More Arrays and Methods

Software Development 1 (F27SA)

Michael Lones

Week 5, lecture 1

Today’s Lecture

- Some more advanced aspects of arrays
- More on using arrays with methods
- Passing object types to and from methods

Arrays (recap)

An array is a variable containing multiple values

- To declare an array variable:
```
int[] numbers;
```
- To initialise an array variable:
```
int[] numbers = {1,5,2,12,4};
```
- To make an empty array:
```
int[] numbers = new int[5];
```

Arrays (recap)

An array is a variable containing multiple values

- To read an element from an array:

```
System.out.println(numbers[0]);
System.out.println(numbers[1]);
```

- To set an element of an array:

```
numbers[0] = 10;
```

- To find the length of an array:

```
int a = numbers.length;   // 5
```

Arrays (recap)

An array is a variable containing multiple values

- To read all elements of an array:

```
for(int i=0; i<numbers.length; i++){
   System.out.println(numbers[i]);
```

- Alternatively, using "for each" notation:

```
for(int n : numbers){
   System.out.println(n);
```

Arrays (recap)

An array is a variable containing multiple values

- To write all elements of an array:

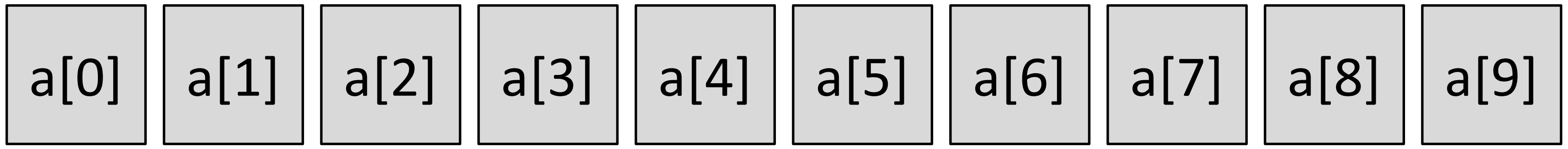
```
for(int i=0; i<numbers.length; i++){
    numbers[i] = 0;
```

- You can't use "for each" notation for this!

Multi-dimensional arrays

So far, you've seen single-dimensional arrays

```
int[] a = new int[10];
```



Multi-dimensional arrays

It is also possible to have multiple dimensions

```
int[][] a = new int[2][10];
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]	a[0][6]	a[0][7]	a[0][8]	a[0][9]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]	a[1][6]	a[1][7]	a[1][8]	a[1][9]

Multi-dimensional arrays

It is also possible to have multiple dimensions

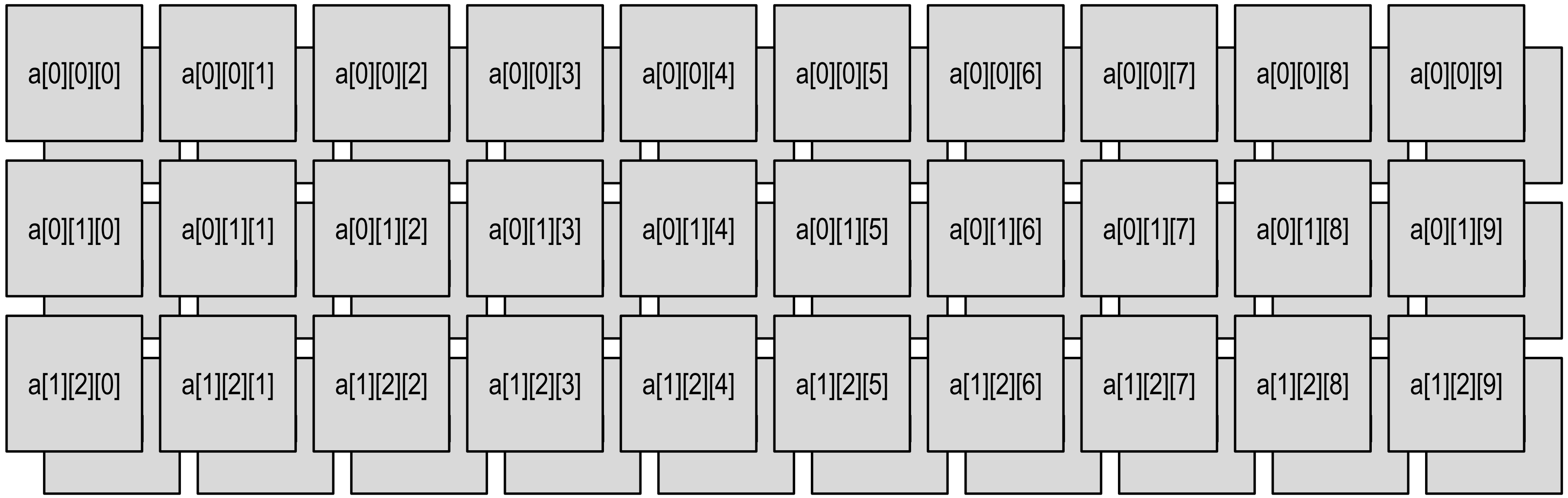
```
int[][] a = new int[3][10];
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]	a[0][6]	a[0][7]	a[0][8]	a[0][9]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]	a[1][6]	a[1][7]	a[1][8]	a[1][9]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2][5]	a[2][6]	a[2][7]	a[2][8]	a[2][9]

