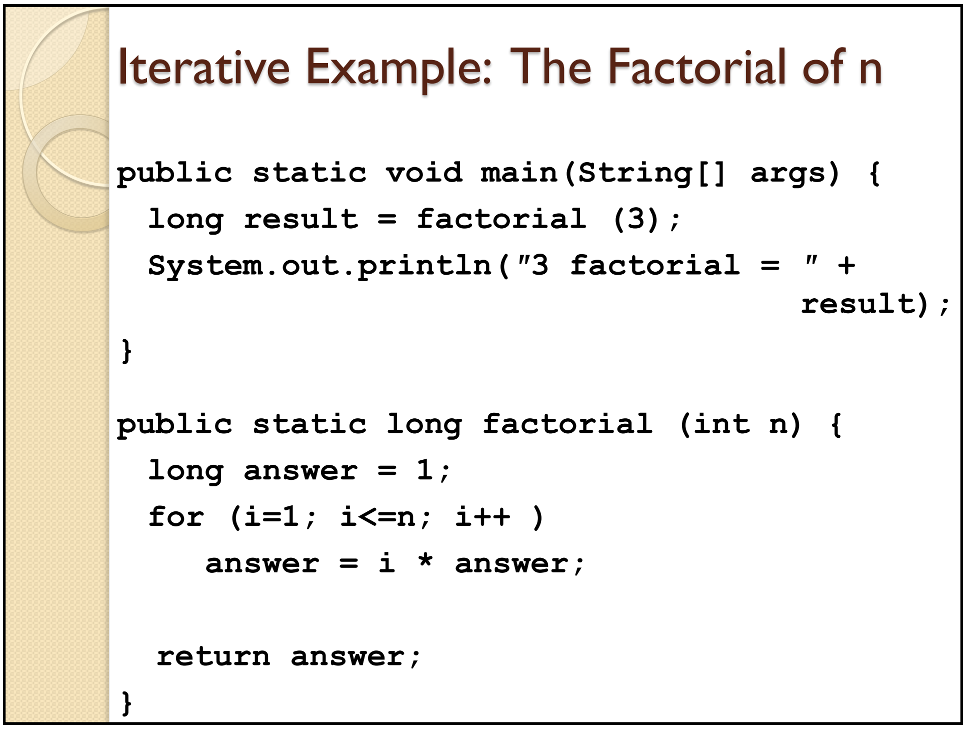
- Problem
  - Compute the factorial of an integer n, written n!

    Lets do an example by hand or with our calculators

A definition of factorial(n)

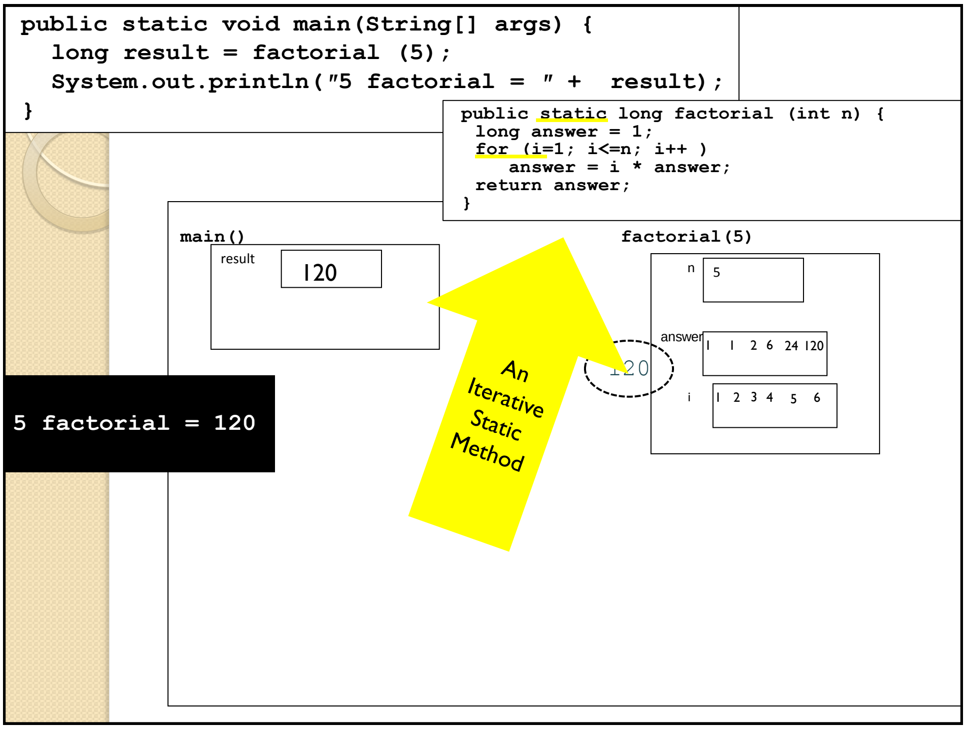
factorial(n) = n * (n-1) * (n-2) * … * 1
for any integer n > 0

