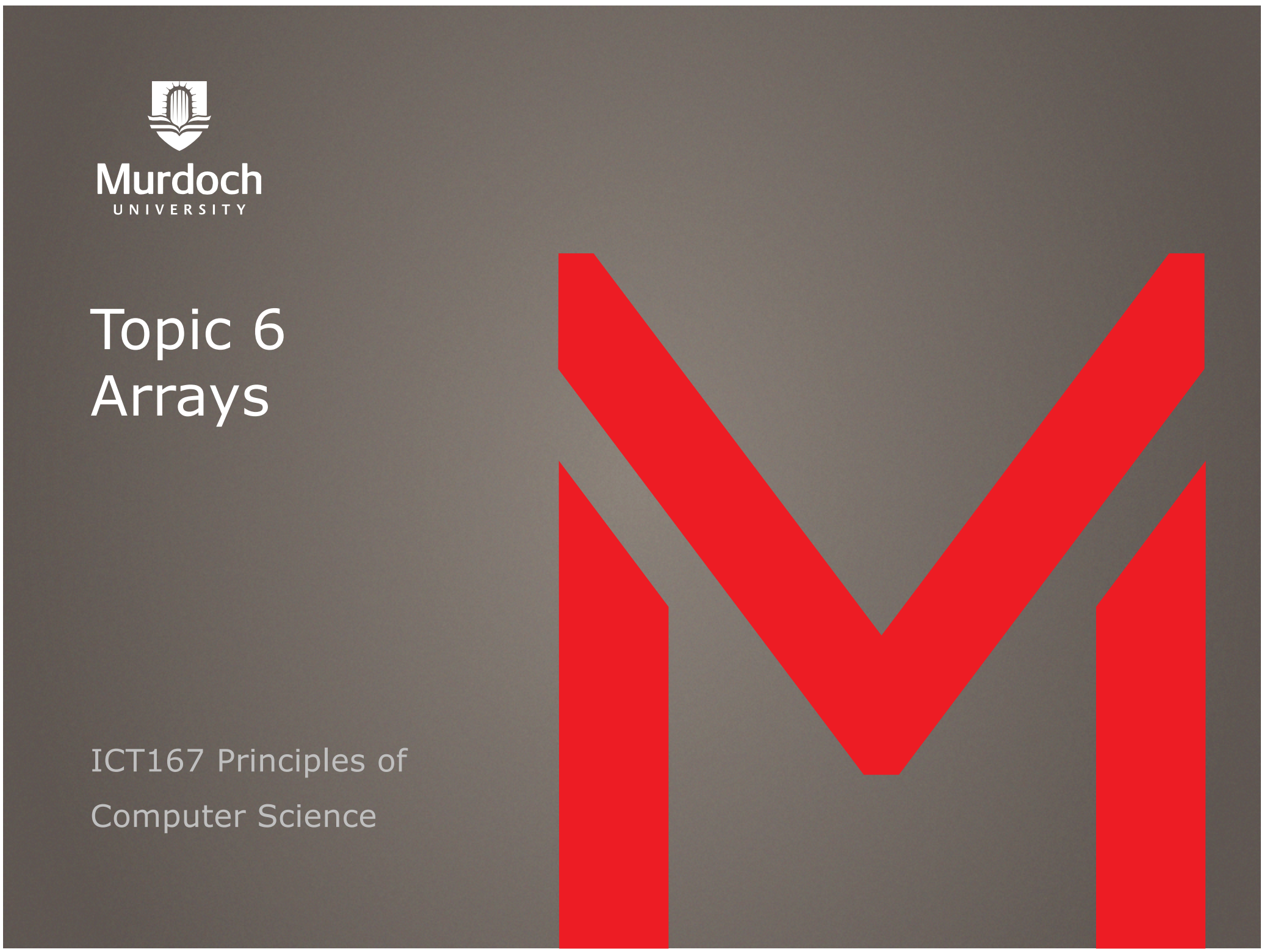


Topic 6
Arrays

ICT167 Principles of
Computer Science

Murdoch UNIVERSITY

Objectives

- Know how to declare, create and use **arrays in Java**
- Use the **length** instance variable to ensure that array indexes remain in bounds
- Be able to **pass array element values as parameters**
- Be able to **pass arrays as parameters**
- Be able to write **methods which return arrays**

Objectives

- Be able to include **arrays as instance variables**
- Be able to declare, create and initialise **arrays of objects**
- Understand == on arrays
- Be able to implement simple **selection algorithms (searching an array)**
  - Sequential search
  - Binary search

