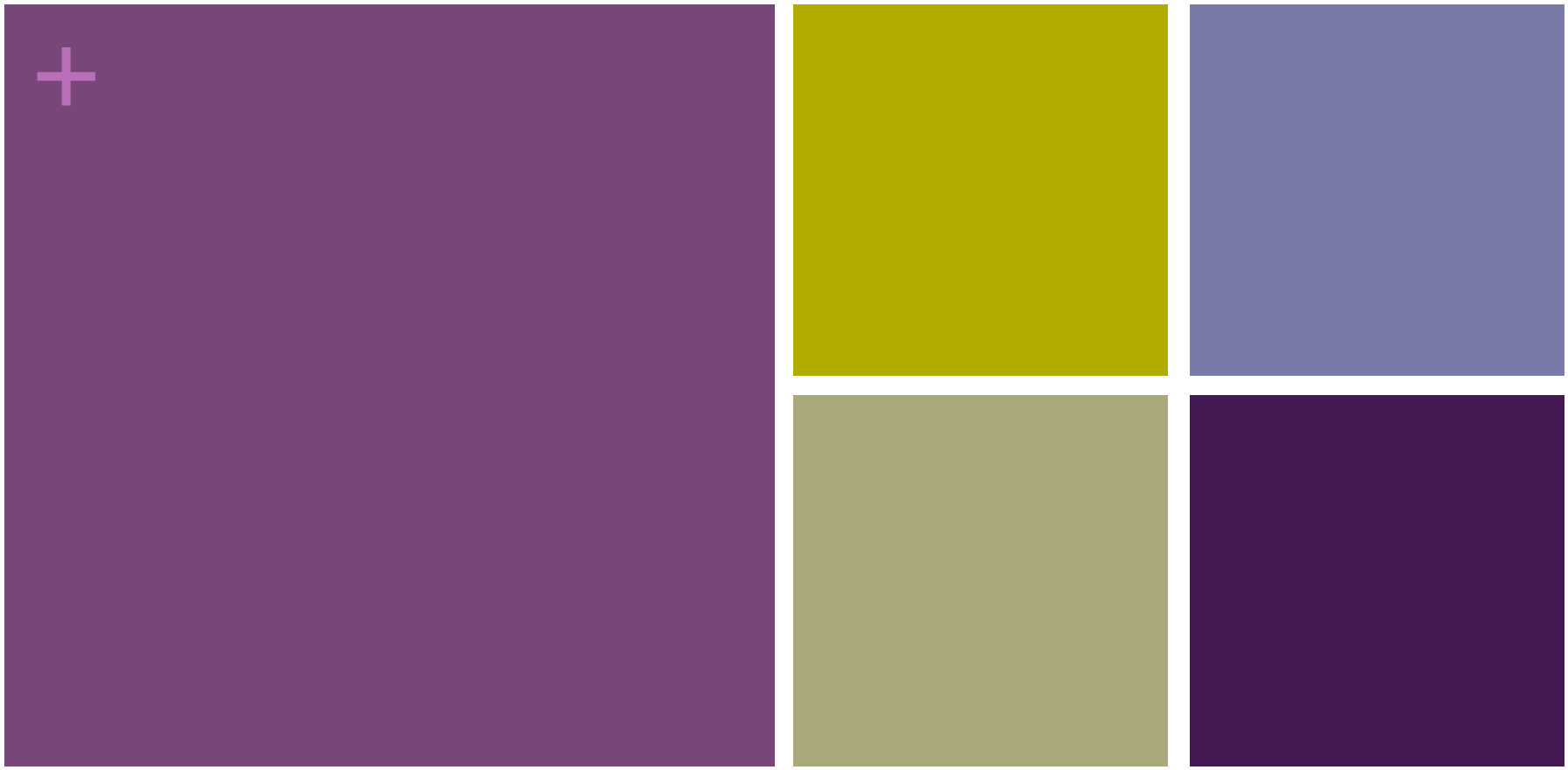


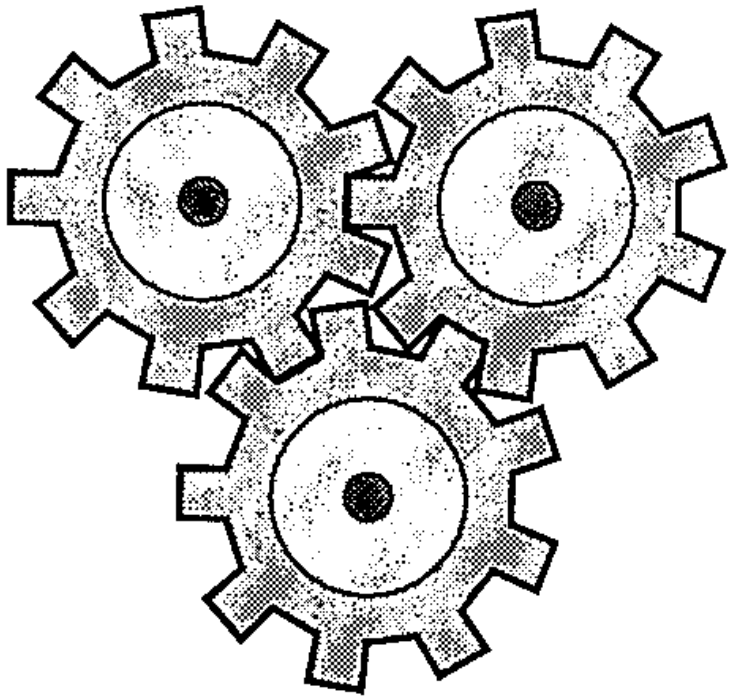
One Dimensional Arrays

Introduction to Computer Science CSCI UA.0101

Lecture 8

Agenda Day 8

- Brief review
- Array Basics
- Array Processing
- Arrays and Methods



Modularizing Code

Modularizing Code

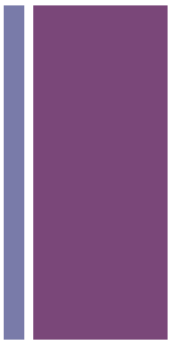
- Methods are often used to reduce redundant code and to facilitate code reuse by other programs.
- For example:

```
for (int i = 0; i < 100; i++)
{
      System.out.println("hello");
}
```

Can be modularized into:

```
public static void m0(int times)
{
      for (int j = 0; j < times; j++)
      {
            System.out.println("hello");
      }
}
```

Modularizing Code



- When modularizing your code you should attempt to identify which aspects of your code can be "encapsulated"
- This allows you to isolate problems in your logic and break down difficult tasks into smaller, more discrete unites
- This technique also allows you to re-use your code in other programs without having to re-write or copy code from program to program.

Programming Challenge

- Write a program to produce a sales report modularizing your code:

  1) ask the user to enter the daily sales

  2) calculate the average daily sales

  3) display the total and average

