CS2400 - Data Structures and Advanced Programming
Module 7: Recursions

Hao Ji

Computer Science Department

Cal Poly Pomona

Recursion

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

- A method that calls itself is a **recursive method**.

Recursion

- **Recursion** is a problem-solving process that breaks a problem into identical but smaller problems.
- A method that calls itself is a **recursive method**.
- Two problem-solving processes involve repetition; they are called **iteration** and **recursion**.

Iterative method contains a loop
Recursive method calls itself

Recursion

- (Example) Recursive Java method to do countDown.

```
/** 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
```

Recursion

- (Example) Recursive Java method to do countDown.

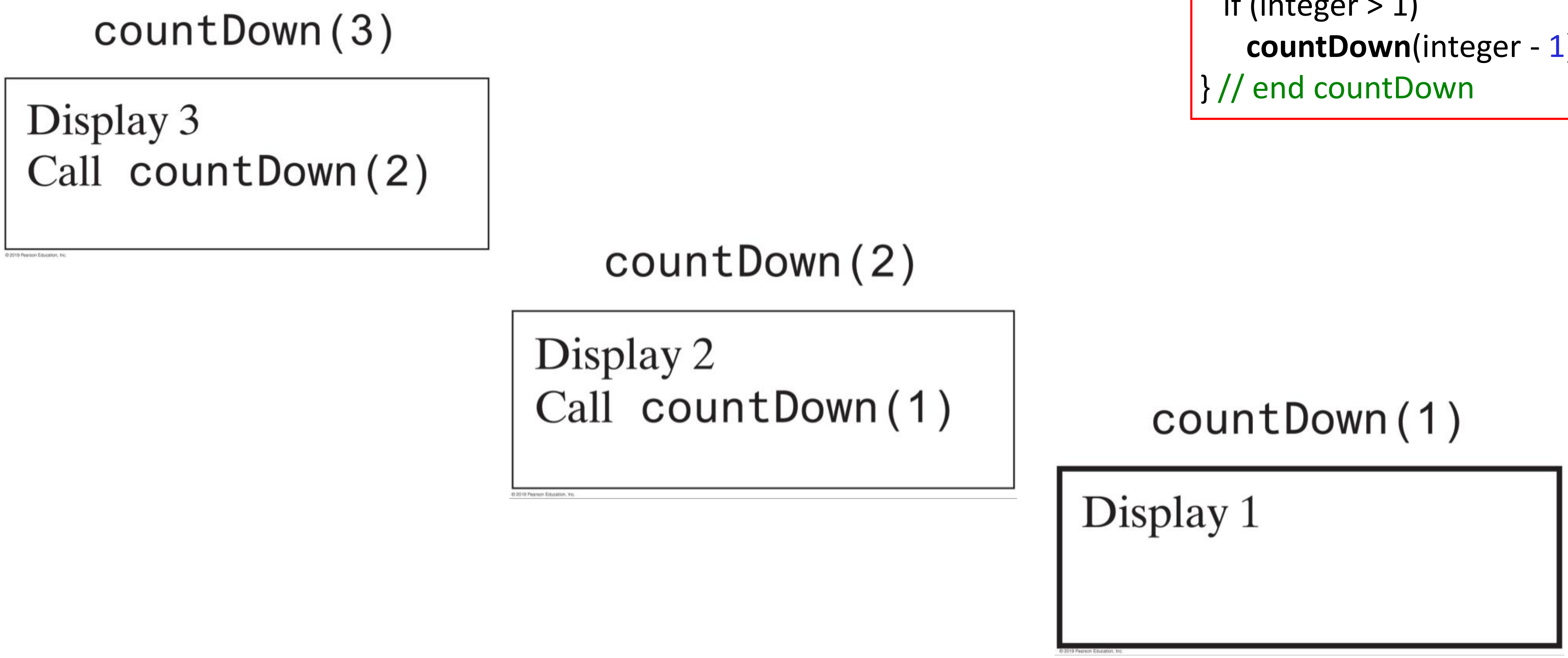
```
/** Counts down from a given positive integer.
    @param integer  An integer > 0.
*/
public static void countDown(int integer)
{
   System.out.println(integer);
   if (integer > 1)
      countDown(integer - 1);
} // end countDown
```

One or more cases should provide solution that does not require recursion; Infinite recursion, otherwise.

Tracing a Recursive Method

- The effect of the method call `countDown(3)`

```
/** 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
```



countDown(3)

Display 3
Call countDown(2)

countDown(2)

Display 2
Call countDown(1)