Objectives

- Be able to implement simple **sorting algorithms (sorting an array)**
  - Bubble sort
  - Selection sort
  - Insertion sort
- Be able to define and use **multi-dimensional arrays**

Reading

Savitch: Chapter 7

Arrays in Programming Languages

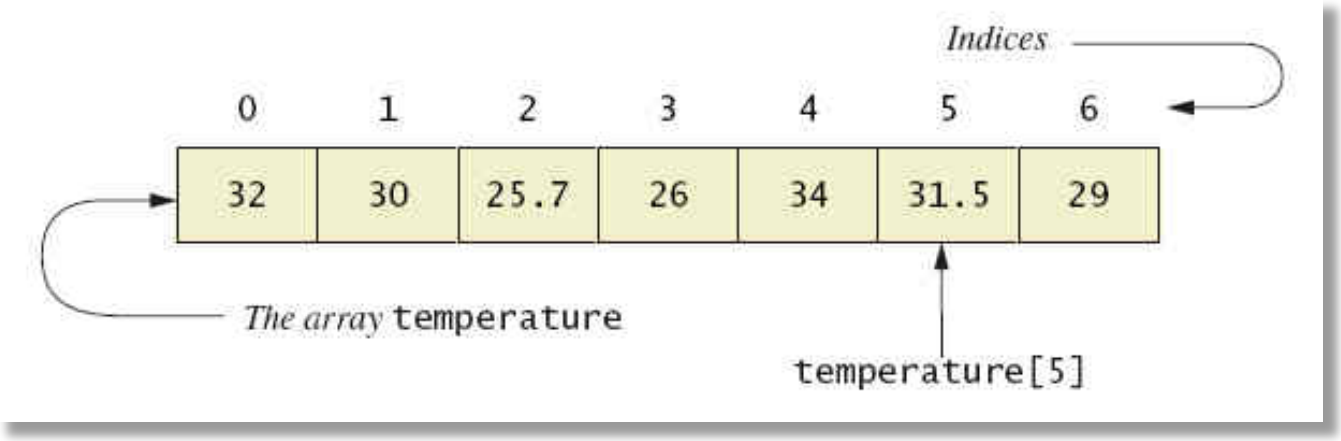
- An array consists of a systematically organised and named sequence of similar variables - called the **elements** of the array
  - That is, it is a single name for a collection of data values, all of the same type
- The elements are numbered: 0, 1, 2, … and so on, called the **index** (or subscript)
- An array is used in place of a lot of separate variables (which are of the same type)

Arrays in Programming Languages

- An array can be small with only 2 or 3 elements (or even zero), or it can be very large with thousands of elements
- An array is an *ordered collection* of data items
- Each item has a position (or index)
- Each item (except first item) has a unique predecessor
- Each item (except last item) has a unique successor

Visualize Array

- Figure 7.1 A common way to visualize an array



- Note sample program, listing 7.1
  `class ArrayOfTemperatures`

Murdoch UNIVERSITY

Arrays in Programming Languages

- An array is a *direct access* data structure
- Each item is accessible without going through any other item
- Arrays are the most frequently used data structure

Advantages / Uses of Arrays

- Advantages:
  - Saves thinking up the names for a lot of variables
  - Easy to change/control how many there are
  - Can process them systematically
  - Can be efficiently stored
  - Idea is common to nearly all programming languages
  - In some programming languages, can pass the whole array full of values around to procedures and back

Advantages / Uses of Arrays

- **Restrictions:**
  - Each item in an array must be of the same type
- **Variations between programming languages:**
  - Run-time changing of array size allowed?
  - Checking of bounds?
  - Passing to procedures/methods allowed?
  - Returning from procedures allowed?
  - Pass by reference or value?
  - Arrays of objects or just primitive values?
  - Multi-dimensional?