Find: 6!



Iterative Example: The Factorial of n

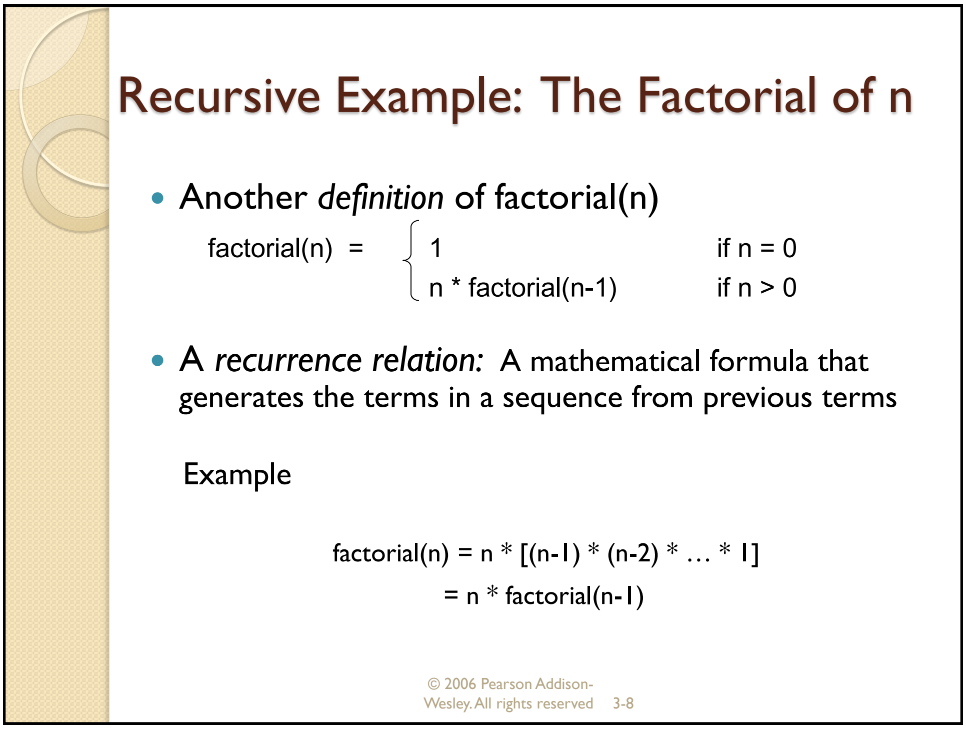
```
public static void main(String[] args) {
  long result = factorial (3);
  System.out.println("3 factorial = " +
                                          result);
}

public static long factorial (int n) {
  long answer = 1;
  for (i=1; i<=n; i++ )
     answer = i * answer;

   return answer;
}
```



```
public static void main(String[] args) {
   long result = factorial (5);
   System.out.println("5 factorial = " +  result);
}
```

```
public static long factorial (int n) {
  long answer = 1;
  for (i=1; i<=n; i++ )
     answer = i * answer;
  return answer;
}
```

```
5 factorial = 120
```



Recursive Example: The Factorial of n

- Another *definition* of factorial(n)

$$\text{factorial(n)} = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial(n-1)} & \text{if } n > 0 \end{cases}$$

- A *recurrence relation:* A mathematical formula that generates the terms in a sequence from previous terms