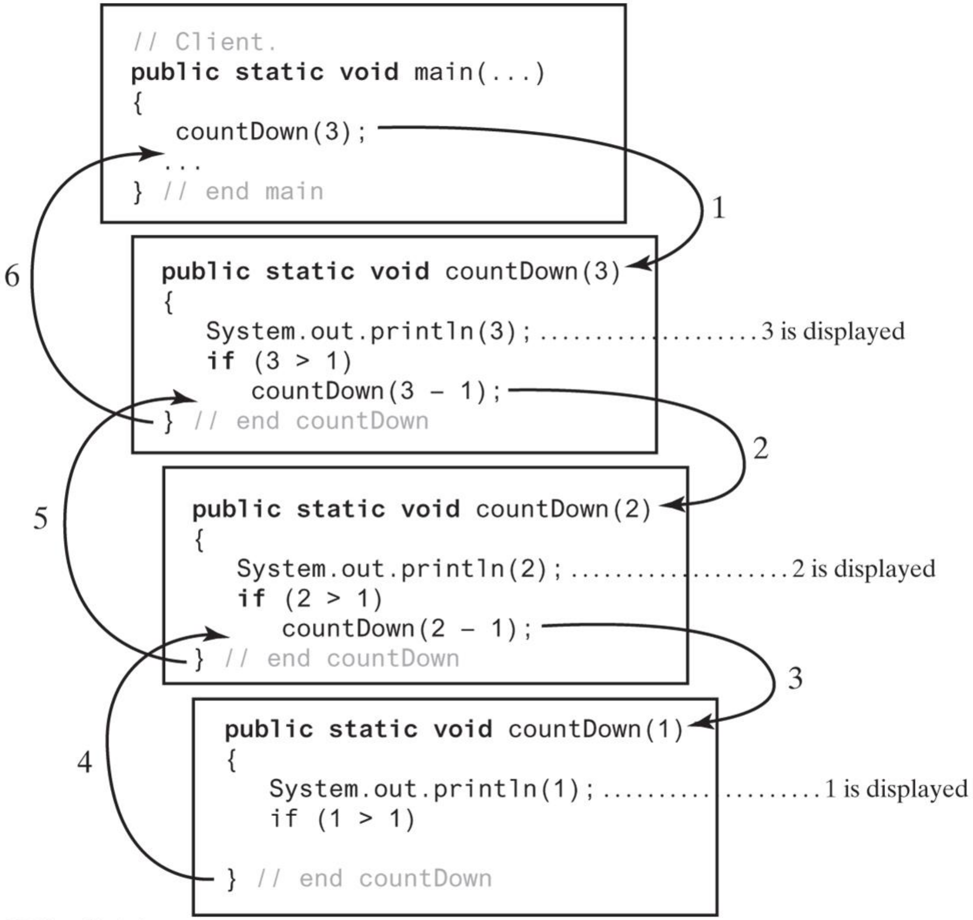
countDown(1)

Display 1

© 2019 Pearson Education, Inc.

Tracing a Recursive Method

- The effect of the method call `countDown(3)`



```
// 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
```

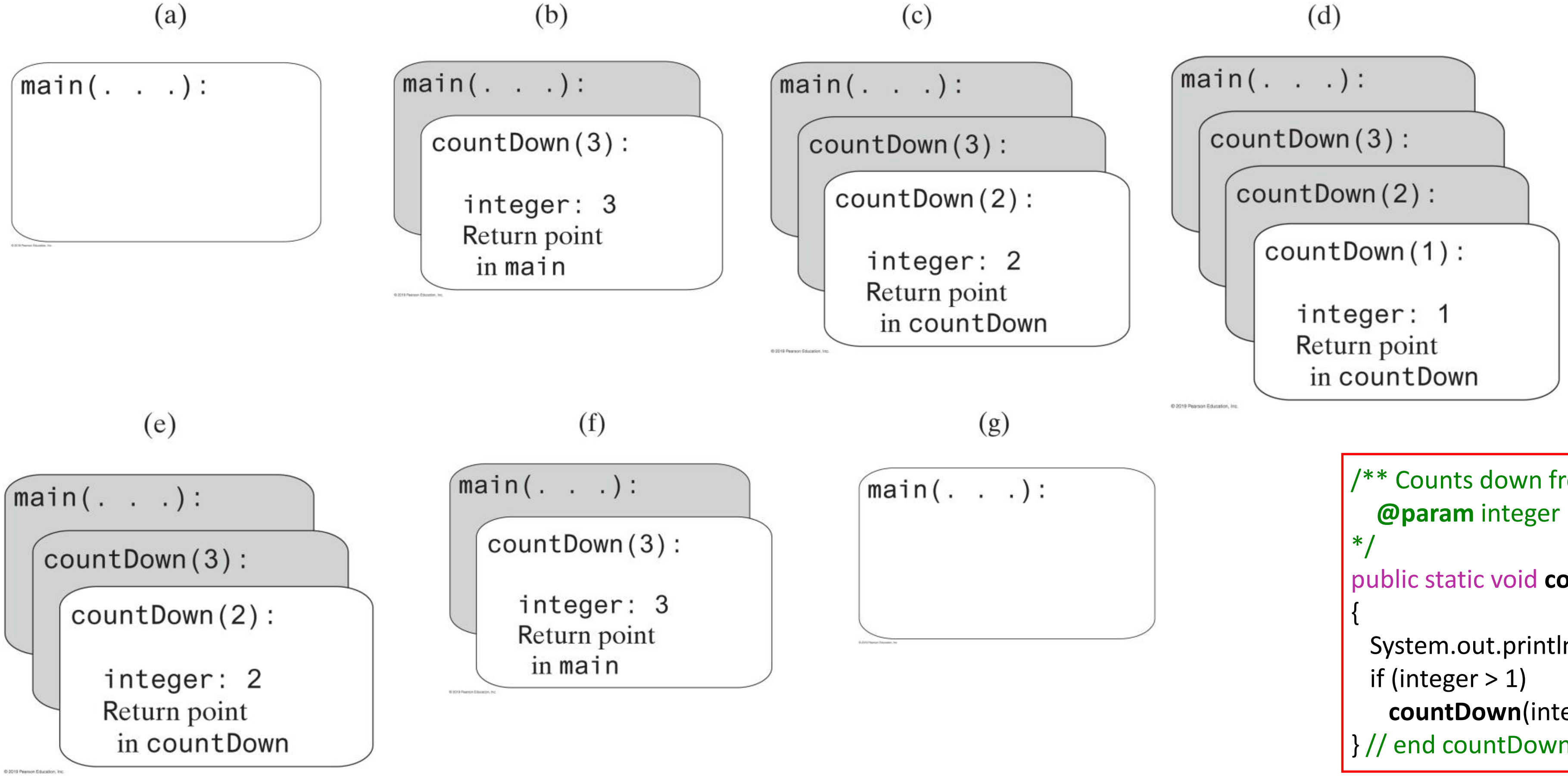
```
public static void countDown(2)
{
   System.out.println(2);
   if (2 > 1)
      countDown(2 – 1);
} // end countDown
```