Creating Arrays in Java

- General syntax for declaring an array:

```
BaseType[] ArrayName= new BaseType[Length];
```

- Examples:

```
// 80-element array with base type char
char[] symbols = new char[80];
// 100-element array of doubles:
double[] readings = new double[100];
//100-element array of Species:
Species[] specimen = new Species[100];
```

Creating Arrays in Java

- Length of an array is specified by the number in brackets when it is created with ***new***
  - it determines the amount of memory allocated for the array elements (values)
  - it determines the *maximum* number of elements the array can hold
  - storage is allocated whether or not the elements are assigned values

Creating Arrays in Java

- The array length is established when the array is created
  - It is automatically stored in the (read-only) instance variable length, and cannot be changed
- An array is a special kind of object in Java
- Eg: declare an array of ints:

```
int[] mark;

// mark is now an “array of int” type variables, with
// null reference
```

Creating Arrays in Java

- Create an array of int “objects” of a certain length:

```
mark = new int[7];
```

// the variable mark now refers to an array of seven ints
// each one initialised to the default int value of zero

- OR, declare and create:

```
int[] mark = new int[7];
```

- Data can now be stored in the array as:

```
mark[0] = 85;
```

Creating Arrays in Java

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
mark:	85	70	50	62	39	92	54

Creating Arrays in Java

- Later on may be make `mark` refer to another array object:

```
mark = new int[9];
```

  - Note: This does not change the size of an array
  - It just makes the variable refer to a new, bigger array
  - The old array is still there, the same size as before, elsewhere in memory

Creating Arrays in Java

- Do somethings with the first element in the array:

```
mark[0]= 30;
System.out.println(mark[0]);
mark[0]= keyboard.nextInt();
System.out.println(mark[4+1-5]);
```

- Do somethings with the last element in the array:

```
mark[8]= 72;
System.out.println(mark[8]);
```

Creating Arrays in Java

- Output all elements of the array:

```
for (int i=0; i<9; i++)
  System.out.println(mark[i]);
```

- Note that array subscripts use zero-numbering. That is:
  - the first element has subscript 0
  - the second element has subscript 1
  - the nth element has subscript n-1
  - the last element has subscript length-1

Array Lengths

- Arrays are sort of objects and have a publically accessible (but read-only) instance variable **length**, which gives the number of elements in the array. Eg:

```
mark.length
```

- So a very common form of loop is:

```
for(index=0;index<mark.length;index++) {
  System.out.println("Enter the "
                     + index + "th mark:");
  mark[index] = keyboard.nextInt();
}
```

Array Lengths

- Note: this form of loop is often used to give initial values to the elements of an array but there is a special way of initialising a small array with a list of values
- This is done at the time the array is declared:

```
int[] coins = {5,10,20,50,100,200};
// implied length = 6
double[] data = {4.5,4.7,5.1,5.5};
// implied length = 4
```

Array Lengths

- `Subscript out of Range Error:`
  - *Note that if the index (inside the [brackets]) evaluates to less than zero, or greater than or equal to the length of the array, then the program will suffer a runtime error:*

    `ArrayIndexOutOfBoundsException`

Example

```
/** Read temperatures from the user and shows which are
 above and which are below the average of all the
 temperatures. From Savitch Listing 7.1 */
import java.util.Scanner;
public class ArrayOfTemperatures2 {
  public static void main(String[] args) {
     Scanner keyboard = new Scanner(System.in);
     System.out.println("How many temperatures
                                    do " + "you have?");
     int size = keyboard.nextInt();
     double[] temperature = new double[size];
```

Example

```
// Read temp's and compute their average
double sum, average;
System.out.println("Enter " + temperature.length + "
                        temperatures:");
sum = 0;
for(int i = 0;i<temperature.length;i++) {
   temperature[i] = keyboard.nextDouble();
   sum = sum + temperature[i];
}// end for
average = sum/temperature.length;
System.out.println("Avg temp is " + average);
```

Example

```
// Display each temp + its relation to avg
System.out.println("The temperatures are");
for (int i = 0;i<temperature.length;i++) {
   if (temperature[i] < average)
      System.out.println(temperature[i] + "
                                    below average.");
   else if (temperature[i] > average)
      System.out.println(temperature[i] + "
                                    above average.");
   else //temperature[i] == average
      System.out.println(temperature[i] + " the
                                        average.");
}// end for
```

Example

```
        System.out.println("Have a nice week.");
    }// end main
}// end class
```

Arrays as Parameters

Passing/Returning Arrays of Primitive types to/from methods

- Passing an array element is like passing any other variable. Eg:

```
double d = Math.sqrt(mark[2]);
// Primitive passed by value: the value cannot be
    // changed by the method
```

Arrays as Parameters

- Passing a whole array is also possible:

```
public static double avg(int[] arr) {
  int total = 0;
  for (int i=0; I < arr.length; i++)
     total = total + arr[i];
  return (double)total/arr.length;
}
```

Arrays as Parameters

- The caller then uses:

```
System.out.print("The average mark is ");
System.out.println(avg(mark));
```

- In this example, the array `mark` is passed as parameter to the method `avg` which returns the average value of numbers stored in the array
- Within the method `avg`, the array is referred to with the name `arr`