Example

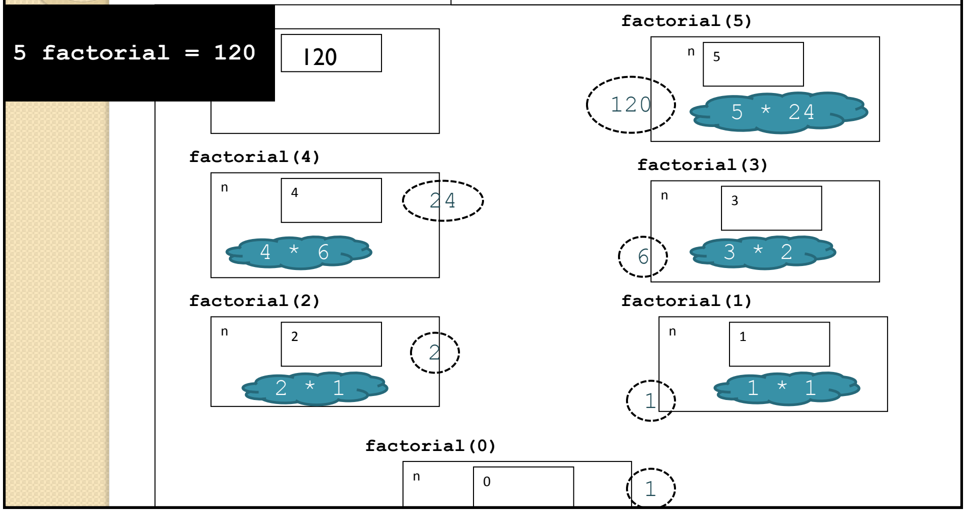
$$\text{factorial(n)} = n * [(n-1) * (n-2) * \ldots * 1]$$
$$= n * \text{factorial(n-1)}$$

Recursive Example: The Factorial of n

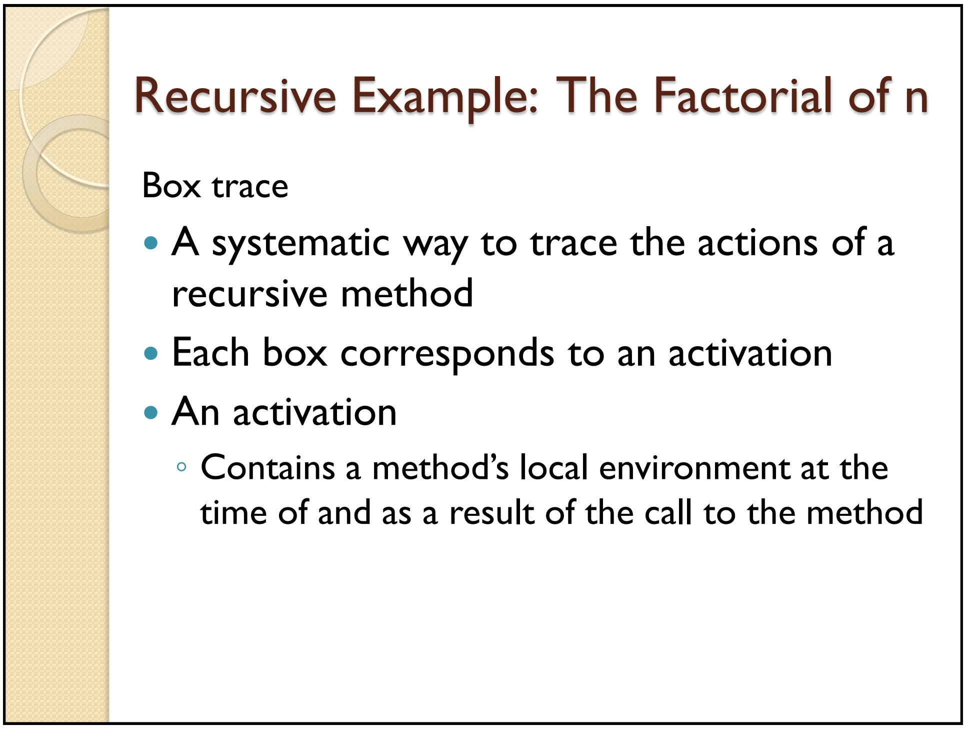
```
public static void main(String[] args) {
  long result = factorial (3);
  System.out.println("3 factorial =  " +
                                         result);
}

public static long factorial (int n) {
  long factorialResult;
  if (n == 0) factorialResult = 1;
  else factorialResult = n* factorial (n-1);
   return factorialResult;
}
```

```
public static void main(String[] args) {
   long result = factorial (5);
   System.out.println("5 factorial =  " +  result);
}
```

```
public static long factorial (int n) {
   if (n == 0) return 1;
   else return n* factorial (n-1);
}
```