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

} // end countDown
```

```
/** 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
```

Stack of Activation Records

- Each call to a method generates an activation record



```
/** 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
```

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, could cause stack overflow

Recursive Methods That Return a Value

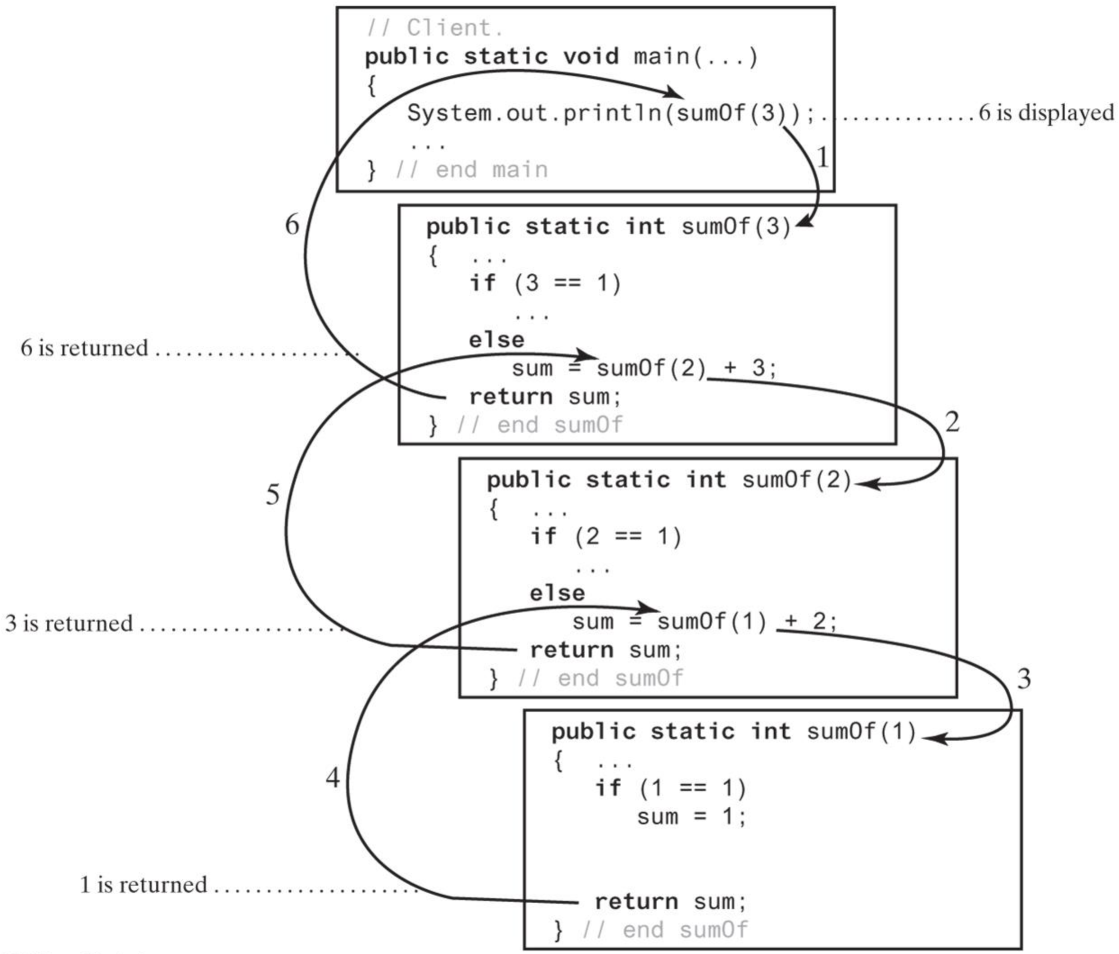
- Recursive method to calculate $\sum_{i=1}^{n} i$