Multi-dimensional arrays

It is also possible to have multiple dimensions

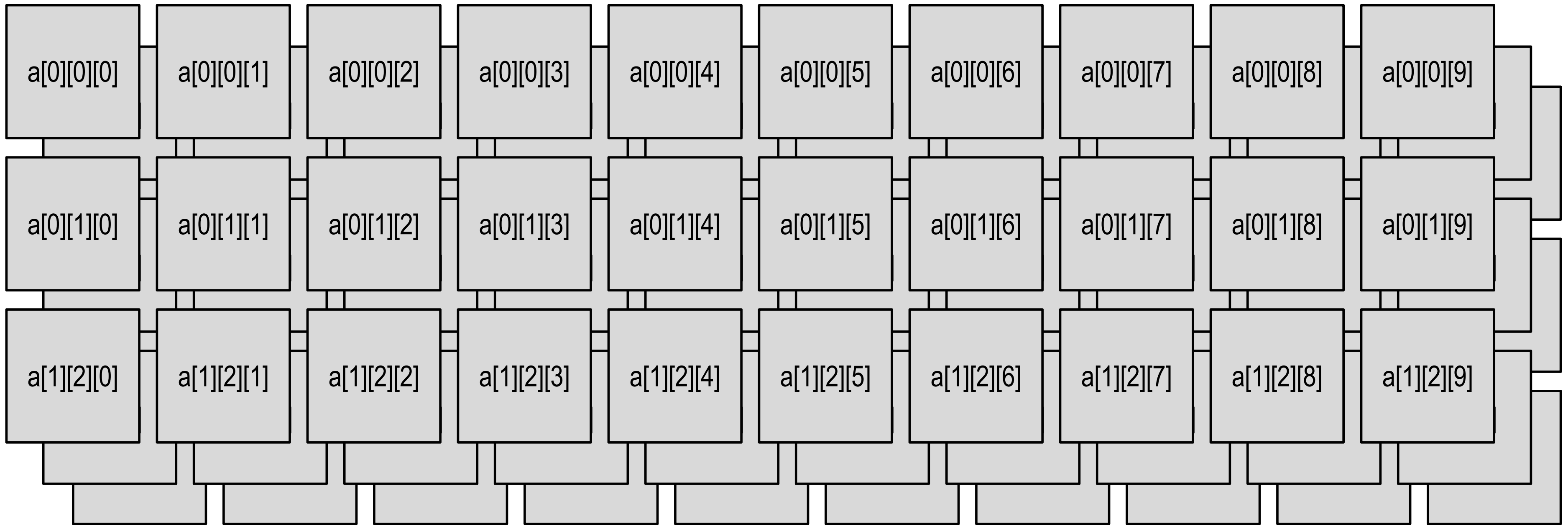
```
int[][][] a = new int[2][3][10];
```



Multi-dimensional arrays

It is also possible to have multiple dimensions

```
int[][][] a = new int[3][3][10];
```



Multi-dimensional arrays

To initialise a two-dimensional array:

```
int[][] a = { { 1, 2, 3, 4 },
              { 5, 6, 7, 8 },
              { 9, 8, 7, 6 } };
```

a[0][0] = 1	a[0][1] = 2	a[0][2] = 3	a[0][3] = 4
a[1][0] = 5	a[1][1] = 6	a[1][2] = 7	a[1][3] = 8
a[2][0] = 9	a[2][1] = 8	a[2][2] = 7	a[2][3] = 6

Multi-dimensional arrays

To initialise a two-dimensional array:

```
int[][] a = { { 1, 2, 3, 4 },
              { 5, 6, 7, 8 },
              { 9, 8, 7, 6 } };
```

We could also have written:

```
int[][] a = new int[3][4];
a[0][0]=1; a[0][1]=2; a[0][2]=3; a[0][3]=4;
a[1][0]=5; a[1][1]=6; a[1][2]=7; a[1][3]=8;
a[2][0]=9; a[2][1]=8; a[2][2]=7; a[2][3]=6;
```

Multi-dimensional arrays

To initialise a three-dimensional array:

```
int[][] a =
   { { { 1, 2, 3, 4 },
       { 5, 6, 7, 8 },
       { 9, 8, 7, 6 } },
     { { 5, 4, 3, 2 },
       { 1, 2, 3, 4 },
       { 5, 6, 7, 8 } } };
```

Remember this?

