Rather than replacing an element, another way to change the contents of an array is to *insert* an element. [Just now](https://www.eimacs.com/eimacs/mainpage?epid=E2267325640&cid=162149), we saw that replacing an array element results in the destruction of the element being replaced. Insertion, on the other hand, does not necessarily entail the destruction of an existing object. In fact, the first stage of performing an element insertion is to create a space for the new element by moving existing elements.

Specifically, starting with the element at the insertion index, each element must be moved one space to the right (see the figure on the right, which illustrates the preparation for inserting an element at index 1). Once the space has been created, the new item can be inserted into it:

There are two "gotchas":

* If elements are moved to the right, what happens to the last element of the array? It has nowhere to go.
* The elements must be moved to the right one at a time. In which order should they be moved?

Regarding the first of these, there are two possible strategies: First, we could allow the last array element to "drop off the edge of the cliff". That is to say, we could simply discard it. Second, we could preserve the element in some way, perhaps by extending the array (which is the strategy employed by ArrayLists). For our purposes here, we adopt the first strategy.

Concerning the second question, consider the following code fragment in which a new Item is inserted at index 1 as follows: First, the element currently at index 1 is copied to index 2. Then the element at index 2 is copied to index 3, and so on, and the element currently at the end of the array (at index 4) is discarded. Finally, the new element is placed into the array at index 1. Is this a correct algorithm for inserting the new item? Run the code to find out.

  public static void main( String[] args )   
  {   
    Item[] array = Item.makeItemArray( 5 );   
  
    System.out.println( "Before: " );   
    Item.displayArray( array );   
  
    // make a new Item   
    Item newItem = new Item( 99 );   
  
    // make space for the item   
    array[ 2 ] = array[ 1 ];   
    array[ 3 ] = array[ 2 ];   
    array[ 4 ] = array[ 3 ];   
  
    // insert new item   
    array[ 1 ] = newItem;   
  
    System.out.println( "\nAfter: " );   
    Item.displayArray( array );   
  }

[Show program details »](https://www.eimacs.com/eimacs/mainpage?cid=162149&epid=E2114558232)

Before:    
Item: 0   
Item: 1   
Item: 2   
Item: 3   
Item: 4   
  
After:    
Item: 0   
Item: 99   
Item: 1   
Item: 1   
Item: 1

No, it is not a correct algorithm. When we copy array[ 1 ] into array[ 2 ], we overwrite the Item currently stored in array[ 2 ], losing it forever. The mistake is then compounded by the subsequent copies, which in fact cause the original object at index 1 to overwrite *all* the elements that come after it.

To avoid this problem, we copy the elements in order from the right to left, as follows: First, we copy array[ 3 ] into array[ 4 ], thereby overwriting array[ 4 ] and effectively discarding it. Next, we copy array[ 2 ] into array[ 3 ]. Although this action overwrites array[ 3 ], that element is not lost because we just got through copying it into array[ 4 ]. Then array[ 1 ] is copied into array[ 2 ], and finally we can place the new item into array[ 1 ]. Run the following code to verify that the insertion is made correctly:

  public static void main( String[] args )   
  {   
    Item[] array = Item.makeItemArray( 5 );   
  
    System.out.println( "Before: " );   
    Item.displayArray( array );   
  
    // make a new Item   
    Item newItem = new Item( 99 );   
  
    // make space for the item   
    array[ 4 ] = array[ 3 ];   
    array[ 3 ] = array[ 2 ];   
    array[ 2 ] = array[ 1 ];   
  
    // insert new item   
    array[ 1 ] = newItem;   
  
    System.out.println( "\nAfter: " );   
    Item.displayArray( array );   
  }

[Show program details »](https://www.eimacs.com/eimacs/mainpage?cid=162149&epid=E2247651365)

Before:    
Item: 0   
Item: 1   
Item: 2   
Item: 3   
Item: 4   
  
After:    
Item: 0   
Item: 99   
Item: 1   
Item: 2   
Item: 3

**Exercise 169**

The above code works correctly provided the array has exactly five elements. In the code fragment below, complete the for-loop so that the insertion works correctly with a variety of array sizes:

  public static void main( String[] args )   
  {   
    Item[] array = Item.makeItemArray(  );   
  
    System.out.println( "Before: " );   
    Item.displayArray( array );   
  
    // make a new Item   
    Item newItem = new Item( 99 );

 for ( int i = array.length - 1 ; i > 1 ; i-- )

  // array[ i ] = array[ i-1 ];

    // insert new item   
    array[ 1 ] = newItem;   
  
    System.out.println( "\nAfter: " );   
    Item.displayArray( array );   
  }

[Show program details »](https://www.eimacs.com/eimacs/mainpage?cid=162149&epid=E2247651365)

Before:    
Item: 0   
Item: 1   
Item: 2   
Item: 3   
Item: 4   
  
After:    
Item: 0   
Item: 99   
Item: 2   
Item: 3   
Item: 4