```
/** @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 Methods That Return a Value

- Recursive method to calculate $\sum_{i=1}^{n} i$

```
/** @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
```



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

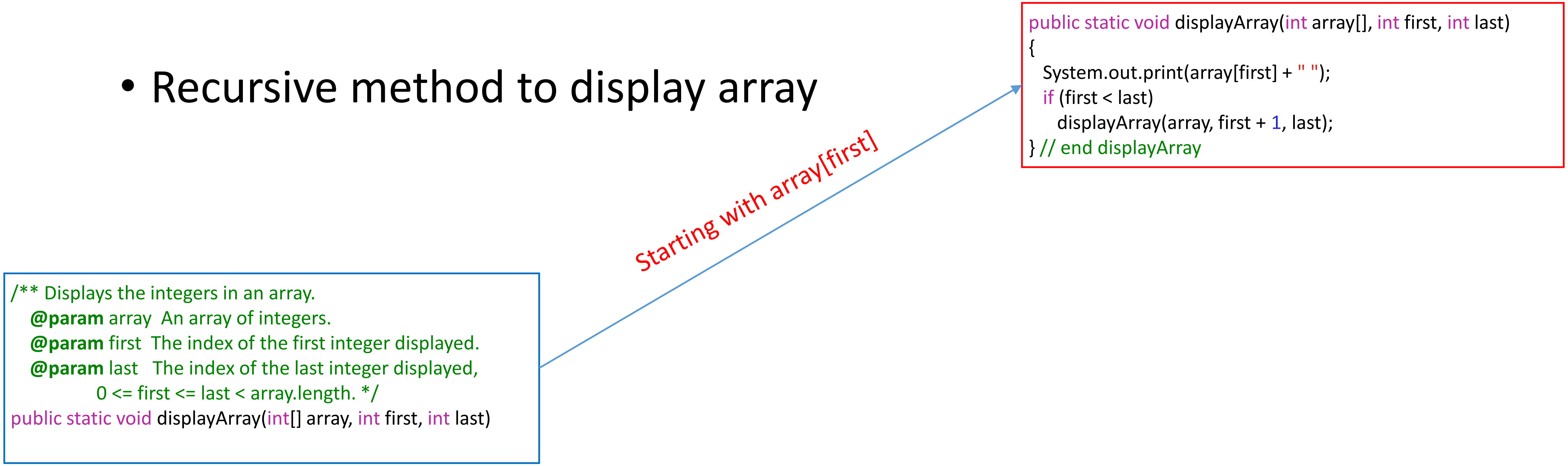
```
public static int sumOf(3)
{  ...
   if (3 == 1)
      ...
   else
      sum = sumOf(2) + 3;
   return sum;
} // end sumOf
```

```
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
```

Recursively Processing an Array

- Recursive method to display array



```
/** Displays the integers in an array.
   @param array  An array of integers.
   @param first  The index of the first integer displayed.
   @param last   The index of the last integer displayed,
            0 <= first <= last < array.length. */
public static void displayArray(int[] array, int first, int last)
```

Starting with array[first]

```
public static void displayArray(int array[], int first, int last)
{
  System.out.print(array[first] + " ");
  if (first < last)
    displayArray(array, first + 1, last);
} // end displayArray
```

Recursively Processing an Array

- Recursive method to display array

```
/** Displays the integers in an array.
   @param array  An array of integers.
   @param first  The index of the first integer displayed.
   @param last   The index of the last integer displayed,
            0 <= first <= last < array.length. */
public static void displayArray(int[] array, int first, int last)
```

Starting with array[first]

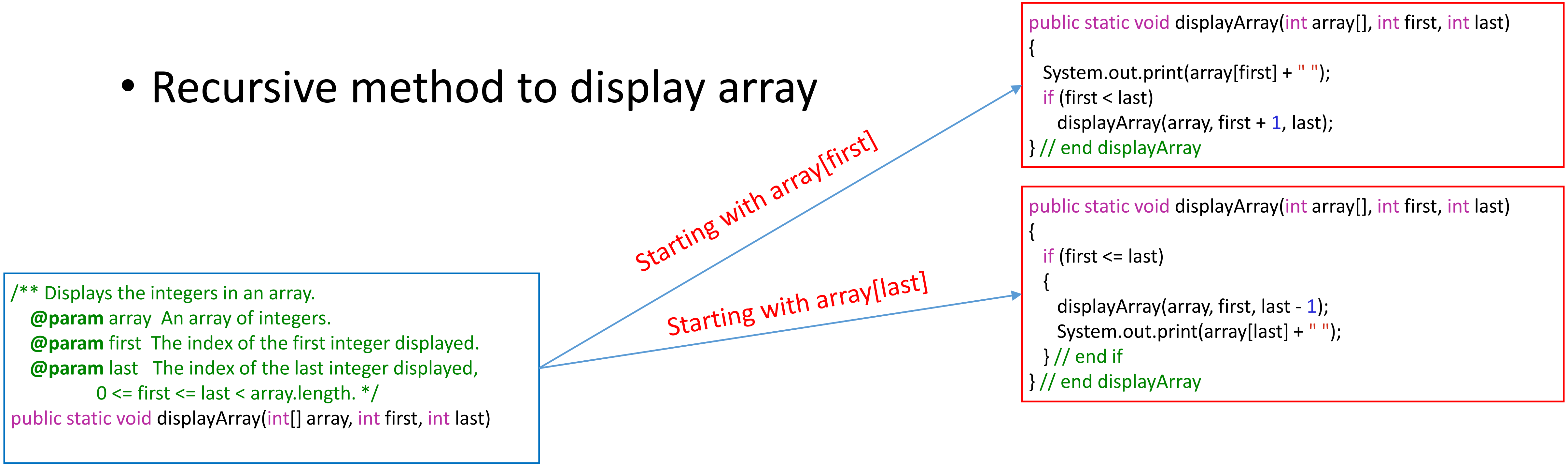
```
public static void displayArray(int array[], int first, int last)
{
  System.out.print(array[first] + " ");
  if (first < last)
    displayArray(array, first + 1, last);
} // end displayArray
```

```
public static void displayArray(int array[], int first, int last)
{
  if (first < last)
    displayArray(array, first + 1, last);

  System.out.print(array[first] + " ");
} // end displayArray
```