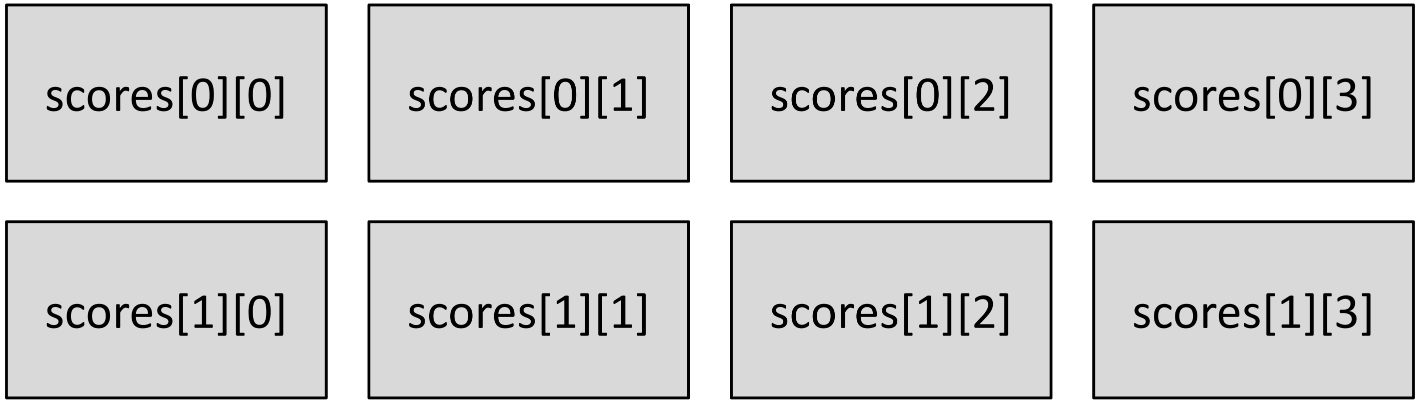
LabScoreAnalysis.java

```
/*
 *  This program analyses student lab scores.
 *  It calculates the lowest, highest and mean scores.
 */
public class LabScoreAnalysis {
   public static void main(String[] args) {
      double[] scores; // lab scores
      int students;    // number of students
      double lowest;   // lowest score
      double highest;  // highest score
      double mean;     // mean score

      // first, find out how many students there are
      System.out.println("How many students?");
      Scanner scan = new Scanner(System.in);
      students = scan.nextInt();

      // make the scores array the correct size
      scores = new double[students];
```

Multi-dimensional example

We're going to extend the previous example to handle multiple labs:

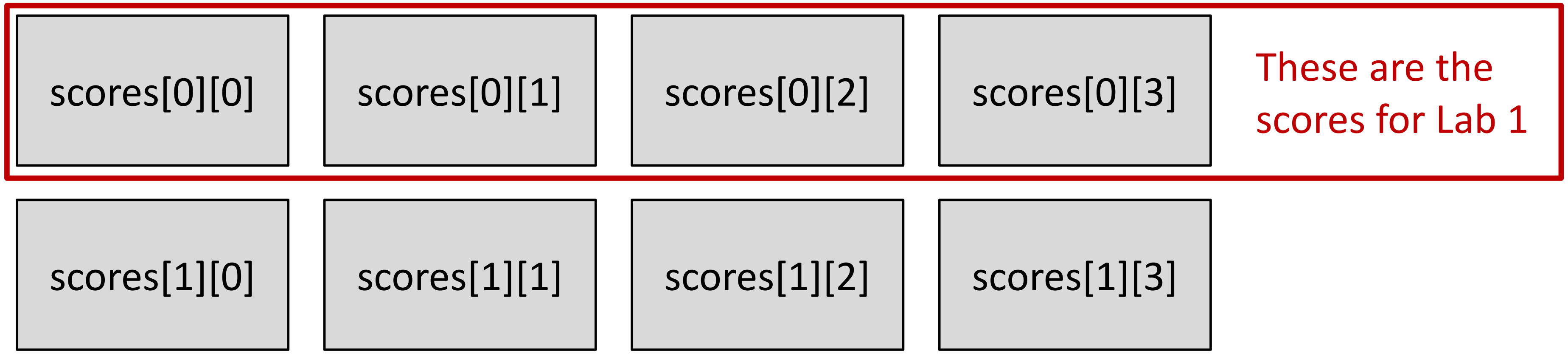
```
double[][] scores = new double[labs][studs];
```



Multi-dimensional example

We're going to extend the previous example to handle multiple labs:

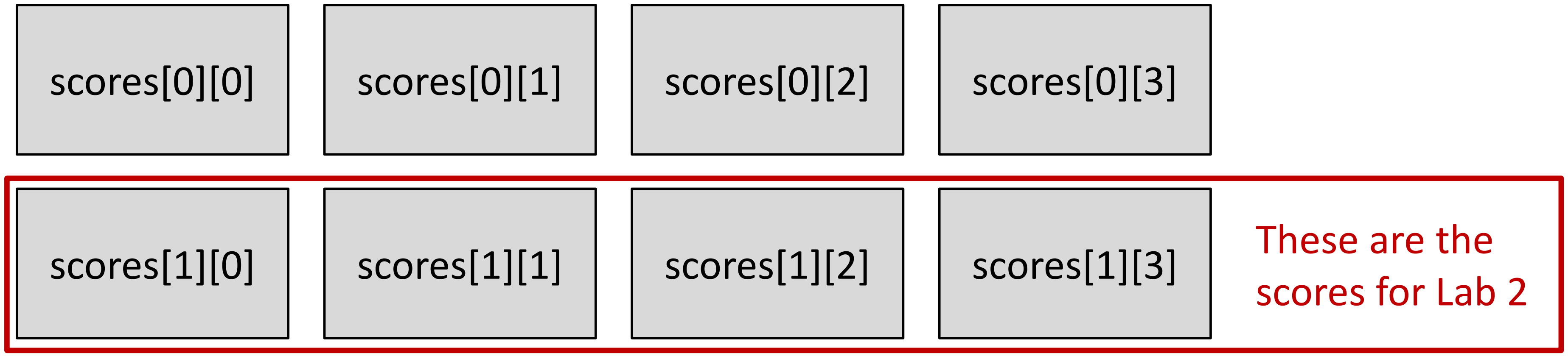
```
double[][] scores = new double[labs][studs];
```



