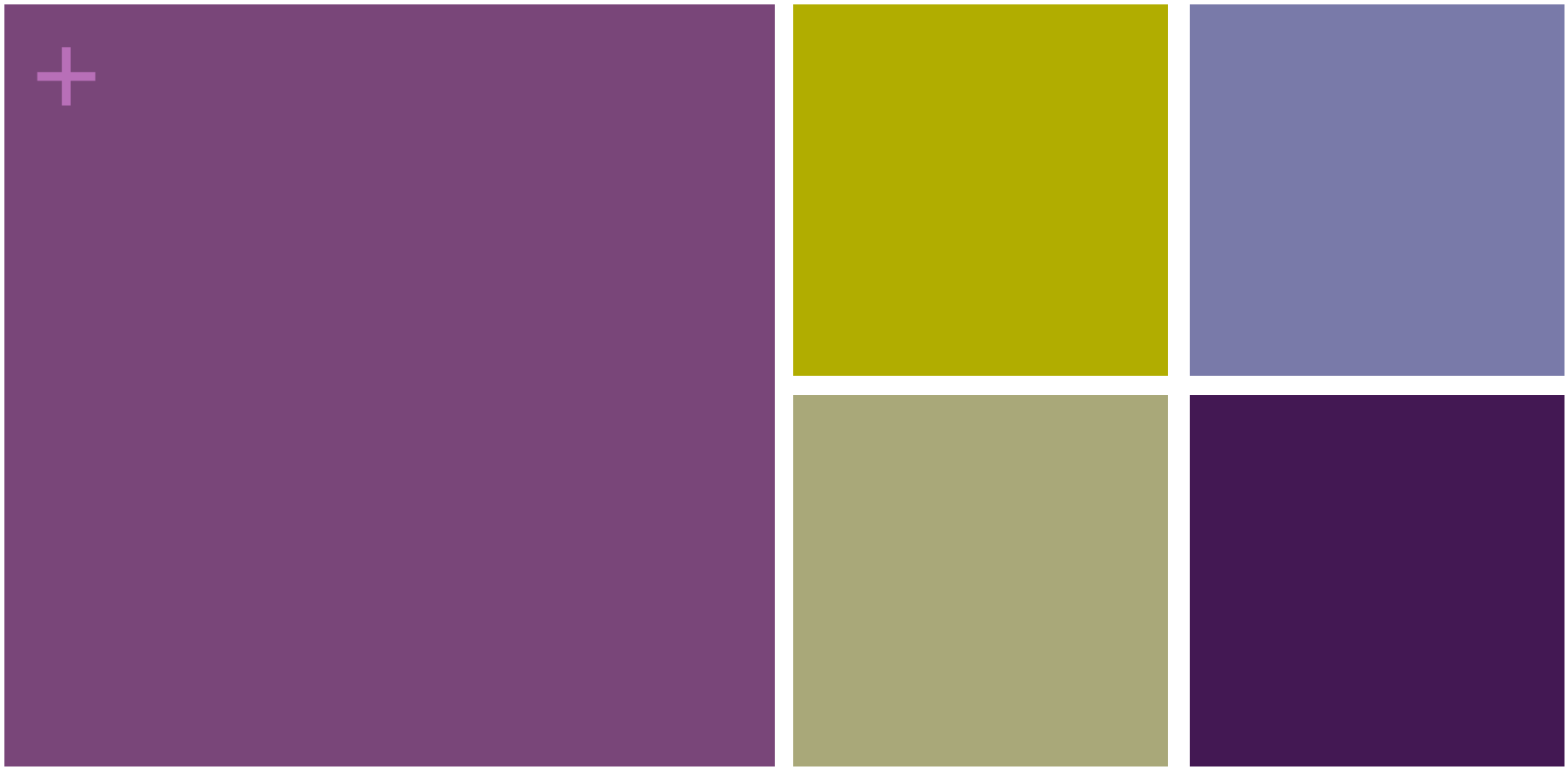


More on One Dimensional Arrays

Introduction to Computer Science CSCI UA.0101

Lecture 9

Agenda Day 9

- Processing arrays
- Arrays and methods
- Variable length argument lists
- Copying arrays
- Searching Arrays
- Sorting Arrays
- Using the Arrays helper class

Programming Challenges Sites

- https://projecteuler.net/
- https://www.codechef.com/
- http://www.spoj.com/
- https://www.topcoder.com/

+

Processing Arrays

Displaying Arrays with “for” loops

- We almost always use a ‘for’ loop when working with our arrays. ‘for’ loops give you the ability to easily iterate over a known range, and we can construct one that will visit every element in an array by using the ‘length’ property of the array in the iteration condition. Here’s an example:

```
int[] myList = new int[5];

for (int c = 0; c < myList.length; c++)
{
     // print out what is at element c
     System.out.println( myList[c] );
}
```

Initializing Arrays with Input Values

- We can use a for loop to iterate over a list and get input from the user in order to populate our array. For example:

```
Scanner input = new Scanner(System.in);

int[] myList = new int[5];
for (int c = 0; c < myList.length; c++)
{
     System.out.print("Give me a number: ");
     myList[c] = input.nextInt();
}

for (int c = 0; c < myList.length; c++)
{
     System.out.println(c + " = " + myList[c]);
}
```

Initializing Arrays with Random Values

- We can also populate an array with random values.
  For example:

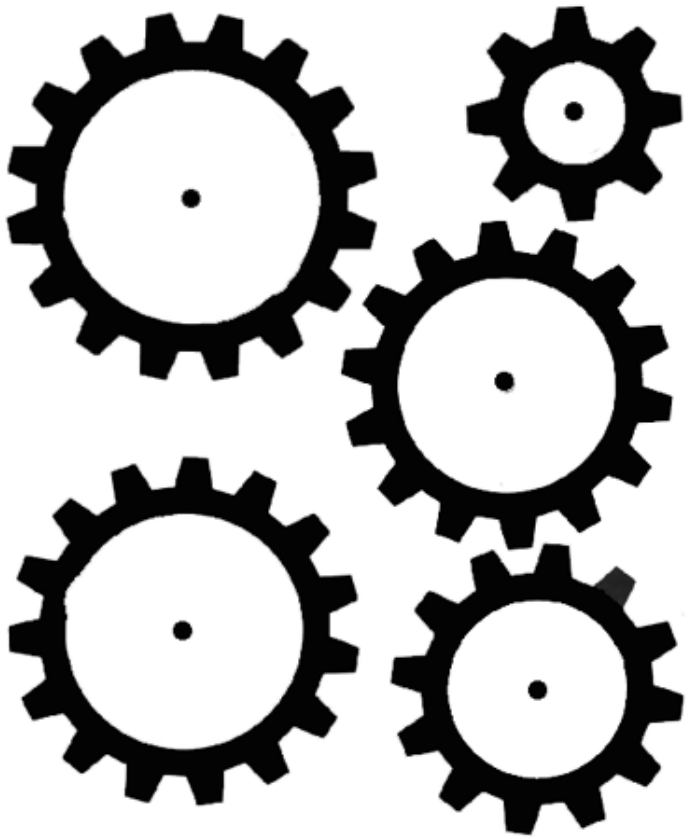
```
int[] myList = new int[5];

for (int c = 0; c < myList.length; c++)
{
     myList[c] = (int) (Math.random() * 10 + 1);
}

for (int c = 0; c < myList.length; c++)
{
     System.out.println(c + " = " + myList[c]);
}
```

Programming Challenge

- Given the following array:

```
int[] myList =
{6,4,2,6,1,2,4,9}
```

- Find the following:
  - The sum of all elements in the array
  - The largest value in the list
  - The smallest value in the list
  - The location of the largest value in the list
- **ProcessingArrays.java**



Programming Challenge

- Given the following array:

```
int[] myList =
{1,2,3,4,5,6,7}
```

- Shift the array so that each element moves one position to the left.
- The leftmost element should cycle around to the end of the list.
- **ArrayShifting.java**

Lookup Arrays

- You can also use arrays to simplify your code by acting as a "lookup" table. For example:

```
Scanner input = new Scanner(System.in);

String[] months = {"Jan", "Feb", "Mar", "Apr",
"May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov",
"Dec"};

System.out.print("Select an integer 1-12");
int m = input.nextInt();

System.out.println("You selected " + months[m - 1]);
```

Arrays and Methods

Passing Arrays to Methods

- Arrays can be passed to methods in the same way that primitive data types can be passed to methods. For example, the following method accepts an array as an argument:

```
public static void doSomething(int[] myList)
```

Arrays as a reference type

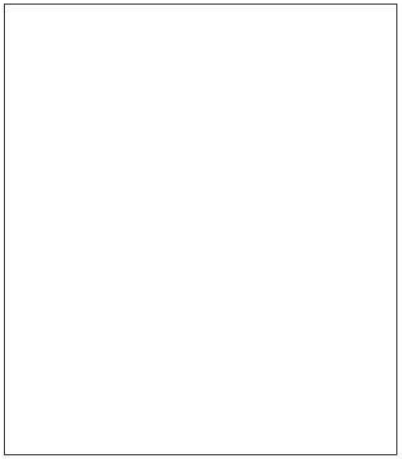
- Array variables are reference type in Java (i.e. arrays are objects in Java).
- This means that the array variable doesn't contain the actual data associated with the array. It simply stores the memory address of where the array can be found.
- When Java passes an array into a method it is passing this reference to the array, not the array itself. Because Java passes by value, it is passing the value of the array reference (the memory location) which results in a situation where a method can change the contents of the array without having to return it to the caller. Your book refers to this behavior as "passing by sharing"

Array Argument Mechanics

Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

The Stack



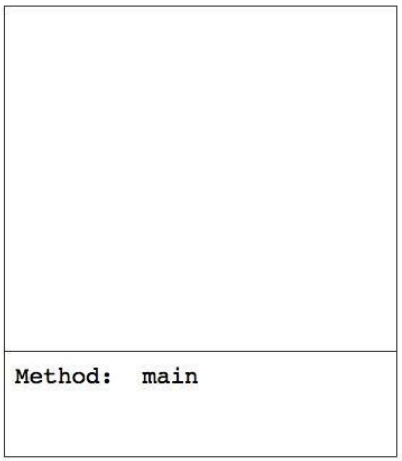
The Heap

Array Argument Mechanics

Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

The Stack



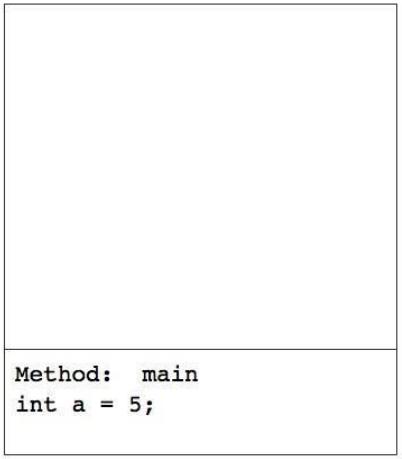
The Heap

Array Argument Mechanics

Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

The Stack



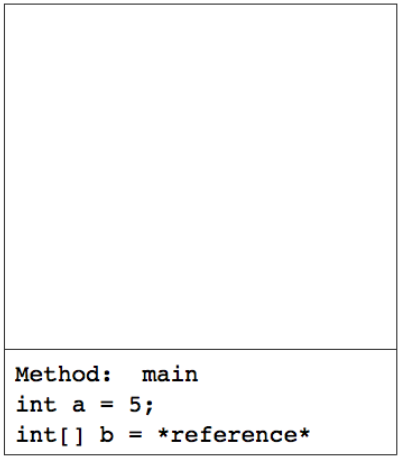
The Heap

Array Argument Mechanics

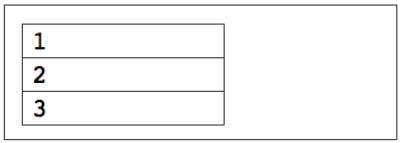
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

The Stack



The Heap

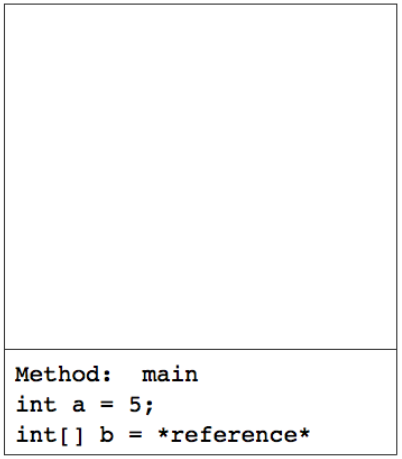


Array Argument Mechanics

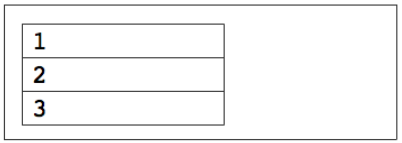
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