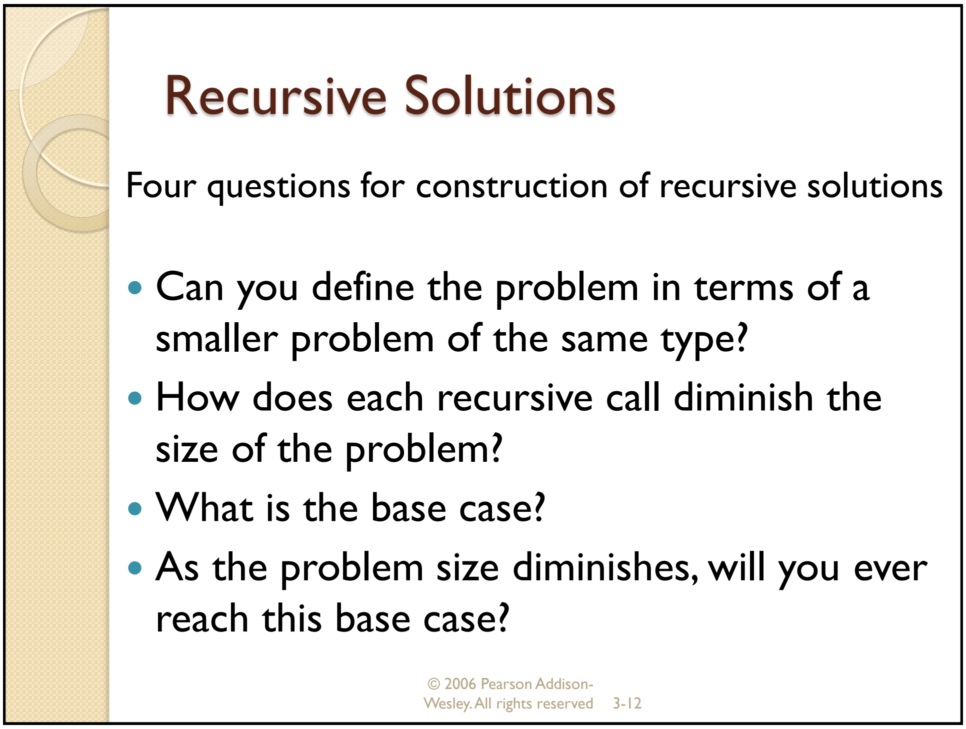
```
5 factorial = 120
```



Recursive Example: The Factorial of n

Box trace

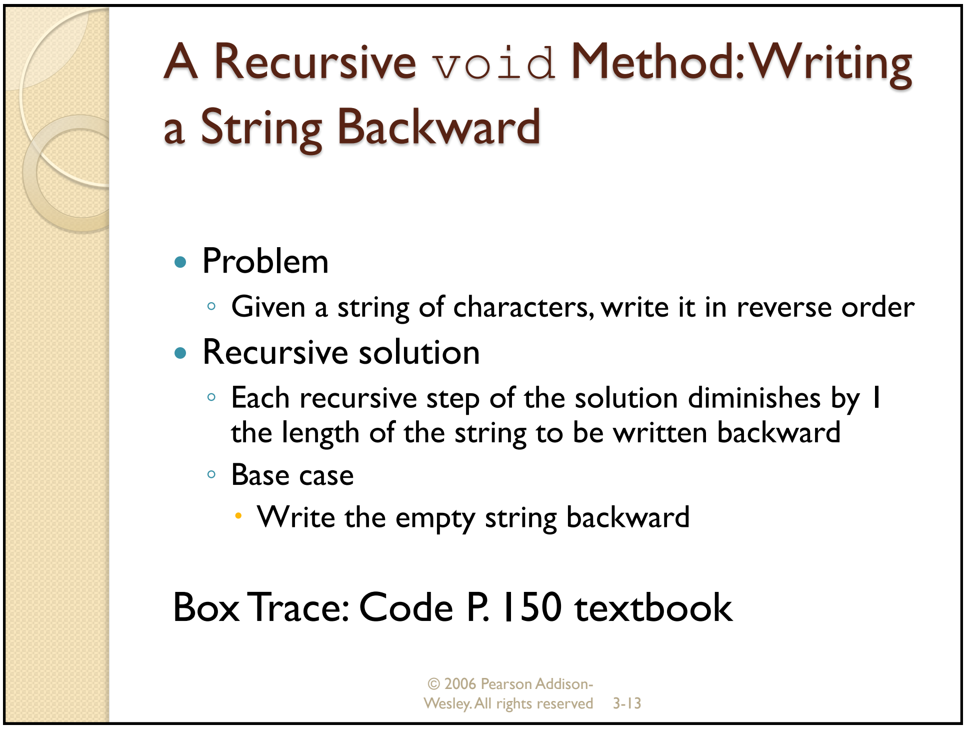
- A systematic way to trace the actions of a recursive method
- Each box corresponds to an activation
- An activation
  - Contains a method's local environment at the time of and as a result of the call to the method