Arrays as Parameters

- *Note that the [brackets] appear in the definition of the method and not in the method call*
- Also note that passing a whole array to a method is done via **pass by reference** (as with other objects) and so that the method is allowed to change the values in the array
- This is useful (for example, if you want to re-arrange or sort the values in the array) but be careful

Example: ReturnArrayDemo

```
/** Program to demonstrate a method returning an
    array */
import java.util.Scanner;
public class ReturnArrayDemo {
  public static void main(String arg[]) {
     Scanner keyboard = new Scanner(System.in);
     System.out.println("Enter score on exam 1:");
     int firstScore = keyboard.nextInt();
     int[] nextScore = new int[3];
     int i;
     for (i = 0; i < nextScore.length; i++)
        nextScore[i] = firstScore  +  5 * i;
```

Example: ReturnArrayDemo

```
        double[] averageScore;
        double averageScore =
                averageArray(firstScore, nextScore);
        for (i = 0; i < nextScore.length; i++) {
            System.out.println("If your score on
                                exam 2 is " + nextScore[i]);
            System.out.println("Your average will be
                                        " + averageScore[i]);
        }
    } // end main
```

Example: ReturnArrayDemo

```
  public static double[] averageArray(int
                        firstScore, int[] nextScore) {
     double[] temp=new double[nextScore.length];
     for (int i = 0; i < temp.length; i++)
       temp[i]=average(firstScore,nextScore[i]);
     return temp;
  } // end averageArray
  public static double average(int n1, int n2){
     return (n1 + n2)/2.0;
  } // end average
} // end class ReturnArrayDemo
```

Arrays in Objects

- It is permitted and is very common to have arrays as instance variables in objects
- Eg:
  - a student might have an array of marks
  - a tour operator has an array of guides
  - an alphabet has an array of letters
  - a letter has an array of dots (for printing)
  - a questionnaire has an array of questions
  - a polygonal shape has an array of vertices
  - etc.

Arrays in Objects

- Eg: in the class Student:

```
private String familyName;
private String[] otherNames;
private long studentNumber;
private int[] componentMarks;
private char grade;
```

Objects in Arrays

- It is permitted and is very common to have arrays of objects (as opposed to primitive values)
- Eg:
  - an array of guides
  - an array of products
  - an array of questions
  - an array of vertices
  - an array of students
  - etc.

Objects in Arrays

- Eg: suppose that we have a class Student available. A client can use this as follows:

```
Student[] pupil = new Student[50];
// declares a Student array type variable pupil,
// creates a Student array object of size 50 and makes
// pupil refer to this object
```

- Now

```
pupil[0], pupil[1], ..., pupil[49]
```

are all variables that can refer to Student objects

Remember: Initialize Elements

- But beware!!

```
pupil[0], pupil[1], ..., pupil[49]
```

// are all **null reference** variables: none of them
// actually refer to any Student objects yet

- If we called:

```
pupil[0].writeDetails();
```

- or even:

```
pupil[0].enterDetails();
```

- then we would get a **`NullPointerException`**

Remember: Initialize Elements

- Before calling any methods on Student objects you have to create some Student objects. Eg:

```
for (int i=0; i<pupil.length; i++)
pupil[i] = new Student();
// that is using the default constructor from the
// Student class for each of the 50 objects
```

- Then you can call:

```
for (int i=0; i<pupil.length; i++){
System.out.println("Next student's details.");
pupil[i].enterDetails();
}
```

Example: Student Class

```
// Student.java
// example Student class with an array of marks
import java.util.Scanner;
public class Student {
  private String name;
  private int[] mark;
  public Student() {
     name = "No name yet.";
     mark = null;
  }
```

Example: Student Class

```
public void enterDetails() {
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Name:");
    name = keyboard.nextLine();
    System.out.print("Number of components:");
    int noc = keyboard.nextInt();
    mark = new int[noc];
    for(int i = 0;i < noc;i++) {
        System.out.println("Enter mark for " +(i+1)
                                    + "th component:");
        mark[i]= keyboard.nextInt();
    }
}
```

Example: Student Class

```
public void writeDetails() {
    System.out.println("Name:" + name);
    System.out.println("Number of components:"
                                        + mark.length);
    for(int i = 0;i < mark.length;i++)
       System.out.println("Mark for " + (i+1)
                          + "th component:" + mark[i]);
    System.out.println("That's all.");
}
```