The Stack



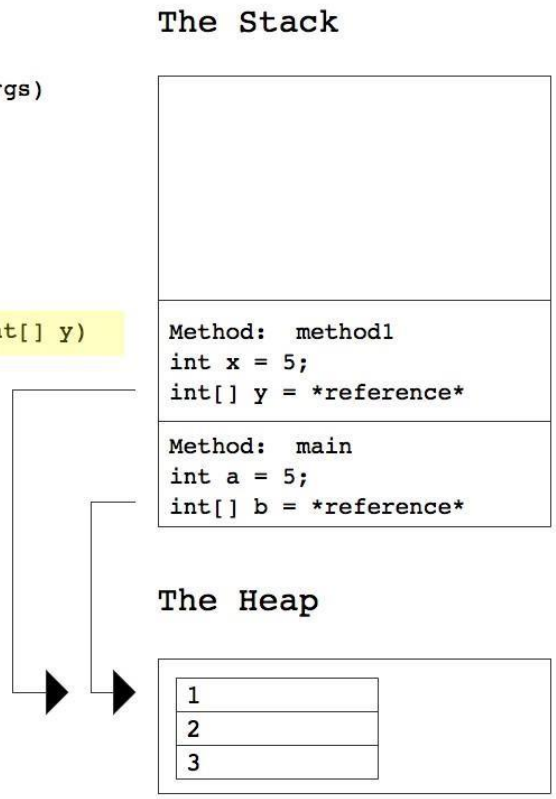
The Heap



Array Argument Mechanics

Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

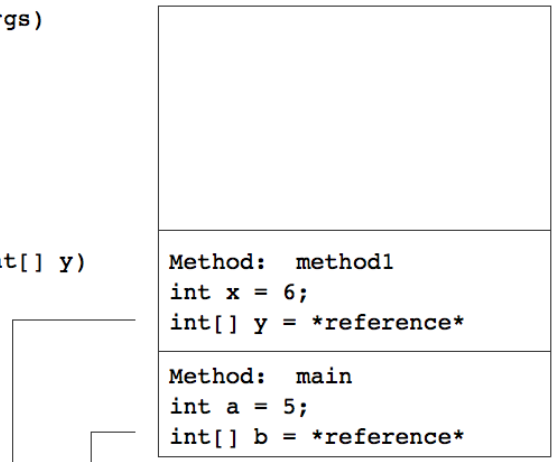


Array Argument Mechanics

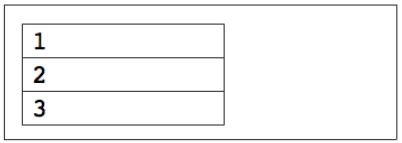
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

The Stack



The Heap

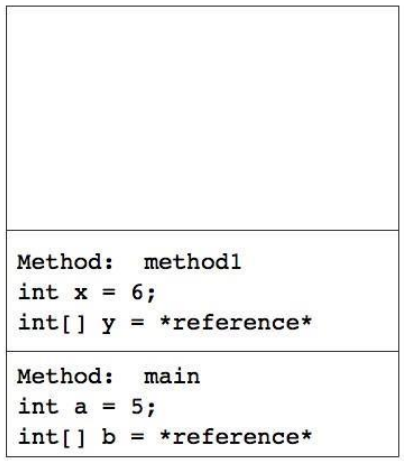


Array Argument Mechanics

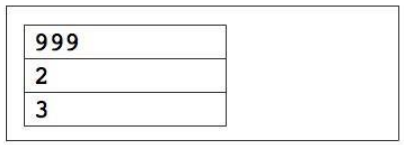
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

The Stack



The Heap

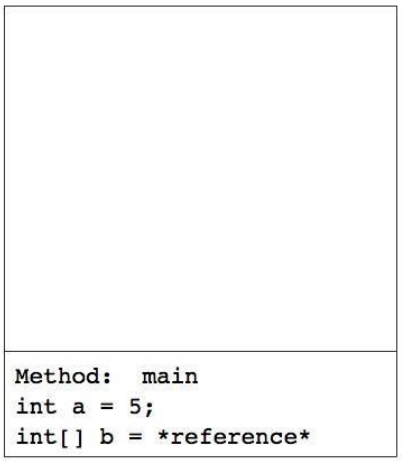


Array Argument Mechanics

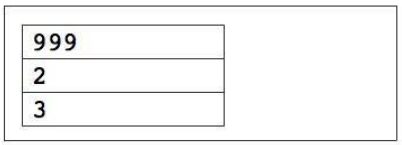
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int[] b = {1,2,3};

    method1(a, b);
}

public static int method1(int x, int[] y)
{
    x += 1;
    y[0] = 999;
}
```

The Stack



The Heap



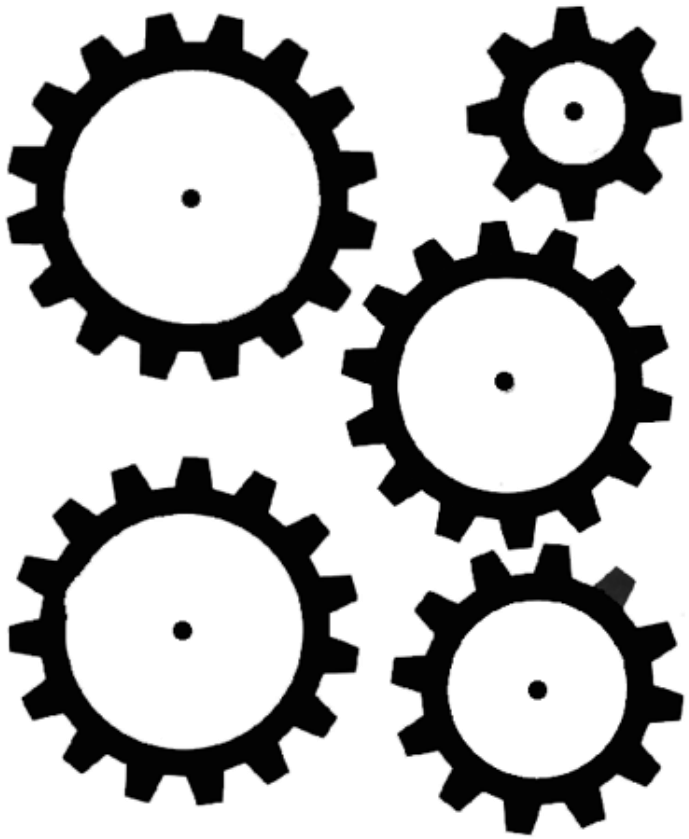
Programming Challenge

- Given the following array:

```
int[] myList =
{6,4,2,6,1,2,4,9}
```

- Find the following:
  - The sum of all elements in the array
  - The largest value in the list
  - The smallest value in the list
  - Solve the problem using methods

- **ArrayNumbers.java**



Programming Challenge (TestPassArray.java)

- Comparison between passing a primitive type to a method and passing an array to a method.
- A primitive type is passed by value: i.e. the original value of the variable is not changed by the method.
- An array is passed by *sharing*: i.e. the original values of the array can be changed by the method.

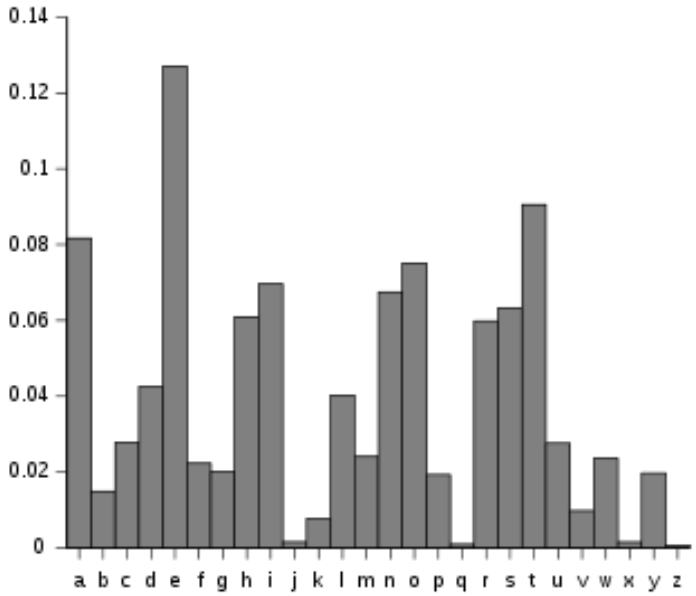
Programming Challenge (PriceValues.java)

- Write a program that asks the user for a number of price values (i.e. 5 prices)
- Then ask the user to enter these prices as a series of doubles and store them in an array.
- Next, apply 7% sales tax to the following array.
- Write the following methods in order to complete this task:

```
public static void enterValues(double[]p)
public static void applyTax(double[]p)
public static void printList(double[] p)
```

Programming Challenge (CountLettersInArray.java)

- Write a program that does the following:
  - Generate an array of 100 random characters (a-z)
  - Display the array, 10 characters per line
  - Count the number of occurrences of each character in a new array
  - Display the frequency (i.e. 'a' was printed 5 times)
- Perform the above steps using methods (1 method per step)



Returning Arrays from Methods

Returning Arrays from Methods

- You can also return an array from a method using the return statement. For example:

```
public static int[] myMethod()
{
     int[] a = {1,2,3};
     return a;
}
```

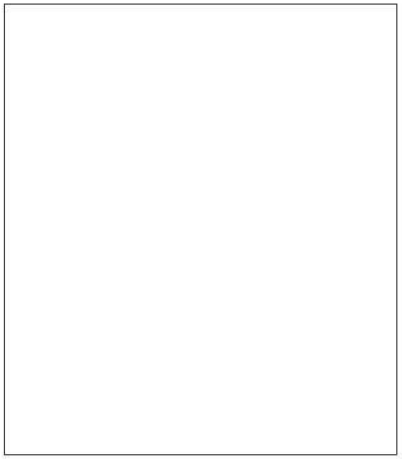
- In this case the array a is created inside the method myMethod. The reference to the array is local to myMethod, but when it is returned to the caller the reference value is fully accessible.

Returning Arrays from Methods

Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



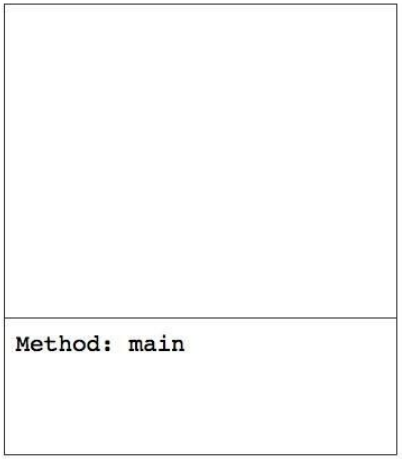
The Heap

Returning Arrays from Methods

Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



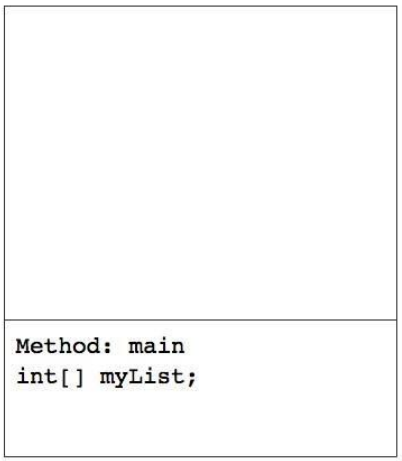
The Heap

Returning Arrays from Methods

Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



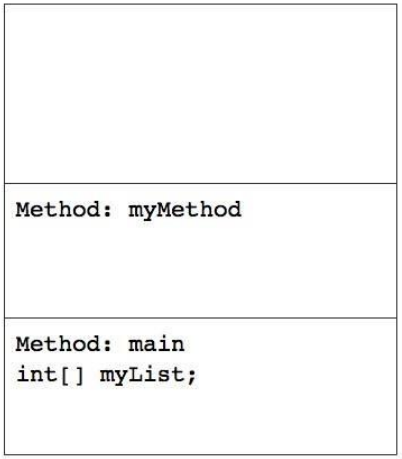
The Heap

Returning Arrays from Methods

Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



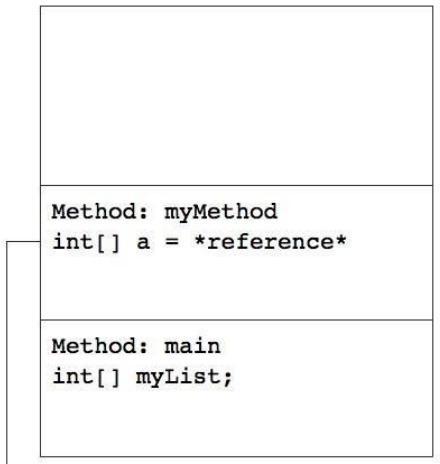
The Heap

Returning Arrays from Methods

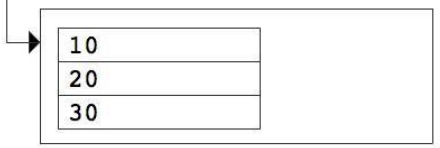
Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



The Heap

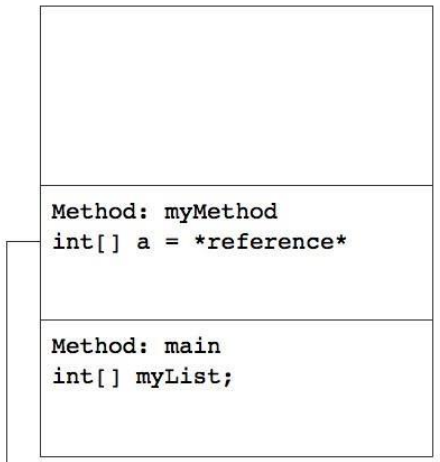


Returning Arrays from Methods

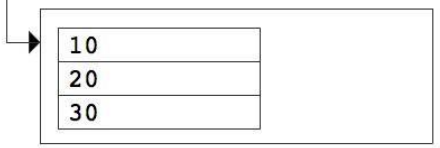
Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



The Heap

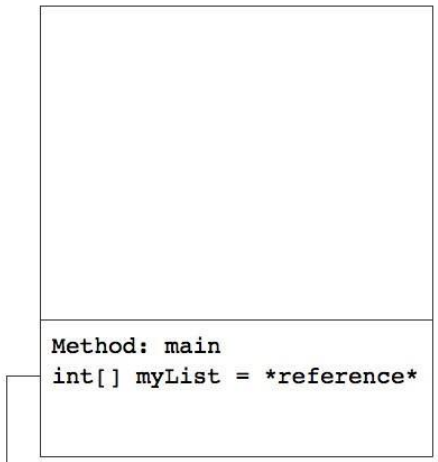


Returning Arrays from Methods

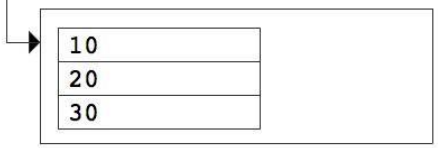
Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



The Heap

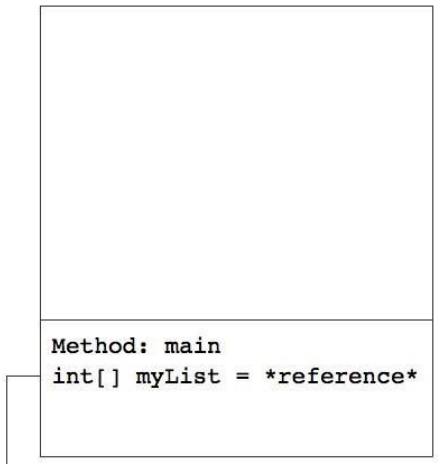


Returning Arrays from Methods

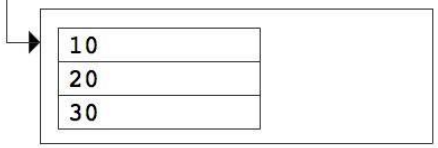
Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



The Heap

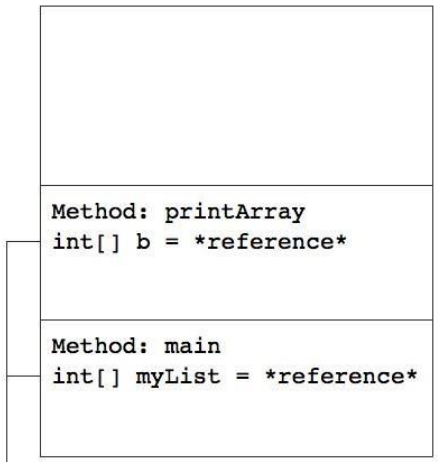


Returning Arrays from Methods

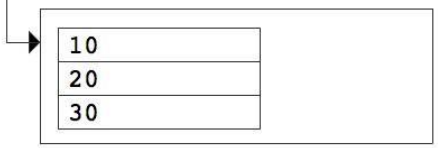
Source Code

```
public static void main(String[] args)
{
    int[] myList = myMethod();

    printArray(myList);
}

public static int[] myMethod()
{
    int[] a = {10,20,30};
    return a;
}

public static void printArray(int[] b)
{
    for (int i = 0; i < b.length; i++)
    {
        System.out.println(b[i]);
    }
}
```

The Stack



The Heap



Variable Length Argument Lists

Variable Length Argument Lists

- There are times when you want to send an arbitrary number of arguments to a method
- For example:

```
int a = max(5,10,20,30,-5,999,300,12);
```

- In this case, building an overloaded version of the max method for n arguments would be incredibly time consuming and inefficient.

Variable Length Argument Lists

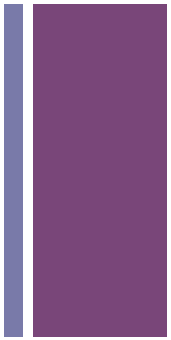
- In Java we can instruct a method to accept a variable number of arguments using "ellipsis" syntax, as follows:

```
public static void myMethod(int… numbers)
```

- This essentially tells the argument that it can accept multiple values of the same time (ints in this case) and that the multiple values should be treated as a single array called "numbers". For example:

```
myMethod(5,10,20,30,999,-50);
```

Variable Length Argument Lists



- You may only have one variable length argument in a method header, and it must always be the last argument. For example:

  public static void method1(int a, double …b) //OK

  public static void method2(String …a, int b) //error (vararg should be last)

  public static void method3(String…a, double …b) //error (two varargs)

- If you omit the variable length argument list when calling a method Java will simply create an empty array and send it to the method – no exception will be raised.

Programming Challenge (VariableLengthArguments.java)

- Write a program that accepts a varying number of doubles
- Compute the following:
  - Total number of items
  - Largest item
  - Smallest item
  - Average item
- Return a new array that contains these values stored in the following format:
  - `returnArray = {total, largest, smallest, average}`

Copying Arrays

Copying Arrays

- With primitive data types you can copy values from one variable to another by using the assignment operator:

```
int a = 5;
int b = a;
```

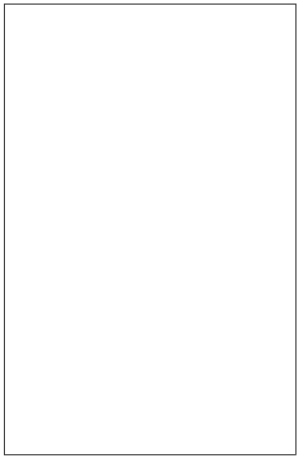
- However, with reference types you cannot do this since Java only stores the value of the memory address of where the object can be found on the heap.

Copying Arrays

Source Code

```
public static void main(String[] args)
{
    int[] a = {9,8,7};
    int[] b = {1,2,3};

    b = a;
}
```

The Stack



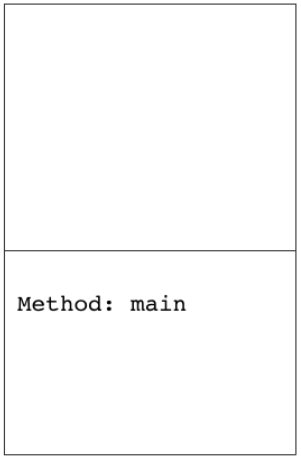
The Heap

Copying Arrays

Source Code

```
public static void main(String[] args)
{
    int[] a = {9,8,7};
    int[] b = {1,2,3};

    b = a;
}
```

The Stack

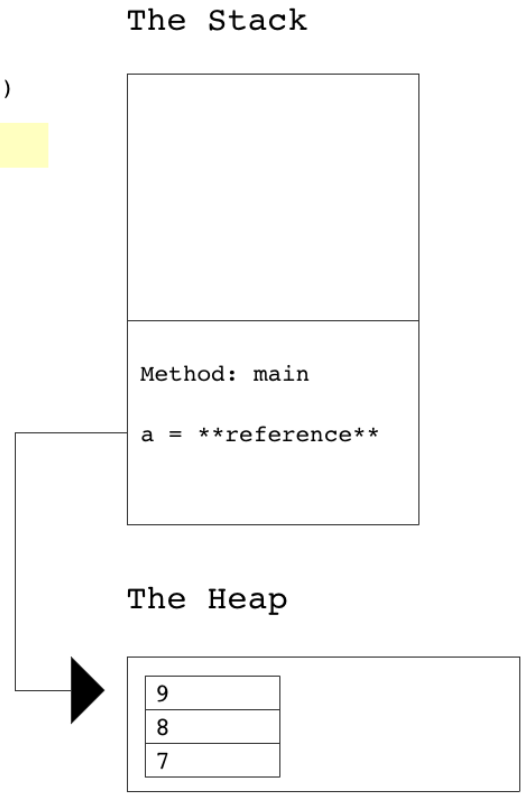


The Heap

Copying Arrays

Source Code

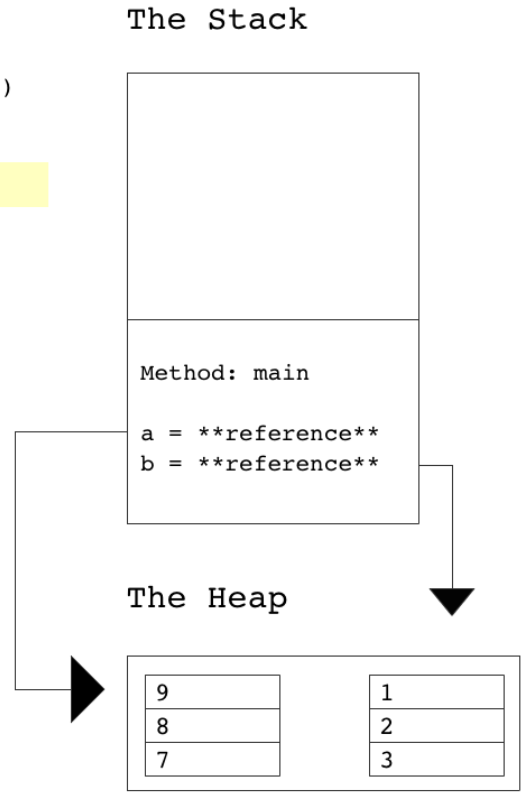
```
public static void main(String[] args)
{
    int[] a = {9,8,7};
    int[] b = {1,2,3};

    b = a;
}
```



Copying Arrays

Source Code

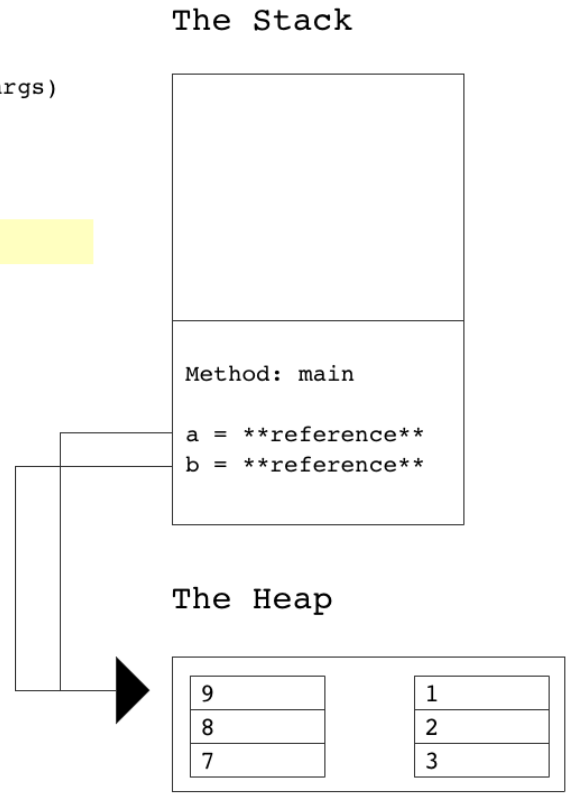
```
public static void main(String[] args)
{
    int[] a = {9,8,7};
    int[] b = {1,2,3};

    b = a;
}
```



Copying Arrays

Source Code

```
public static void main(String[] args)
{
    int[] a = {9,8,7};
    int[] b = {1,2,3};

    b = a;
}
```



Copy Method 1: Looping

- One way to copy an array is to use a loop to copy array elements individually. For example:

```
int[] a = {1,2,3};
int[] b = new int[3];

for (int x = 0; x < a.length; x++)
{
     b[x] = a[x];
}
```

Copy Method 2: System.arraycopy

- The System class in Java contains a method called "arraycopy" that allows you to copy over information from one array to another using.
- The syntax for using this method is as follows:

```
System.arraycopy(sourceArray, sourcePosition,
                 destArray, destPosition,
                 length)
```

- Where sourceArray is the array you wish to copy from, sourcePosition is the position you want to start copying from, destArray is the array you wish to copy to, destPosition is the position you want to start placing elements, length is the number of elements you wish to copy.
- Note that you must first create destArray if you want to use System.arraycopy – the method won't allocate space on the heap for your new array.

Programming Challenge (MergeArrays.java)

- Write a method that merges two arrays into a single array. Use the following method header. Implement this method by using the "loop" technique.

```
public static int[] merge(int[] a, int[] b)
```

- Write a method that merges two arrays using the "arraycopy" technique. Use the following method header:

```
public static double[] merge(double[] a, double [] b)
```

Searching Arrays

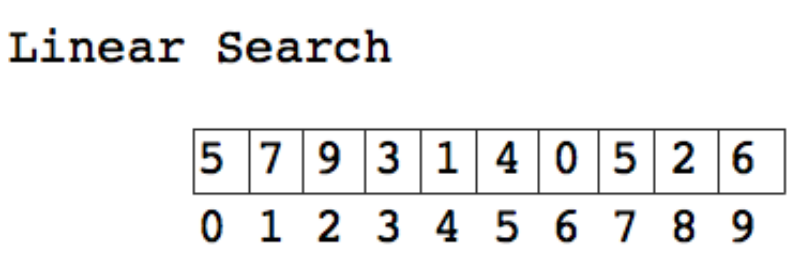
Searching Arrays

- Searching is the process of looking for a specific data element and isolating its location for use by another part of your program.
- There are many approaches and techniques that you can use to search an array for a desired element. Each one has its own advantages and benefits depending on what you're trying to accomplish in your program.

Linear Search

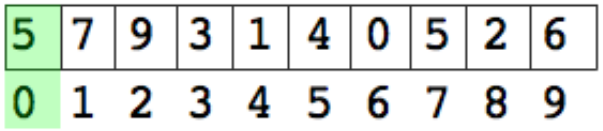
- A linear search will sequentially iterate through each element in a list until a match is found.
- Typically a linear search will return the location of the found data element, or a special value (such as -1) if the element cannot be found.

Linear Search



Linear Search

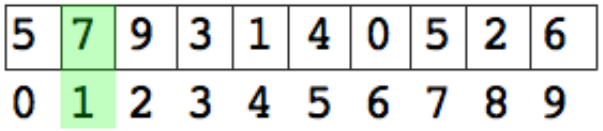
Linear Search



Searching for 1

Linear Search

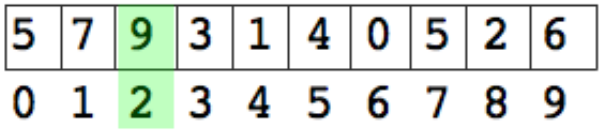
Linear Search



Searching for 1

Linear Search

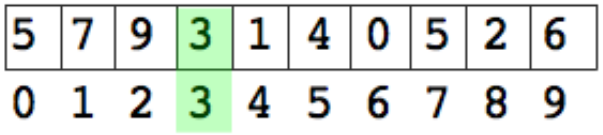
Linear Search



Searching for 1

Linear Search

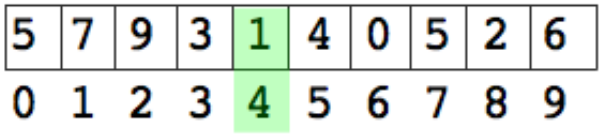
Linear Search



Searching for 1

Linear Search

Linear Search



Searching for 1

Found at position 4

Binary Search

- A binary search is an efficient search that allows you to quickly find a desired element in a sorted array. Here's how it works:
  - A binary search will begin by defining the upper and lower boundary of the array.
  - It will then look at the middle element between the upper and lower boundary. If that is the element in question, the search stops.
  - If it is not, the search will determine if it should look in the upper half of the region or the lower half. It will then repeat this process on that smaller region.
- Binary searches do not work on arrays that are unsorted, so you must ensure that your array is in ascending order before attempting to use this technique.

Binary Search

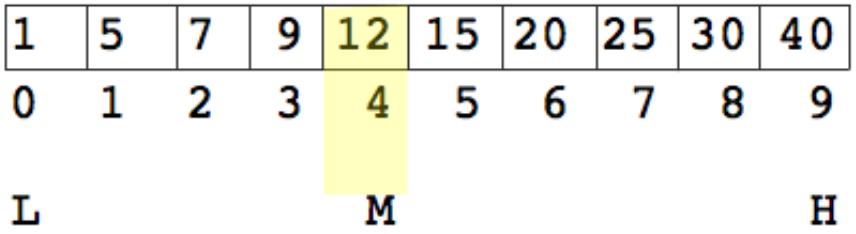
Binary Search

1	5	7	9	12	15	20	25	30	40
0	1	2	3	4	5	6	7	8	9

Searching for 7

Binary Search

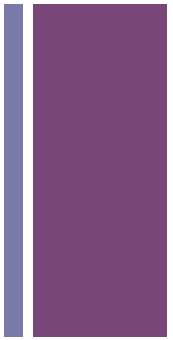
Binary Search



Searching for 7