?

Recursively Processing an Array

- Recursive method to display array



```
/** Displays the integers in an array.
   @param array  An array of integers.
   @param first  The index of the first integer displayed.
   @param last   The index of the last integer displayed,
             0 <= first <= last < array.length. */
public static void displayArray(int[] array, int first, int last)
```

Starting with array[first]

```
public static void displayArray(int array[], int first, int last)
{
  System.out.print(array[first] + " ");
  if (first < last)
    displayArray(array, first + 1, last);
} // end displayArray
```

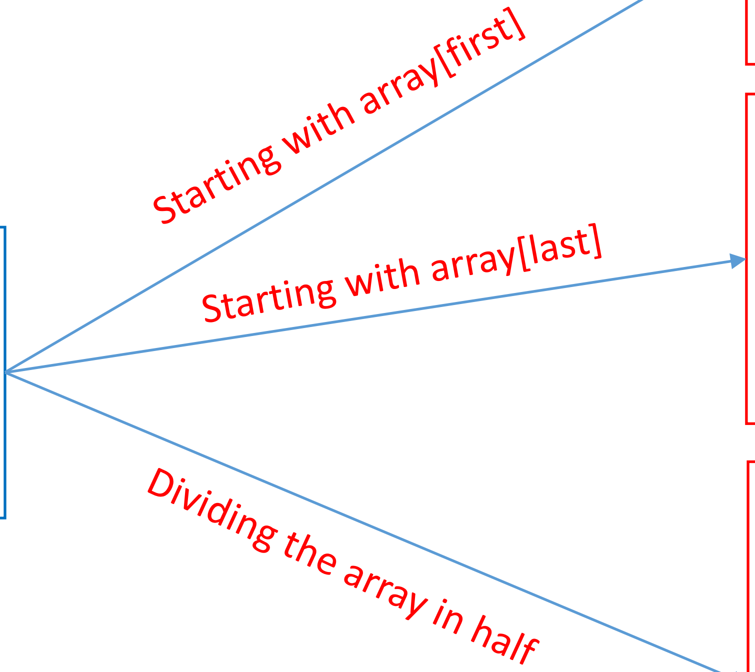
Starting with array[last]

```
public static void displayArray(int array[], int first, int last)
{
  if (first <= last)
  {
    displayArray(array, first, last - 1);
    System.out.print(array[last] + " ");
  } // end if
} // end displayArray
```

Recursively Processing an Array

- Recursive method to display array

```
/** Displays the integers in an array.
    @param array  An array of integers.
    @param first  The index of the first integer displayed.
    @param last   The index of the last integer displayed,
              0 <= first <= last < array.length. */
public static void displayArray(int[] array, int first, int last)
```



Starting with array[first]

```
public static void displayArray(int array[], int first, int last)
{
  System.out.print(array[first] + " ");
  if (first < last)
    displayArray(array, first + 1, last);
} // end displayArray
```

Starting with array[last]