- Each step should be handled by a different method.

- **Sales.java**

“return” mechanics

- The return statement must be used in methods that declare a return value. For example:

```
public static int soSomething()
```

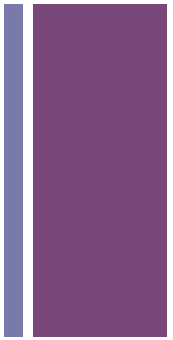
- This method must at some point use the “return” statement to return an integer to the calling method.

“return” mechanics

- Java forces you to ensure that at least one of your “return” statements is “reachable”
- This means that there must be a 100% chance that one of your return statements will execute. For example, the following method contains a potentially unreachable return statement:

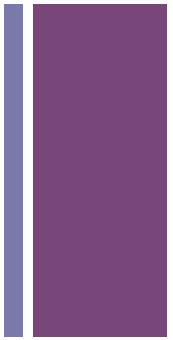
```
public static int doSomething(int a) {
     if (a > 0) {
          return 100;
     }
}
```

“reachable” or “unreachable”?



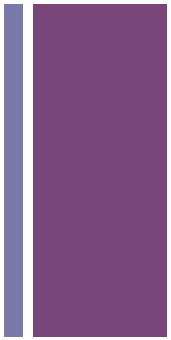
```
public static int doSomething(int a)
{
        if (a > 0)
        {
                return 100;
        }
        if (a < 0)
        {
                return -100;
        }
}
```

“reachable” or “unreachable”?



```
public static int doSomething(int a)
{
        if (a > 0)
        {
                return 100;
        }
        if (a < 0)
        {
                return -100;
        }
}
```

“reachable” or “unreachable”?