```
low  = 0
high = 9
mid  = (0 + 9) / 2 = 4
```

Binary Search



Binary Search

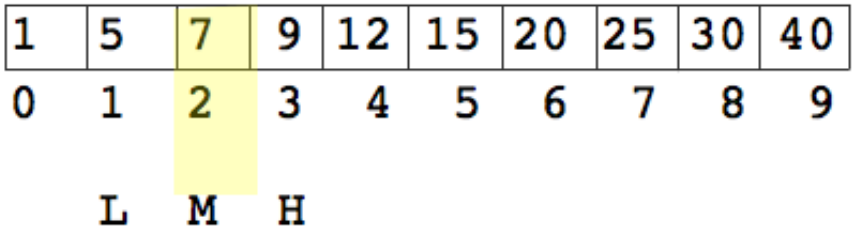
1	5	7	9	12	15	20	25	30	40
0	1	2	3	4	5	6	7	8	9
L	M		H						

Searching for 7

```
low  = 0
high = 3
mid  = (0 + 3) / 2 = 1
```

Binary Search

Binary Search



Searching for 7

```
low  = 2
high = 3
mid  = (2 + 3) / 2 = 2
```

+

Sorting Arrays

Sorting Arrays

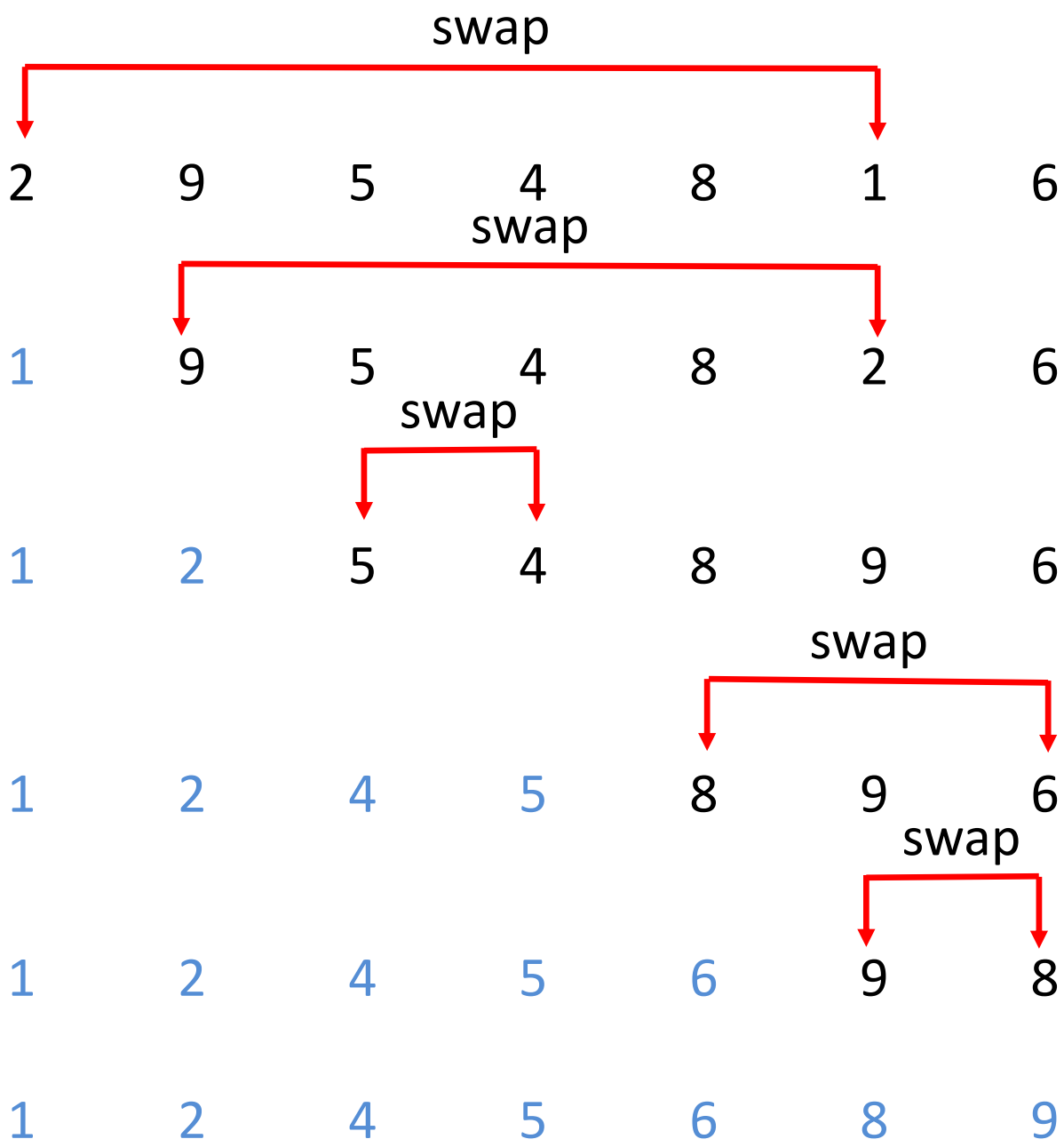
- Sorting is a common technique in programming, and there are many strategies that can be employed to reorder an array in ascending or descending order.

- One common sorting technique for sorting an array is the "Selection Sort" which works by linearly searching an array for the smallest item and swapping its position with the lowest unsorted item in the list.

Selection Sort

- The selection sort works by starting off at element zero
- It then looks at elements 1 through n in order
- If the element in question is less than element zero the elements are swapped
- When we reach the end of the list we are guaranteed element zero is the smallest element
- We then repeat the process for elements 1 through n-1

Selection Sort



Using the Arrays class

java.util.Arrays

- Java contains a built-in class for handling many common array tasks, such as sorting, searching and printing an array.
- For example, many of the techniques we have covered so far have corresponding methods in this class:
  - java.util.Arrays.sort(list) – sorts an array in ascending order
  - java.util.Arrays.binarySearch(list,item) – performs a binary search on a sorted array and returns the index of the element that contains the item in question
  - java.util.Arrays.toString(list) – quickly prints the contents of an array to a String

Programming Challenge (SearchAlgorithms.java)

- Write a program that creates 100 unique random integers between 0 and 999 and stores them in an array.
- Sort the array in ascending order and print out the result to the user using methods of the Arrays class
- Ask the user to enter a number and return the index of the location of the number in the array or a negative number if the number is not on the array using linear and binary search algorithms.