Multi-dimensional example

We're going to extend the previous example to handle multiple labs:

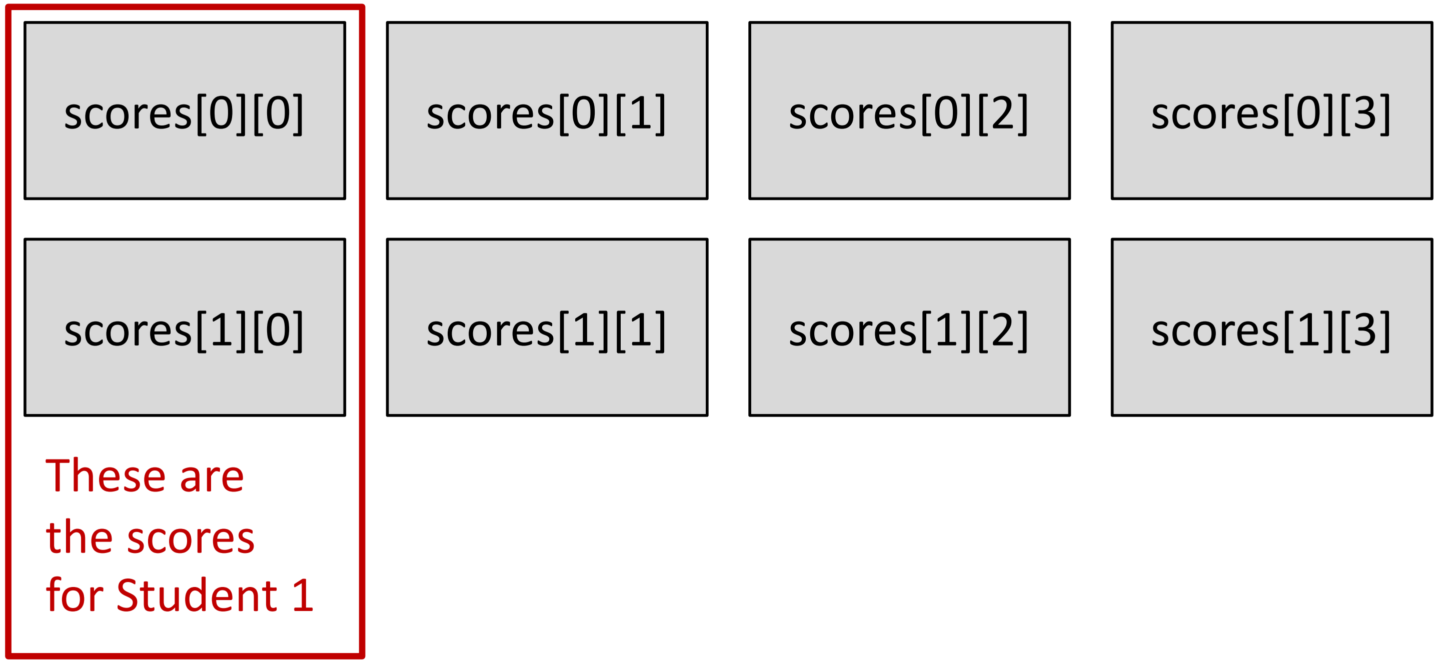
```
double[][] scores = new double[labs][studs];
```



Multi-dimensional example

We're going to extend the previous example to handle multiple labs:

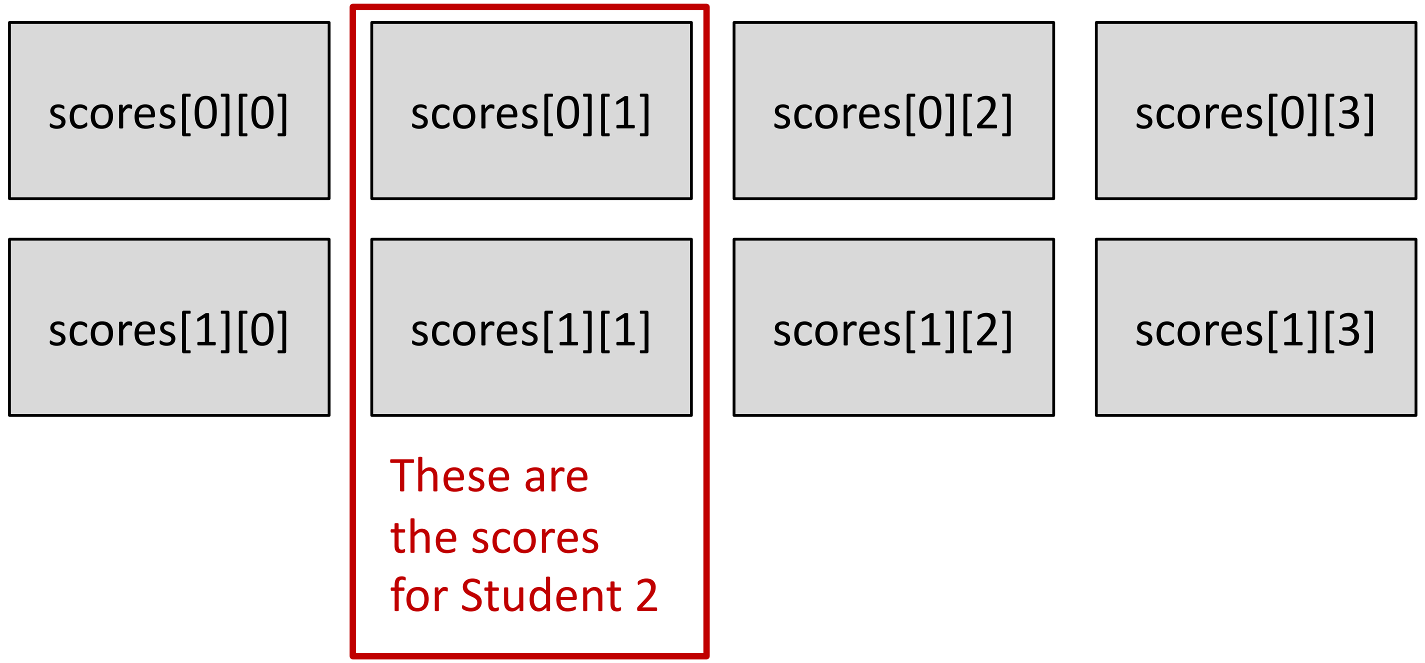
```
double[][] scores = new double[labs][studs];
```



Multi-dimensional example

We're going to extend the previous example to handle multiple labs:

```
double[][] scores = new double[labs][studs];
```



Multi-dimensional example

LabScoresAnalysis.java

```
public class LabScoresAnalysis {
    public static void main(String[] args) {
        double[][] scores; // lab scores
        int labs;          // number of labs
        int students;      // number of students
        double lowest;     // lowest score
        double highest;    // highest score
        double mean;       // mean score

        // find out how many students there are
        System.out.println("How many students?");
        Scanner scan = new Scanner(System.in);
        students = scan.nextInt();

        // find out how many labs there are
        System.out.println("How many labs?");
        labs = scan.nextInt();

        // make the scores array the correct size
        scores = new double[labs][students];
```

Multi-dimensional example

```
// read scores from user
for(int lab=0; lab<labs; lab++) {
    System.out.println("Input scores for lab"
                      +(lab+1)+":");
    for(int stud=0; stud<students; stud++)
        scores[lab][stud] = scan.nextDouble();
}

// find mean score for each lab
for(int lab=0; lab<labs; lab++) {
    mean = 0;
    for(int stud=0; stud<students; stud++)
        mean += scores[lab][stud];
    mean /= students;
    System.out.println("Mean for lab "
                       +(lab+1)+" is "+mean);
}
```

Multi-dimensional example

```
        // find mean score for each student
        for(int stud=0; stud<students; stud++) {
            mean = 0;
            for(int lab=0; lab<labs; lab++)
                mean += scores[lab][stud];
            mean /= students;
            System.out.println("Mean for student "
                                        +(stud+1)+" is "+mean);
        }
    }
}
```

For conciseness, the lowest and highest scores for each lab and student aren't calculated. I'll leave this as an exercise for you to try out outside the lecture.

Any Questions?

Methods (recap)

In Java, methods are sub-programs that can be **called** from other parts of the program

- You can pass inputs (or **arguments**) to them and they can **return** a result
- Their **signature** declares their arguments and their return type
- They can't see variables declared in other methods, such as main (because of **scope**)
- **`main`** is a special method that always gets called when the program is first run

Methods (recap)

A very simple example:

```
public class MethodDemo {
   public static void main(String[] args) {
      int x = 1;
      int y = addOne( x );
      System.out.println(x + "+1= " + y);
   }
   static int addOne( int input ) {
      return  input + 1;
   }
}
```

Terminal

```
1+1= 2
```

Methods (recap)

"addOne" is the method's name

```
public class MethodDemo {
  public static void main(String[] args) {
     int x = 1;
     int y = addOne( x );
     System.out.println(x + "+1= " + y);
  }
  static int addOne( int input ) {
     return  input + 1;
  }
}
```

Terminal

```
1+1= 2
```

Methods (recap)

This is how it is **called** (i.e. executed)

```
public class MethodDemo {
   public static void main(String[] args) {
      int x = 1;
      int y = addOne( x );
      System.out.println(x + "+1= " + y);
   }
   static int addOne( int input ) {
      return  input + 1;
   }
}
```

Terminal

```
1+1= 2
```

Methods (recap)

This specifies the input (**argument**) it expects

```
public class MethodDemo {
   public static void main(String[] args) {
      int x = 1;
      int y = addOne( x );
      System.out.println(x + "+1= " + y);
   }
   static int addOne( int input ) {
      return  input + 1;
   }
}
```

Terminal

```
1+1= 2
```

Methods (recap)

This is how the argument is **passed**

```
public class MethodDemo {
   public static void main(String[] args) {
      int x = 1;
      int y = addOne( x );
      System.out.println(x + "+1= " + y);
   }
   static int addOne( int input ) {
      return  input + 1;
   }
}
```