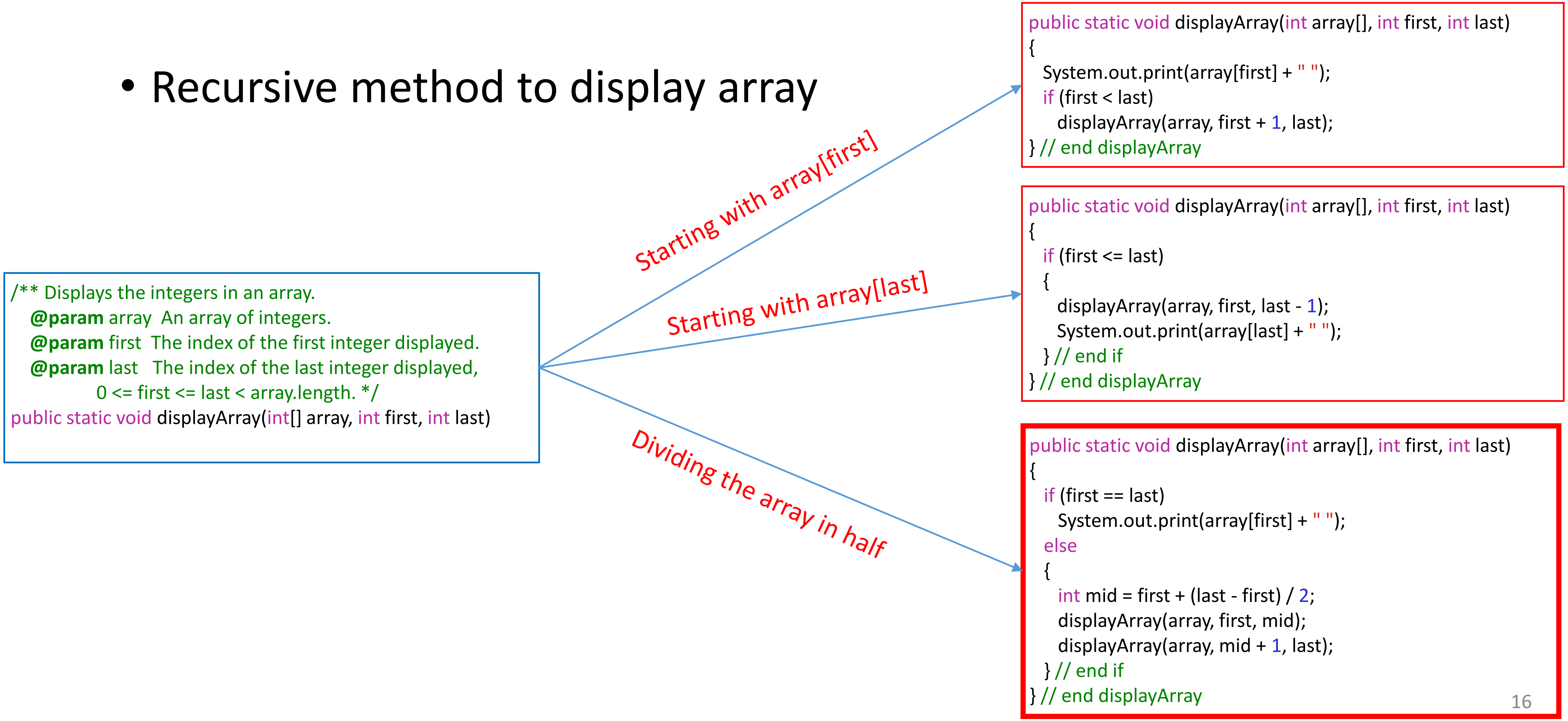
```
public static void displayArray(int array[], int first, int last)
{
  if (first <= last)
  {
    displayArray(array, first, last - 1);
    System.out.print(array[last] + " ");
  } // end if
} // end displayArray
```

Dividing the array in half

```
public static void displayArray(int array[], int first, int last)
{
  if (first == last)
    System.out.print(array[first] + " ");
  else
  {
    int mid = first + (last - first) / 2;
    displayArray(array, first, mid);
    displayArray(array, mid + 1, last);
  } // end if
} // end displayArray
```

Recursively Processing an Array

- Recursive method to display array



```
/** Displays the integers in an array.
   @param array  An array of integers.
   @param first  The index of the first integer displayed.
   @param last   The index of the last integer displayed,
            0 <= first <= last < array.length. */
public static void displayArray(int[] array, int first, int last)
```

Starting with array[first]

```
public static void displayArray(int array[], int first, int last)
{
  System.out.print(array[first] + " ");
  if (first < last)
    displayArray(array, first + 1, last);
} // end displayArray
```

Starting with array[last]

```
public static void displayArray(int array[], int first, int last)
{
  if (first <= last)
  {
    displayArray(array, first, last - 1);
    System.out.print(array[last] + " ");
  } // end if
} // end displayArray
```

Dividing the array in half

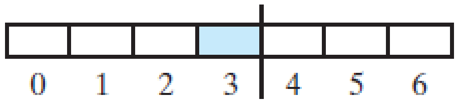
```
public static void displayArray(int array[], int first, int last)
{
  if (first == last)
    System.out.print(array[first] + " ");
  else
  {
    int mid = first + (last - first) / 2;
    displayArray(array, first, mid);
    displayArray(array, mid + 1, last);
  } // end if
} // end displayArray
```

Recursively Processing an Array

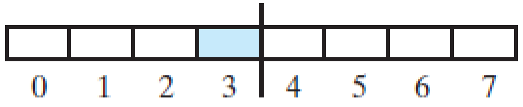
- Dividing the array in half

```
public static void displayArray(int array[], int first, int last)
{
  if (first == last)
    System.out.print(array[first] + " ");
  else
  {
    int mid = first + (last - first) / 2;
    displayArray(array, first, mid);
    displayArray(array, mid + 1, last);
  } // end if
} // end displayArray
```