Recursive Solutions

Four questions for construction of recursive solutions

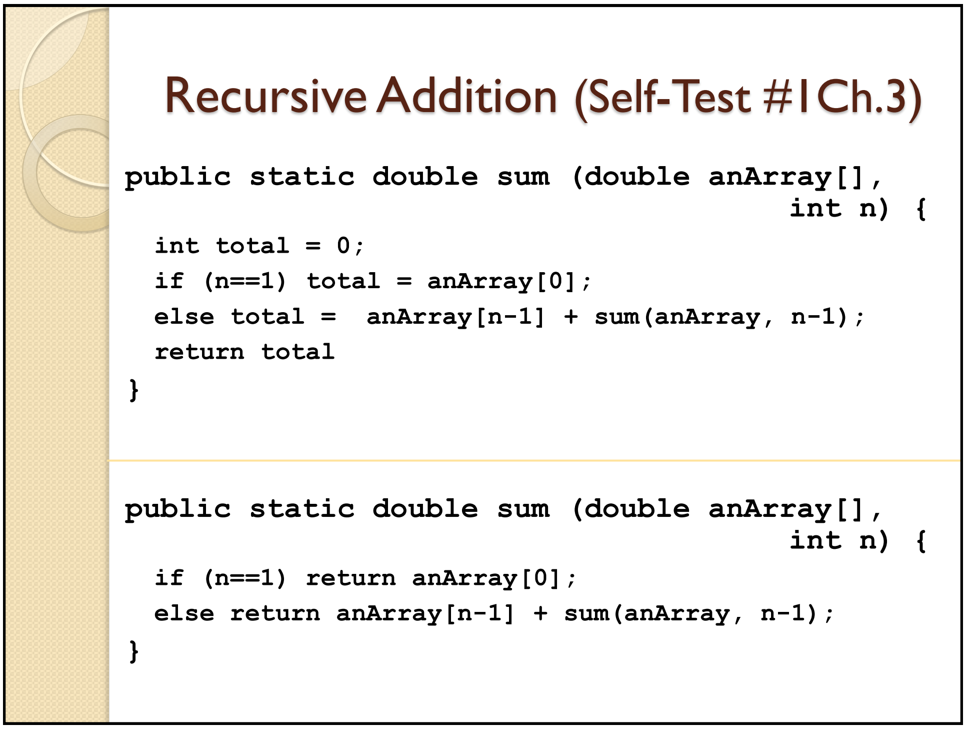
- Can you define the problem in terms of a smaller problem of the same type?
- How does each recursive call diminish the size of the problem?
- What is the base case?
- As the problem size diminishes, will you ever reach this base case?