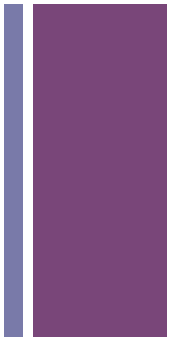


```
public static int doSomething(int a)
{
        if (a > 0)
        {
                return 100;
        }
        if (a < 0)
        {
                return -100;
        }
        if (a == 0)
        {
                return 0;
        }
}
```

“reachable” or “unreachable”?

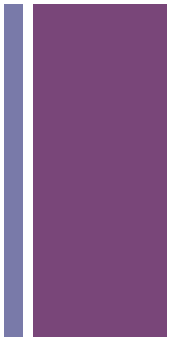
```
public static int doSomething(int a)
{
        if (a > 0)
        {
                return 100;
        }
        if (a < 0)
        {
                return -100;
        }
        if (a == 0)
        {
                return 0;
        }
}

//EXAMPLE: ReturnReachable.java
```

“reachable” or “unreachable”?



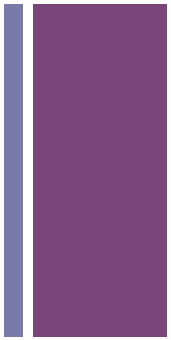
```
public static int doSomething(int a)
{
        if (a > 0)
        {
                return 100;
        }
        else if (a < 0)
        {
                return -100;
        }
        else if (a == 0)
        {
                return 0;
        }
}
```

“reachable” or “unreachable”?



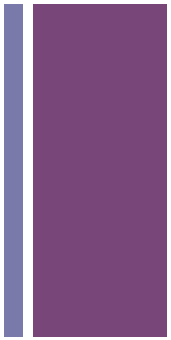
```
public static int doSomething(int a)
{
        if (a > 0)
        {
                return 100;
        }
        else if (a < 0)
        {
                return -100;
        }
        else if (a == 0)
        {
                return 0;
        }
}
```

“reachable” or “unreachable”?



```
public static int doSomething(int a)
{
        if (a > 0)
        {
                return 100;
        }
        else if (a < 0)
        {
                return -100;
        }
        else
        {
                return 0;
        }
}
```

“reachable” or “unreachable”?



```
public static int doSomething(int a)
{
        if (a > 0)
        {
                return 100;
        }
        else if (a < 0)
        {
                return -100;
        }
        else
        {
                return 0;
        }
}
```

+ “return” Statement Design Strategies

- You can have as many return statements in your methods as you’d like
- At least one must be “reachable” at all times
- It’s easier to have a single return statement in your code, like this:

```
public static int doSomething(int a)
{
        int retValue = 0;

        if (a > 0)
        {
                retValue = 100;
        }
        if (a < 0)
        {
                retValue = -100;
        }
        if (a == 0)
        {
                retValue =  0;
        }

        return retValue;          //SINGLE RETURN STATEMENT
}                                 //ReturnReachable1.java
```

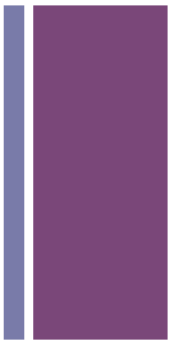
Overloading Methods

Overloading Methods

- When calling a method you must part careful attention to the parameters you are sending. Any mismatch – either in number, data type or parameter ordering – can raise an exception and cause your program to crash. For example:

```
public static void main (String[] args)
{
     double x = max(1.25, 5.75);
}

public static int max(int a, int b)
{
     if (a < b) { return b; }
     else { return a; }
}
```

Overloading Methods



- Java supports a technique called "overloading" that lets you define multiple versions of the same method that can be invoked using different parameter sets.
- When overloading a method you simply define a different method for each version you wish to create, making sure that each version uses the same name but different parameter sets. Java will figure out at compile time which method to invoke based on the data provided. Example:

```
public static int addTwo (int a, int b)

public static double addTwo (double a, double b)
```

Programming Challenge

- Write a method that calculates the maximum value of 2 integers.
- Then write an overloaded version of this method that calculates the maximum value of 2 doubles.
- Finally, write an overloaded version of this method that calculates the maximum value of 3 doubles.

MethodOverloading.java

Array Basics

Arrays

- An array in Java is defined as a “fixed-sized, sequential collection of elements of the same data type”
- Arrays allow us to create a single identifier – called an array reference – that can be used to organize many items of the same type.
- Arrays are OBJECTS.