Terminal

```
1+1= 2
```

Methods (recap)

This is how its **return type** is specified

```
public class MethodDemo {
   public static void main(String[] args) {
      int x = 1;
      int y = addOne( x );
      System.out.println(x + "+1= " + y);
   }
   static int addOne( int input ) {
      return  input + 1;
   }
}
```

Terminal

```
1+1= 2
```

Methods (recap)

This is how it **returns** a value

```
public class MethodDemo {
  public static void main(String[] args) {
     int x = 1;
     int y = addOne( x );
     System.out.println(x + "+1= " + y);
  }
  static int addOne( int input ) {
     return  input + 1;
  }
}
```

Terminal

```
1+1= 2
```

Methods (recap)

And this is how you use the returned value

```
public class MethodDemo {
   public static void main(String[] args) {
      int x = 1;
      int y = addOne( x );
      System.out.println(x + "+1= " + y);
   }
   static int addOne( int input ) {
      return  input + 1;
   }
}
```

Terminal

```
1+1= 2
```

Methods (recap)

addOne can't see variables in main

```
public class MethodDemo {
   public static void main(String[] args) {
      int x = 1;
      int y = addOne( x );
      System.out.println(x + "+1= " + y);
   }
   static int addOne( int input ) {
      return  x + 1;
   }
}
```

Terminal

```
error: cannot find symbol x
```

Methods (recap)

This is known as the method's signature

```
public class MethodDemo {
   public static void main(String[] args) {
      int x = 1;
      int y = addOne( x );
      System.out.println(x + "+1= " + y);
   }
   static int addOne( int input ) {
      return  input + 1;
   }
}
```

Terminal

```
1+1= 2
```

Quiz

Will this work?

```
public class Q6 {
   public static void main(String[] args) {
      int[] as = {1,2,3,4,5};
      int c = sum(as);
   }
   static int sum(int[] as) {
      int total = 0;
      for(int a : as) total += a;
      return total;
   }
}
```

Quiz

Will this work?

```
public class Q6 {
   public static void main(String[] args) {
      int[] as = {1,2,3,4,5};
      int c = sum(as);
   }
   static int sum(int[] as) {
      int total = 0;
      for(int a : as) total += a;
      return total;
   }
}
```

Yes, this is all fine. An array can be passed to a method.

Quiz

Will this work?

```
public class Q7 {
   public static void main(String[] args) {
      int[] as = {1,2,3,4,5};
      int[] c = summarise(as);
      System.out.println("Sum: "+c[0]+" Prod: "+c[1]);
   }
   static int[] summarise(int[] as) {
      int[] summary = {0,1};
      for(int a : as) {
         summary[0] += a; // array sum
         summary[1] *= a; // array product
      }
      return summary;
   }
}
```

Quiz

Will this work?

```
public class Q7 {
   public static void main(String[] args) {
      int[] as = {1,2,3,4,5};
      int[] c = summarise(as);
      System.out.println("Sum: "+c[0]+" Prod: "+c[1]);
   }
   static int[] summarise(int[] as) {
      int[] summary = {0,1};
      for(int a : as) {
         summary[0] += a; // array sum
         summary[1] *= a; // array product
      }
      return summary;
   }
}
```

Yes, this is all fine. A method can return an array.
This is one way of returning multiple values from a method.

Arrays and Methods

Arrays can be used as arguments and return types of methods, but you have to be careful!

- If you're just **reading** from an array you **pass into** a method, then everything will work fine
- If you're **returning** an array **created within** a method, then this will also be fine
- But if a method **modifies** an array **passed in** as an argument, then things get more interesting

Digression: Object types

In Week 2, Lecture 1, you heard there are 2 kinds of variable: **primitive types** and **object types**

- So far, the distinction hasn't been important
- However, arrays are object types, and you need to know more about them before you can fully use arrays in the context of methods
- The distinction between these two types lies in how they are stored in memory…

Digression: Object types

Variables of primitive type (`int`, `double` etc.) are basically locations in a computer's memory where a single value is stored

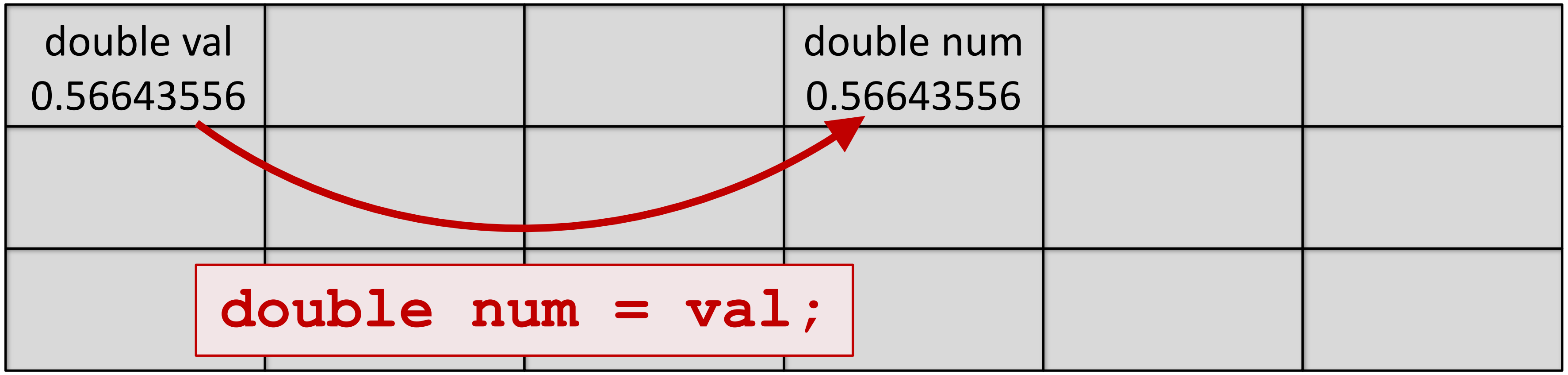
A block of memory,
somewhere in your computer