A Recursive `void` Method: Writing a String Backward

- Problem
  - Given a string of characters, write it in reverse order
- Recursive solution
  - Each recursive step of the solution diminishes by 1 the length of the string to be written backward
  - Base case
    - Write the empty string backward

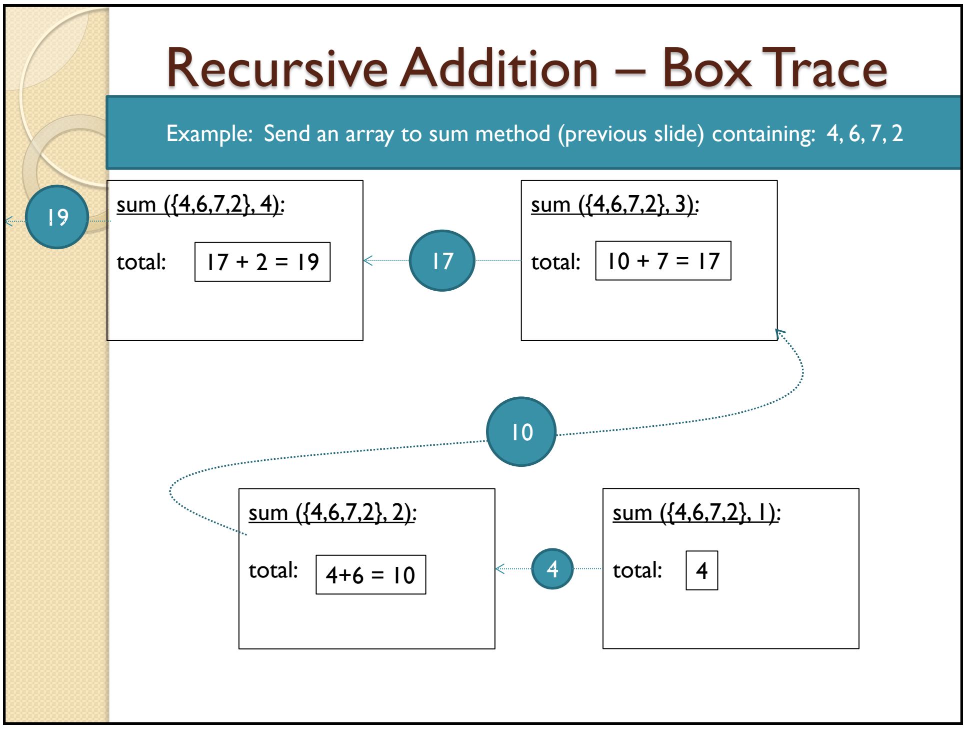
Box Trace: Code P. 150 textbook



Recursive Addition (Self-Test #1 Ch.3)

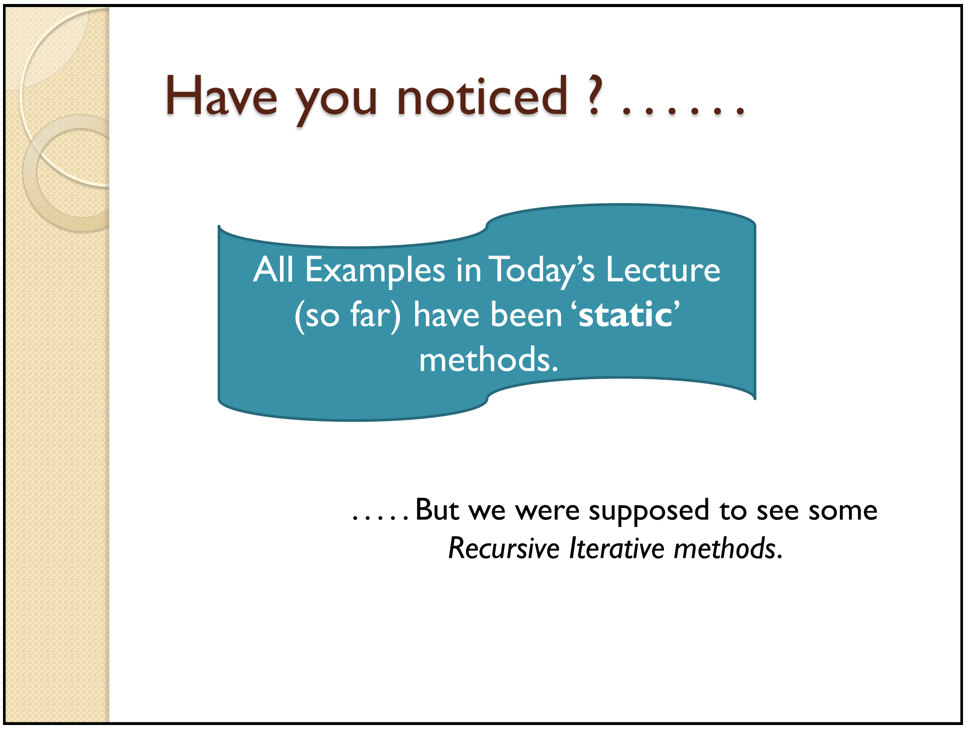
```
public static double sum (double anArray[],
                                      int n)  {
  int total = 0;
  if (n==1) total = anArray[0];
  else total =  anArray[n-1] + sum(anArray, n-1);
  return total
}
```

```
public static double sum (double anArray[],
                                      int n)  {
  if (n==1) return anArray[0];
  else return anArray[n-1] + sum(anArray, n-1);
}
```