Declaring an Array

- Arrays, like all variables, must be declared prior to use.
- You can declare an Array using the following syntax. Note the bracket notation after the data type:

```
elementType[] arrayReferenceVariable;
```

- "elementType" can be any data type you want. The trailing brackets after the data type tell Java that this variable will be treated as a collection of items of those data types.
- All elements in an array must be of the same type (i.e. you can't have an array that holds both ints and doubles)
- For example, if you want to declare an Array of integers you could do the following:

```
int[] myIntegerArray;
```

Creating Arrays

- Declaring an array is not enough to begin using it in your program. Before you use an array you need to tell Java to create space in memory for all of the elements it will be holding.

- You can do this using the following syntax:

```
elementType[] arrayReferenceVariable;
arrayReferenceVariable = new elementType[size];
```

- The "size" value determines the number of items that your array can hold. Note that arrays cannot grow beyond this size, unlike in Python. We will discuss techniques to get around this later in this chapter.

Creating Arrays

- For example, if you wanted to create an array to hold 100 integers you could do the following:

```
int[] myIntArray;
myIntArray = new int[100];
```

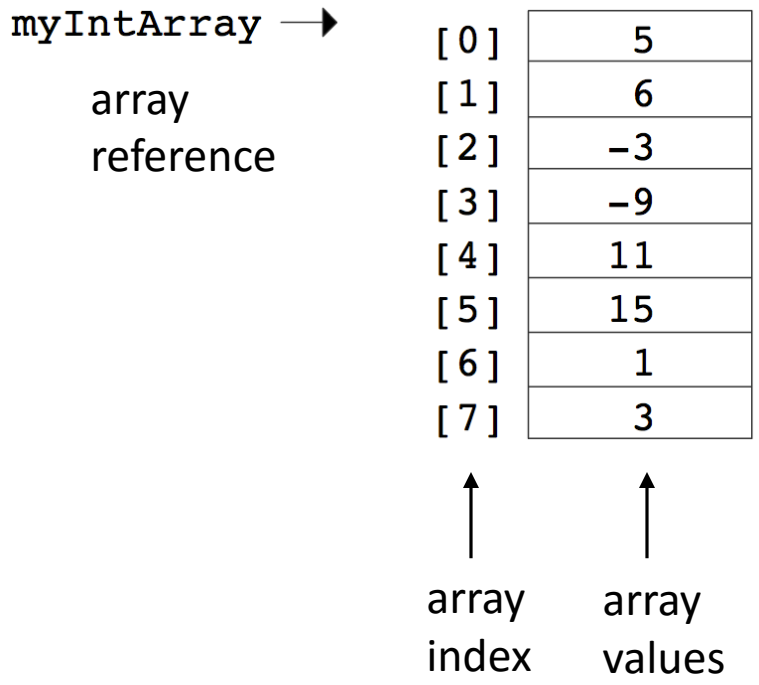
- Alternately, you could perform these two lines in one step:

```
int[] myIntArray = new int[100];
```

Accessing Array Elements

- Arrays are sequential data structures, meaning that items are stored one after another in an array.

- You can access an individual element of an array by using bracket notation. For example, we can access the 3rd element in the array to the right by doing the following:



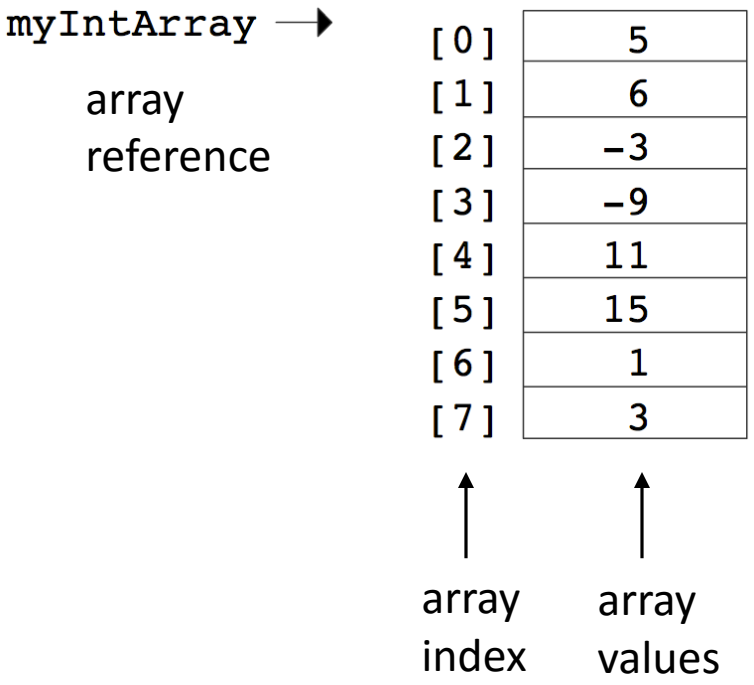

```
System.out.println( myIntArray[2] );
```