Example: Student Class

```
  public int total() {
     int sum = 0;
      for(int i = 0;i < mark.length;i++)
        sum = sum + mark[i];
     return sum;
  }
}// end class Student
```

Example: Client Class

```
//StudentClient.java
// example of client program using the Student class
// and making an array of students
import java.util.Scanner;
public class StudentClient {
   public static void main(String[] args) {
      // array of student objects, currently null
      Student[] pupil;
      // array of total marks, currently null
      int[] tot;
      // array of letter grades, currently null
      char[] grade;
```

Example: Client Class

```
// The above arrays are an example of parallel
// arrays
int nops=0;
Scanner keyboard = new Scanner(System.in);
System.out.println("Welcome to class mark
                                        helper.");
System.out.println("Please enter number of
                                        pupils:");
// length of array
nops = keyboard.nextInt();
// create array of type Student; refer to Student
// objects
pupil = new Student[nops];
```

Example: Client Class

```
// create array of ints to store total marks
tot = new int[nops];
// create an array of chars to store grades
grade = new char[nops];
System.out.println("Enter students' details.");
for (int i = 0;i < pupil.length;i++) {
   //  very important !!!
   pupil[i] = new Student();
   pupil[i].enterDetails();
   tot[i] = pupil[i].total();
   System.out.println("Total mark for that"
                      + " student is "+ tot[i]);
}// end for
```

Example: Client Class

```
double avg = findClassAv(tot);
System.out.println("Pass mark is " + avg);
System.out.println();
System.out.println("Here's the results.");
for (int i = 0;i < pupil.length;i++) {
   if (tot[i] >= avg) grade[i] = 'P';
   else grade[i] = 'F';
   pupil[i].writeDetails();
   System.out.println("Final grade for that" +
                      " student is "+ grade[i]);
}
```

Example: Client Class

```
      System.out.println("Thank you, BYE.");
}//end of main
public static double findClassAv(int[] arr) {
   int sum = 0;
   for (int i =0;i < arr.length;i++)
     sum = sum + arr[i];
   return (double)sum/arr.length;
  }//end of findClassAv
}// end class StudentClient
```

Example: Output

```
/* Sample test run:
Welcome to class mark helper.
Please enter number of pupils: 2
Please enter students' details.
Name: J Blogg
Number of components: 3
Enter mark for 1th component: 60
Enter mark for 2th component: 70
Enter mark for 3th component: 80
Thank you.
The total mark for that student is 210
```

Example: Output

```
Name: A N Other
Number of components: 3
Enter mark for 1th component: 75
Enter mark for 2th component: 85
Enter mark for 3th component: 90
Thank you.
The total mark for that student is 250

Pass mark is 230.0
```

Example: Output

```
Here are the results.
Name: J Blogg
Number of components: 3
Mark for 1th component: 60
Mark for 2th component: 70
Mark for 3th component: 80
That's all.
The final grade for that student is F
```

Example: Output

```
Name: A N Other
Number of components: 3
Mark for 1th component: 75
Mark for 2th component: 85
Mark for 3th component: 90
That's all.
The final grade for that student is P

Thank you for using class mark helper.
*/
```

Checking Arrays for Equality

- Arrays are like objects as far as = = and != are concerned
- These equality tests will compare the memory addresses of two objects, not the data values that they hold
- If `marks` and `sums` are arrays of the same type, then you can ask:
- `if (marks == sums) ...`
- but they are only equal if both variables refer to the same object in memory

Checking Arrays for Equality

- Eg: if you created one object and made marks refer to it:

```
int[] marks = new int[10];
for (int i=0; i<10; i++)
  marks[i] = 10-i;
```

- and then you wrote:

```
int[] sums;
sums = marks;
```

- The == test will hold in the above case

Checking Arrays for Equality

- So, for example, you can have two arrays of ints of the same length with the same numbers in them but they are not equal (according to ==).
- To test two arrays for equality you need to define an equals method that returns true if and only if the arrays have the same length and all their corresponding values are equal
- The code below shows an example of an equals method

Checking Arrays for Equality

```
public static boolean equals(int[] a, int[] b) {
   boolean match = true;   // tentatively
   if (a.length != b.length) match = false;
   else {
      int i = 0;
      while (match && (i < a.length)) {
         if (a[i] != b[i])
            match = false;
         i++;
      }
   }
   return match;
}
```

Checking Arrays for Equality

- A method call:

```
boolean same = equals(marks, sums);
// where marks and sums are int arrays
```