double val 0.56643556					
	int count 42			boolean b true	
			char initial 'M'		

Digression: Object types

Variables of primitive type (`int`, `double` etc.) are basically locations in a computer's memory where a single value is stored

- When you copy a primitive variable, the value it contains is copied to the new variable

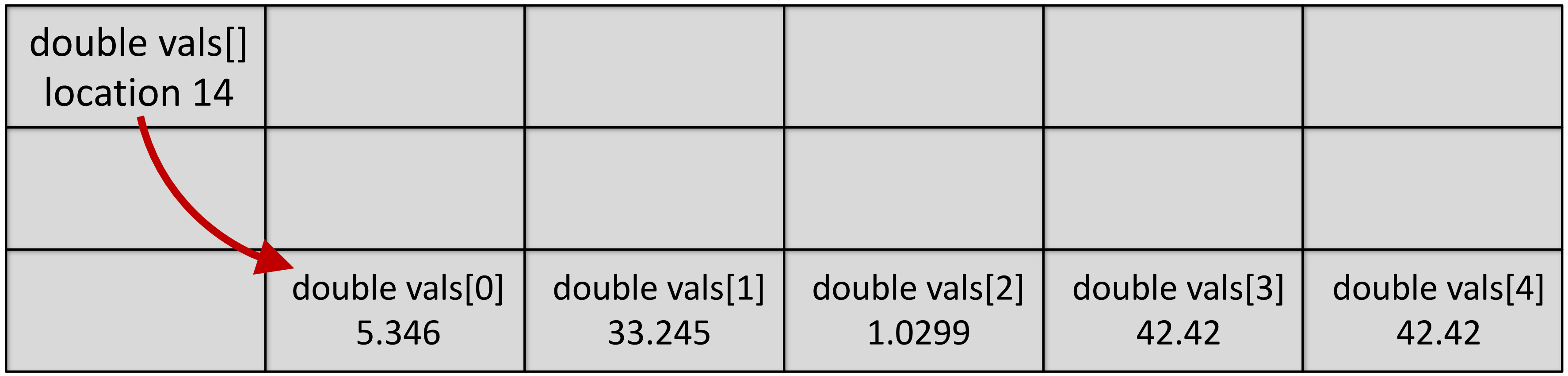


```
double num = val;
```

Digression: Object types

Object types, such as arrays, are different

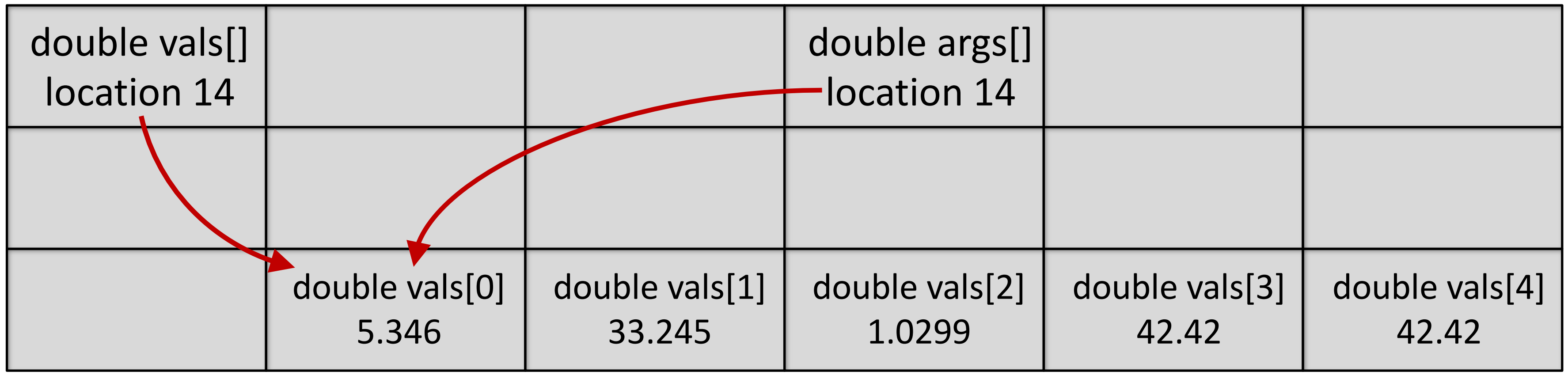
- An object variable holds an address, which points to the memory location where its contents are actually stored, elsewhere in memory
- In other languages, object types are called **pointers**



Digression: Object types

When you copy an object variable, it actually copies this address, not the object's contents

- Which means that the copy actually points to the same object. This can be a source of confusion!
- E.g. `double[] args = vals;`



Copying array variables

Here's an example:

CopyingArrays.java

```java
public class CopyingArrays {
   public static void main(String[] args) {
      String[] array1 = {"Hello","there"};
      String[] array2 = array1;
      array2[0] = "Goodbye";
      System.out.println(array1[0]+" "+array1[1]);
      System.out.println(array2[0]+" "+array1[1]);
   }
}
```

Terminal

```
$ java CopyingArrays
Hello there
Goodbye there
```

– or –

?