Accessing Array Elements

- Array elements always begin at index 0.
- You can count upwards from there up until the length of the array-1. For example the array to the left was created with 8 elements, like this:

```
int[] myIntArray;
myIntArray = new int[8];
```

- This will give you an array with 8 "slots", starting at position 0 and going up to position 7 (length –1)



Accessing Array Elements

- You can read any element from the array by using index notation, like this:

```
System.out.println ( myArray[0] );
```

- You can edit (write) the contents of an array element by using index notation as well:

```
// put 100 into the array at position 0
myArray[0] = 100;
```

Default Array Values

- Upon creation, arrays holding numeric types (int, double, etc) come defaulted with all elements containing a zero.
- Upon creation, arrays holding characters (char) come defaulted with all elements containing \u0000 (unicode value).
- Upon creation, arrays holding booleans come defaulted with all elements containing the value false.
- It is a good practice to initialize your arrays to a known value.

Array Initializers

- If you want to quickly create an array using a series of known values you can use the array initializer syntax as follows:

```
double myList[] = {1.1, 2.0, 5.7, 3.5};
```

- This is a stand-alone statement – if you initialize an array in this way you don't need to declare and create it. The syntax above is equivalent to the following:

```
double myList[];
myList = new double[4];
myList[0] = 1.1;
myList[1] = 2.0;
myList[2] = 5.7;
myList[3] = 3.5;
```

Array Length

- Arrays know their own length (i.e. number of elements).
- When an array is created it is created with a certain number of elements. Arrays cannot grow beyond this size.
- You can access the size of any array by using the array's "length" property. For example:

```
myList = new int[5];
System.out.println( myList.length );

// will print 5
```

Programming Challenge

- Array basics:

  -array declaration
  -array creation
  -array initialization
  -accessing elements of an array

- **ArrayBasics.java**

+

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:

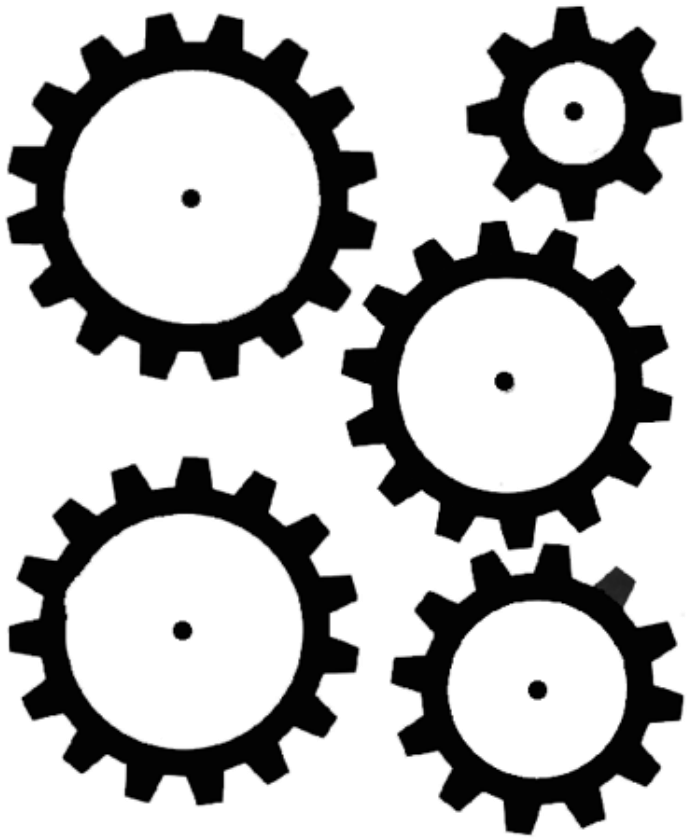
```
int[] myList = new int[5];

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

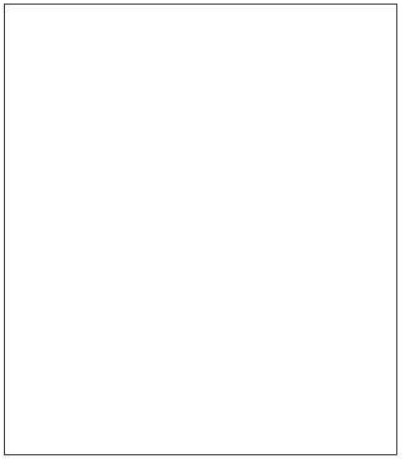
- Arrays are considered a reference type in Java (i.e. arrays are objects in Java).
- This means that the array variable doesn't contain the actual data associated with the array. It simply stores the memory address of where the array can be found.
- When Java passes an array into a method it is passing 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



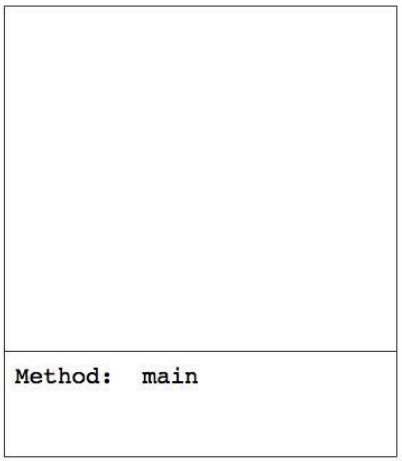
The Heap

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



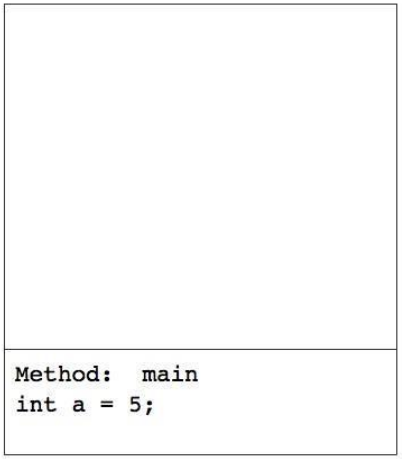
The Heap

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



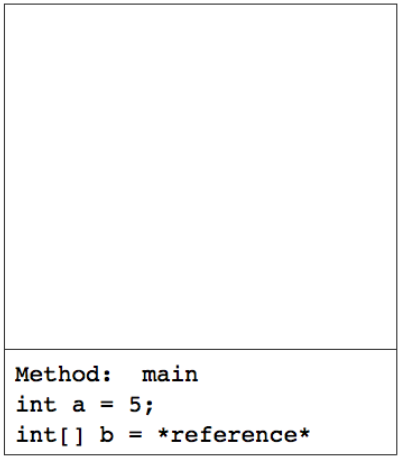
The Heap

Array Argument Mechanics

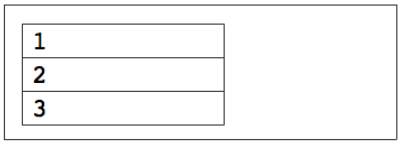
Source Code