Searching in Arrays

- There are many techniques for searching an array for a particular value
- Sequential search:
  - Start at the beginning of the array and proceed in sequence until either the value is found or the end of the array is reached
    - Or, just as easy, start at the end and work backwards toward the beginning
  - If the array is only partially filled, the search stops when the last meaningful value has been checked

Searching in Arrays

- It is not the most efficient method to search an array but it works and is easy to program
- Can be performed on both unsorted and sorted arrays

Searching in Arrays

```
public static boolean search(int[] a, int item){
   boolean found = false;
   if (a.length > 0) {
      int i = 0;
      while (!found) && (i < a.length)) {
         if (a[i] == item) found = true;
         i++;
      }
   }
   return found;
}
```

Searching in Arrays

- **A method call:**

```
boolean itemFound = search(array, num);
if (itemFound)
System.out.println ("The value " + num + "
                                exists in the array");
else
System.out.println ("The value " + num + "
                              is not found in the array");
```

Searching in Arrays

- It is common to have to find (or select) the maximum element in an array. Eg:

```
int indexOfMaxSoFar = 0;
for (int i = 1;i < arr.length;i++)
  if (arr[i] > arr[indexOfMaxSoFar])
     indexOfMaxSoFar = i;
indexOfMax = indexOfMaxSoFar;
```

Searching in Arrays

- Note this assumes that there is at least one element in the array
- Variations on this idea will find the minimum element, or the second biggest, etc.
- Notice that we make about 100 comparisons to find the maximum element in a list of 100 numbers, etc.

Sorting Algorithms

- It is also very common to have to sort a whole array of values into some order (numeric, alphabetical), including having to sort an array of complex objects into order according to a complex ordering relationship
  - Eg: sort by surname and then by given name (for people with the same surname, etc)
- This is a very important problem in computing and it has been well studied

Sorting Algorithms

- It can be complicated and there are several general approaches. Eg:
  - bubble sort
  - insertion sort
  - selection sort
  - quick sort
- It is good practice for a beginner programmer to make a working sorting program; it is not too hard

Sorting Algorithms

- However, the easiest approaches give inefficient programs
  - Eg: bubble sort might take 1 million moves to sort 1,000 entries
- If program speed is more important than programmer effort (and debugging time, etc.) then use quicksort which will take about 10,000 moves to sort 1,000 entries

Selection Sort

- One of many algorithms for sorting data items in ascending or descending order
- Selection sort method:
- Repeat
  - Find the *largest* item in the *unsorted* array
  - Swap it with the *last* item in the *unsorted* array
  - Reduce the unsorted array size by 1
- Until the array is sorted

Selection Sort

- Array to sort

Pass 1:	43	22	17	36	16
Pass 2:	16	22	17	36	43
Pass 3:	16	22	17	36	43
Pass 4:	16	17	22	36	43

Selection Sort

- In the above example, there are four passes needed to sort a list of five data items
- A variation on the above method is to find the *smallest* item in the unsorted array, swap it with the *first* item in the unsorted array, reduce the unsorted array size by 1
  - Repeat until the array is sorted

Selection Sort

```
// SelectSortV1.java
// To sort an array using Selection Sort
public class SelectSortV1 {
  public static void main( String[] args) {
     int[] anArray =
                {98,76,65,105,45,1,199,15,88,100};
     // sort the array in ascending order:
     SortArrayBySelection (anArray);
     // output the sorted numbers:
     System.out.println("Sorted numbers are:");
     for (int i = 0;i < anArray.length;i++)
        System.out.println( anArray[i]);
     System.out.println("End program - Bye.");
  } // end main
```

Selection Sort

```
public static void SortArrayBySelection(int[]
                                        arrayToSort) {
   int indexOfLargest, last, temp;
   for (last = arrayToSort.length-1;last >= 1;
                                            last--) {
      indexOfLargest = 0;
      // find index of largest in unsorted array
      for (int i = 1;i <= last;i++)
         if (arrayToSort[i] >
                        arrayToSort[indexOfLargest])
            indexOfLargest = i;
         // end if
      // end i for
```

Selection Sort

```
            // swap largest with last
            temp = arrayToSort[last];
            arrayToSort[last] =
                              arrayToSort[indexOfLargest];
            arrayToSort[indexOfLargest] = temp;
        } // end outer for
    } // end method SortArrayBySelection
}//end of class SelectSortV1
```

Selection Sort

```
/* Output
The sorted numbers are:
1
15
45
65
76
88
98
100
105
199
End of program - Bye.
*/
```