(a)



(b)



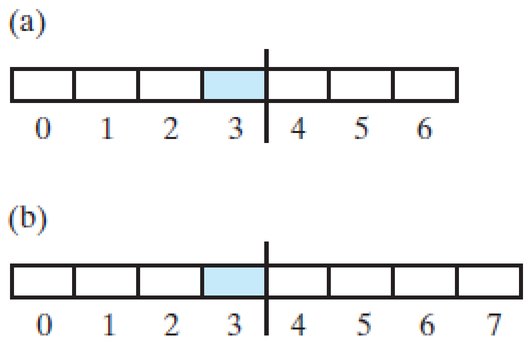
Recursively Processing an Array

- Dividing the array in half

```
public static void displayArray(int array[], int first, int last)
{
  if (first == last)
    System.out.print(array[first] + " ");
  else
  {
    int mid = first + (last - first) / 2;
    displayArray(array, first, mid);
    displayArray(array, mid + 1, last);
  } // end if
} // end displayArray
```

Why? Instead of

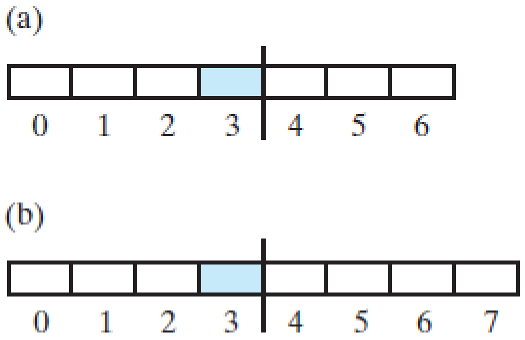
int mid = (first + last)/ 2;



Recursively Processing an Array

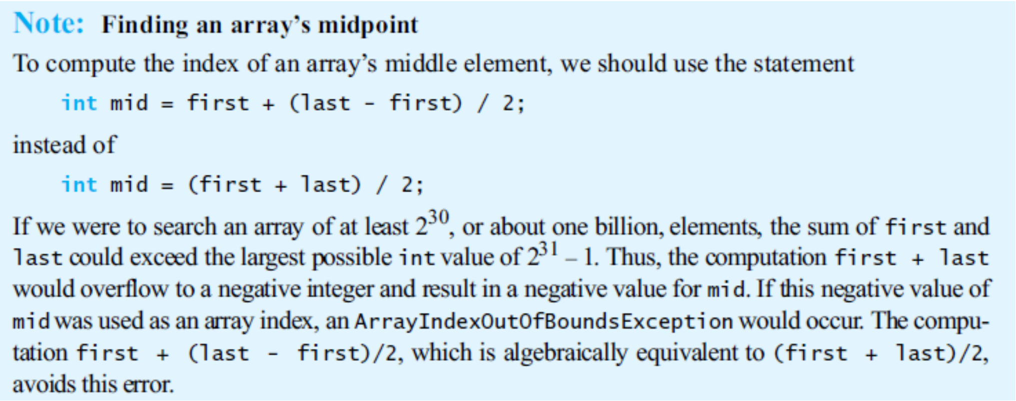
- Dividing the array in half

```
public static void displayArray(int array[], int first, int last)
{
  if (first == last)
    System.out.print(array[first] + " ");
  else
  {
    int mid = first + (last - first) / 2;
    displayArray(array, first, mid);
    displayArray(array, mid + 1, last);
  } // end if
} // end displayArray
```



Why? Instead of

int mid = (first + last)/ 2;



Note: Finding an array's midpoint

To compute the index of an array's middle element, we should use the statement

```
int mid = first + (last - first) / 2;
```

instead of

```
int mid = (first + last) / 2;
```

If we were to search an array of at least 2^{30}, or about one billion, elements, the sum of `first` and `last` could exceed the largest possible `int` value of $2^{31} - 1$. Thus, the computation `first + last` would overflow to a negative integer and result in a negative value for `mid`. If this negative value of `mid` was used as an array index, an `ArrayIndexOutOfBoundsException` would occur. The computation `first + (last - first)/2`, which is algebraically equivalent to `(first + last)/2`, avoids this error.

Recursively Processing a Linked Chain

- Display data in first node and recursively display data in rest of chain.

```
public void display()
{
  displayChain(firstNode);
} // end display

private void displayChain(Node nodeOne)
{
  if (nodeOne != null)
  {
    System.out.println(nodeOne.getData()); // Display data in first node
    displayChain(nodeOne.getNextNode());   // Display rest of chain
  } // end if
} // end displayChain
```

Recursively Processing a Linked Chain

- Display data in first node and recursively display data in rest of chain.

```
public void display()
{
  displayChain(firstNode);
} // end display

private void displayChain(Node nodeOne)
{
  if (nodeOne != null)
  {
    System.out.println(nodeOne.getData()); // Display data in first node
    displayChain(nodeOne.getNextNode());   // Display rest of chain
  } // end if
} // end displayChain
```

How to display a chain backwards?