```
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

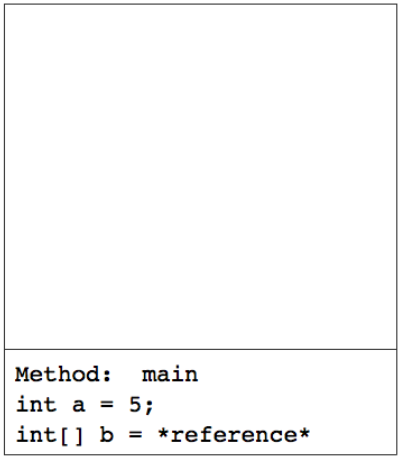


Array Argument Mechanics

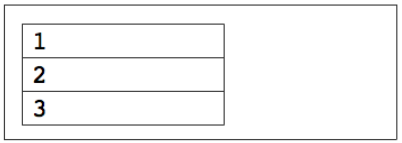
Source Code

```
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



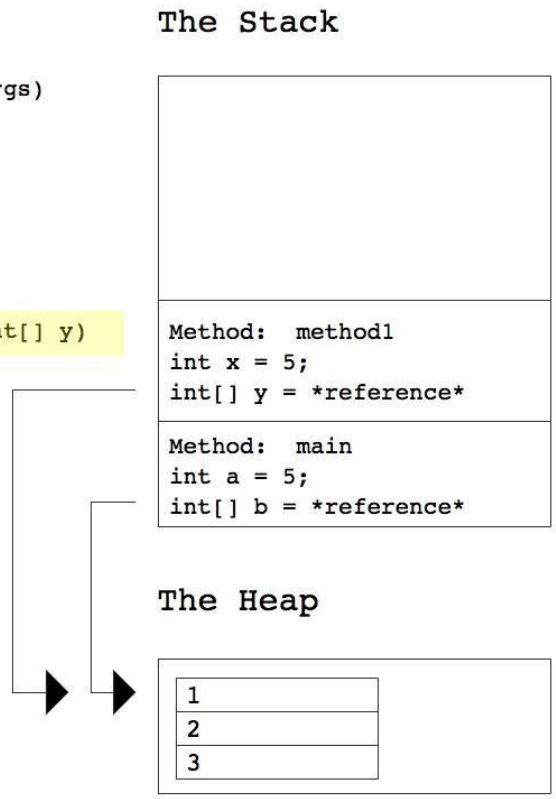
The Heap



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;
}
```

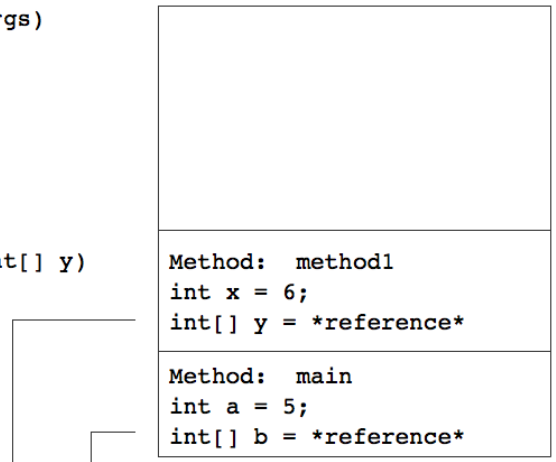


Array Argument Mechanics

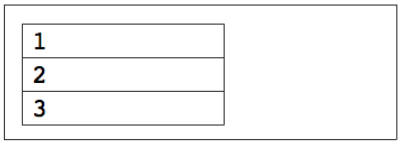
Source Code

```
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

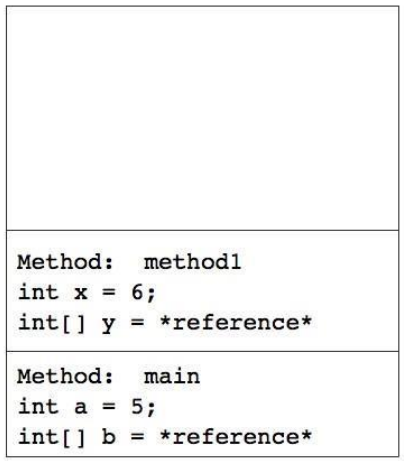


Array Argument Mechanics

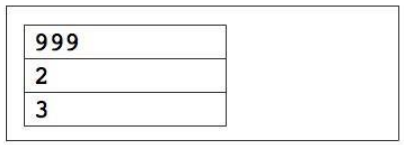
Source Code

```
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

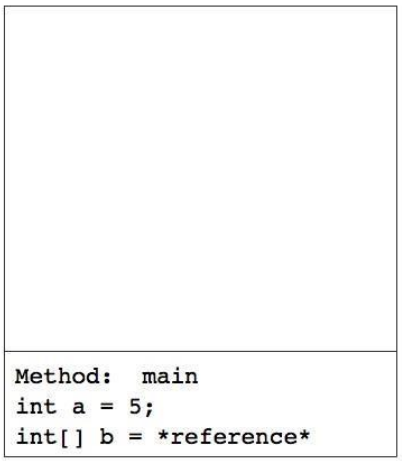


Array Argument Mechanics

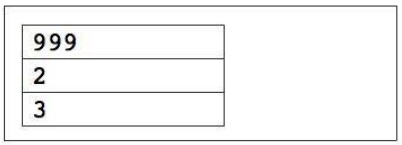
Source Code

```
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

- 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)
```

- **PriceValues.java**

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**