Insertion Sort

- Another example of an easy algorithm to sort an array of integers of into ascending order:

```
for i = 1 to arrayLength-1
  temp = arr[i]
  j = 0
  while (temp > arr[j])
     j = j+1
  end while
  for k=i downto j
     arr[k] = arr[k-1]
  end k for
  arr[j] = temp
end i for
```

Insertion Sort

- Each element is copied and inserted into the correct position in the array
- After the ith pass through the loop-body the first i+1 elements are in order
- During the ith pass, the value arr[i] is put in its right place amongst the first i+1 elements by finding the place and then moving all the rest of the sorted values along one place: that is, arr[i] is inserted in its right place

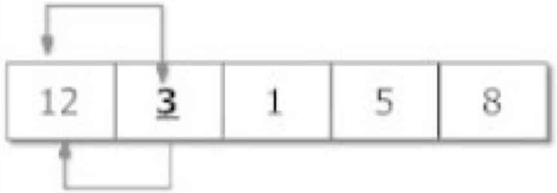
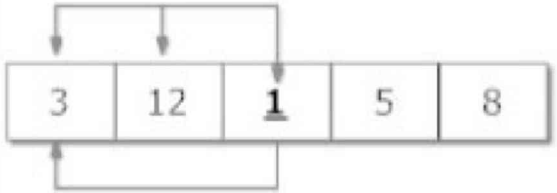
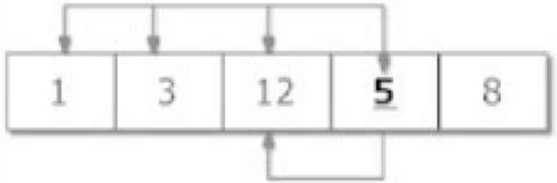
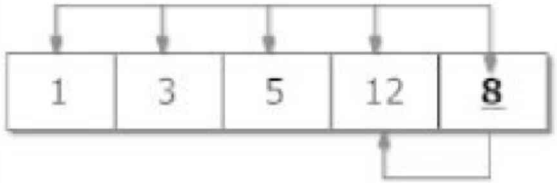
Step 1		Checking second element of array with element before it and inserting it in proper position. In this case, 3 is inserted in position of 12.
Step 2		Checking third element of array with elements before it and inserting it in proper position. In this case, 1 is inserted in position of 3.
Step 3		Checking fourth element of array with elements before it and inserting it in proper position. In this case, 5 is inserted in position of 12.
Step 4		Checking fifth element of array with elements before it and inserting it in proper position. In this case, 8 is inserted in position of 12.
		Sorted Array in Ascending Order