Recursively Processing a Linked Chain

- Display a chain backwards

```
public void displayBackward()
{
  displayChainBackward(firstNode);
} // end displayBackward

private void displayChainBackward(Node nodeOne)
{
  if (nodeOne != null)
  {
    displayChainBackward(nodeOne.getNextNode());
    System.out.println(nodeOne.getData());
  } // end if
} // end displayChainBackward
```

Time Efficiency of Recursive Methods

countDown

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

Efficiency of algorithm is O(n).

Computing x^n

$$x^n = (x^{n/2})^2 \text{ when } n \text{ is even and positive}$$

$$x^n = x\,(x^{(n-1)/2})^2 \text{ when } n \text{ is odd and positive}$$

$$x^0 = 1$$

Efficiency of algorithm is O(log *n*)

Using a Stack Instead of Recursion

- Converting a recursive method to an iterative one

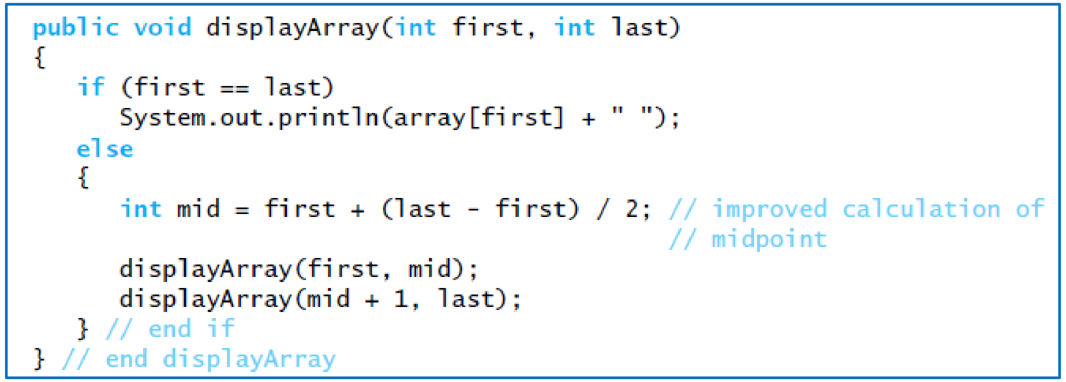
```
public static void countDown(int integer)
{
  if (integer >= 1)
  {
    System.out.println(integer);
    countDown(integer - 1);
  } // end if
} // end countDown
```

- An iterative version

```
public static void countDown(int integer)
{
  while (integer >= 1)
  {
    System.out.println(integer);
    integer = integer – 1;
  } // end while
} // end countDown
```

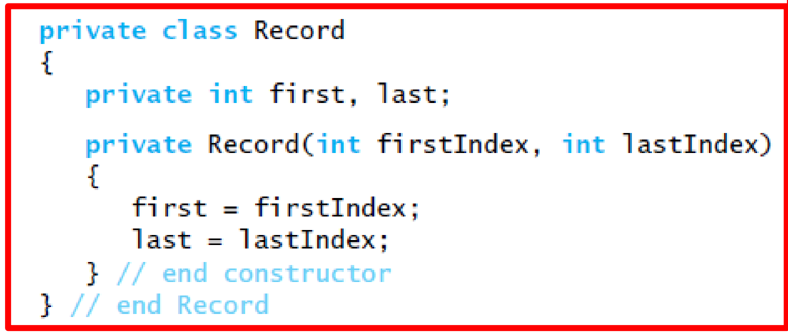
Using a Stack Instead of Recursion

- An iterative displayArray to maintain its own stack



```
public void displayArray(int first, int last)
{
   if (first == last)
      System.out.println(array[first] + " ");
   else
   {
      int mid = first + (last - first) / 2; // improved calculation of
                                             // midpoint
      displayArray(first, mid);
      displayArray(mid + 1, last);
   } // end if
} // end displayArray
```

```
public void displayArray(int first, int last)
{
  boolean done = false;
  StackInterface<Record> programStack = new LinkedStack<>();
  programStack.push(new Record(first, last));
  while (!done && !programStack.isEmpty())
  {
    Record topRecord = programStack.pop();
    first = topRecord.first;
    last = topRecord.last;

    if (first == last)
      System.out.println(array[first] + " ");
    else
    {
      int mid = first + (last - first) / 2;
      // Note the order of the records pushed onto the stack
      programStack.push(new Record(mid + 1, last));
      programStack.push(new Record(first, mid));
    } // end if
  } // end while
} // end displayArray
```



```
private class Record
{
   private int first, last;

   private Record(int firstIndex, int lastIndex)
   {
      first = firstIndex;
      last = lastIndex;
   } // end constructor
} // end Record
```

In-Class Exercises

- Write a recursive method and an iterative method to generate Fibonacci Sequence

Fibonacci Sequence

$F(1) = 1$

$F(2) = 1$

$F(n) = F(n\text{-}1) + F(n\text{-}2)$ for $n > 2$

Sequence $F \Rightarrow 1, 1, 2, 3, 5, 8, 13,\ldots.$

Summary

- Recursion

What I Want You to Do

- Review class slides
- Review Chapter 9