Recursive Addition – Box Trace

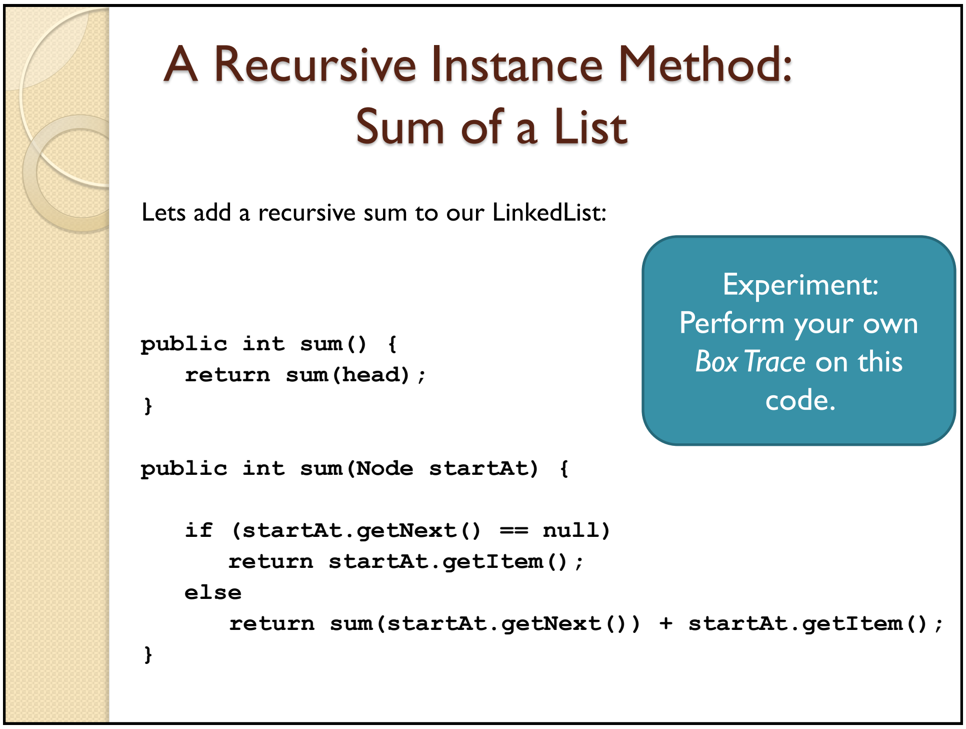
Example: Send an array to sum method (previous slide) containing: 4, 6, 7, 2



Have you noticed ? ……

All Examples in Today's Lecture (so far) have been '**static**' methods.

….. But we were supposed to see some *Recursive Iterative methods*.



A Recursive Instance Method: Sum of a List

Lets add a recursive sum to our LinkedList:

Experiment: Perform your own *Box Trace* on this code.

```
public int sum() {
   return sum(head);
}

public int sum(Node startAt) {

   if (startAt.getNext() == null)
      return startAt.getItem();
   else
      return sum(startAt.getNext()) + startAt.getItem();
}
```