Terminal

```
$ java CopyingArrays
Goodbye there
Goodbye there
```

Copying array variables

Here's an example:

CopyingArrays.java

```
public class CopyingArrays {
   public static void main(String[] args) {
      String[] array1 = {"Hello","there"};
      String[] array2 = array1;
      array2[0] = "Goodbye";
      System.out.println(array1[0]+" "+array1[1]);
      System.out.println(array2[0]+" "+array1[1]);
   }
}
```

Terminal

```
$ java CopyingArrays
Hello there
Goodbye there
```

✗

Terminal

```
$ java CopyingArrays
Goodbye there
Goodbye there
```

Copying array variables

Here's an example:

CopyingArrays.java

```
public class CopyingArrays {
   public static void main(String[] args) {
      String[] array1 = {"Hello","there"};
      String[] array2 = array1;
      array2[0] = "Goodbye";
      System.out.println(array1[0]+" "+array1[1]);
      System.out.println(array2[0]+" "+array1[1]);
   }
}
```

`array2` now points to the same structure in memory as `array1`, so anything done to `array2` will affect `array1`, and vice versa

Terminal

```
$ java CopyingArrays
Goodbye there
Goodbye there
```

Passing arrays

When you pass an argument to a method, you are copying one variable to another

- You are copying a variable in the calling method into a new **local variable** in the called method
- This means that if you pass an array to a method, the method receives the same array object, and any changes you make to it will affect the original copy

Passing arrays

PassingArrays.java

```
public class PassingArrays {
    public static void main(String[] args) {
        double[] vals = {1,2,3,4,5};
        square(vals);
        print(vals);
    }
    // square each element of an array
    static void square(double[] nums) {
        for(int i=0; i<nums.length; i++)
            nums[i] = nums[i] * nums[i];
    }
    // print each element of an array
    static void print(double[] array) {
        for(double d : array)
            System.out.print(d+" ");
    }
}
```

Terminal

```
1.0 4.0 9.0 16.0 25.0
```

Passing arrays

PassingArrays.java

```
public class PassingArrays {
    public static void main(String[] args) {
        double[] vals = {1,2,3,4,5};
        square(vals);
        print(vals);
    }
    // square each element of an array
    static void square(double[] nums) {
        for(int i=0; i<nums.length; i++)
            nums[i] = nums[i] * nums[i];
    }
    // print each element of an array
    static void print(double[] array) {
        for(double d : array)
            System.out.print(d+" ");
    }
}
```

In effect, square changes the values of vals in main.
This is different to what happens with primitives.

Terminal

```
1.0 4.0 9.0 16.0 25.0
```

Deep copying

PassingArraysDeep.java

```java
public class PassingArraysDeep {
   public static void main(String[] args) {
      double[] vals = {1,2,3,4,5};
      double[] squared = square(vals);
      print(vals);
      System.out.println();
      print(squared);
   }

   // square each element of an array
   static double[] square(double[] nums) {
      double[] copy = new double[nums.length];
      for(int i=0; i<nums.length; i++)
         copy[i] = nums[i] * nums[i];
      return copy;
   }
   …
```

Terminal

```
1.0 2.0 3.0 4.0 5.0
1.0 4.0 9.0 16.0 25.0
```

Deep copying

PassingArraysDeep.java

```java
public class PassingArraysDeep {
    public static void main(String[] args) {
        double[] vals = {1,2,3,4,5};
        double[] squared = square(vals);
        print(vals);
        System.out.println();
        print(squared);
    }

    // square each element of an array
    static double[] square(double[] nums) {
        double[] copy = new double[nums.length];
        for(int i=0; i<nums.length; i++)
            copy[i] = nums[i] * nums[i];
        return copy;
    }

    …
```

To avoid this, you can create a new array, and copy the original array into this item by item (after some manipulation, in this example). This is known as **deep copying**.

Terminal

```
1.0 2.0 3.0 4.0 5.0
1.0 4.0 9.0 16.0 25.0
```

Deep copying

Whether you want to deep copy or shallow copy depends upon the context

- In this example, deep copying preserves the original information, allowing it to be reused
- If you don't need the original information any more, then shallow copying is more efficient
- This is a brief introduction to this issue. It will come up again when you start to do object-oriented programming

An exception: Strings

Strings are also variables of object type

- However, in Java they behave more like primitives
- This is an intentional design choice: a result of careful design of the String class and the Java language
- It means you don't have to worry whether you are shallow or deep copying strings when you use them
- This is not the case in some other languages which you may come across, e.g. C

Next Lecture

- Review of Part 1 and looking ahead to Part 2