Figure: Sorting Array in Ascending Order Using Insertion Sort Algorithm

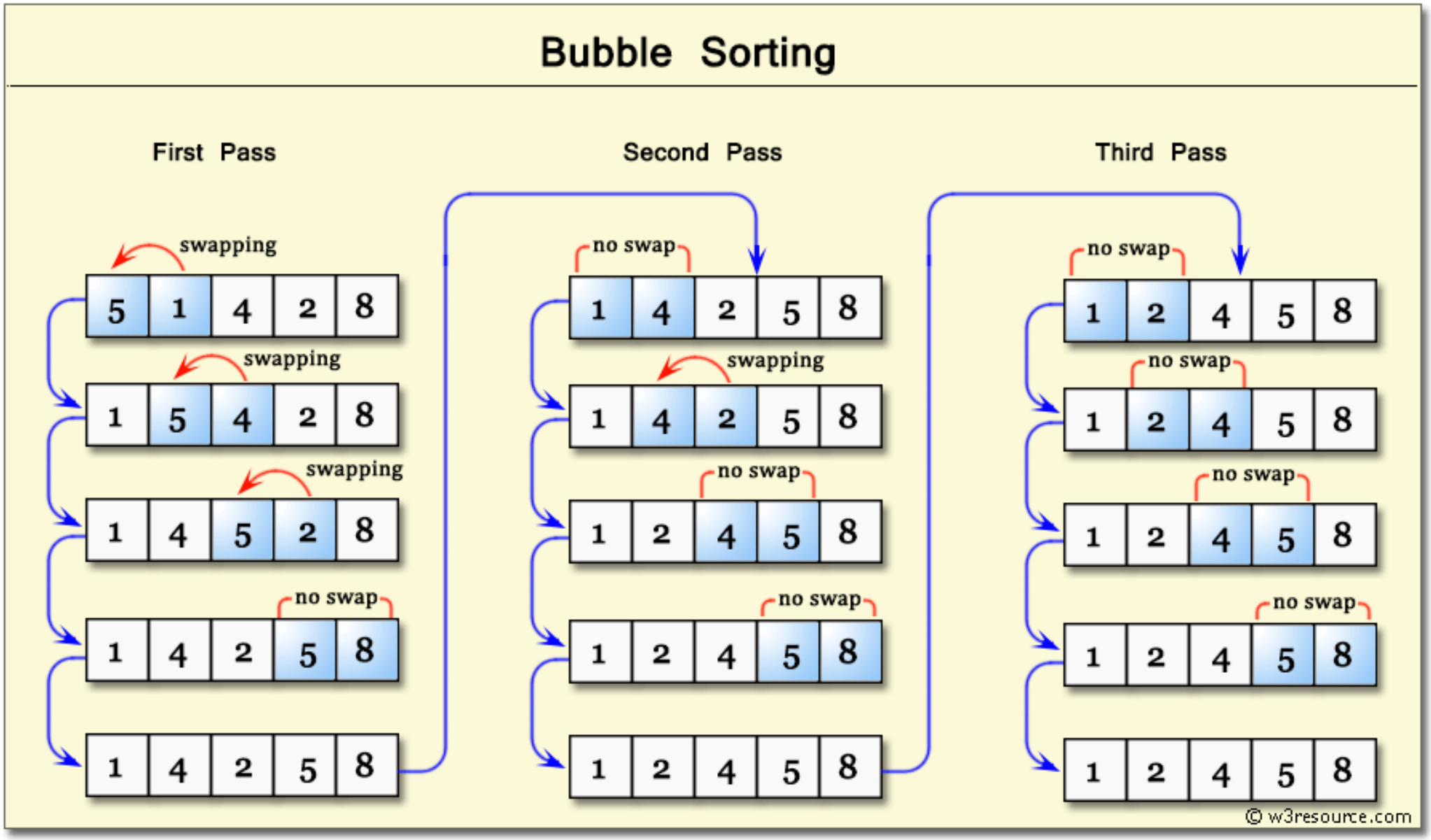
Murdoch UNIVERSITY

Insertion Sort

- **EXERCISE IN TOPIC 7**
  - Write down the array contents at every step during the sorting of the values, 12, 18, 2, -4, 17, 12 using insertion sort
  - Note down how many comparisons were made between values

Bubble Sort

- Also known as *sinking sort* because the smaller values in the array gradually bubble their way towards the top of the array (towards index 0), if sorting in ascending order
- This sorting involves several passes through the array
- On each pass, successive pairs are compared
- If the pairs are in decreasing order they are swapped



Murdoch UNIVERSITY

Bubble Sort

```
// bubblesort.java
// To sort an array using Bubble Sort
public class bubblesort {
   public static void main(String[] args) {
      int[] anArray ={10,9,8,7,6,5,4,3,2,1.0.-1};
      // sort the array in ascending order:
      SortArrayByBubbleSort(anArray);
      // output the sorted numbers:
      System.out.println("The sorted numbers are:");
      for (int i = 0;i < anArray.length;i++)
         System.out.println(anArray[i]);
      System.out.println("End program: Bye.");
   } // end main
```

Bubble Sort

```
public static void SortArrayByBubbleSort(
                                    int[] arrayToSort) {
   // number of passes
   for(int i = 1;i <arrayToSort.length;i++) {
      // perform one pass
      for(int j=0;j<arrayToSort.length-1;j++)
         // perform one comparison
         if (arrayToSort[j]>arrayToSort[j+1])
            swap (arrayToSort, j, j+1);
   } // end i for
} // end method SortArrayByBubbleSort
```

Bubble Sort

```
public static void swap (int[]a, int first,
                                        int second)
{
      int temp;
      temp = a[first];
      a[first] = a[second];
      a[second] = temp;
   } // end method swap
}//end of class bubblesort
```

Bubble Sort

```
/* Output
The sorted numbers are:
-1
0
1
2
3
4
5
6
7
8
9
10
End of program - Bye.
*/
```

Another Searching Algorithm: Binary Search

- Binary Search:
  - Can only be performed on sorted arrays
  - Much faster than linear searching but more complex
  - The search item is compared against the value of middle element of the array
  - If search item < middle element of array, search is restricted to first half of the array, and so on …
  - If search item > middle element of array, search second half of the array, and so on …
  - If search item = middle element, search is complete
  - Thus, each subsequent pass divides the array by half

Another Searching Algorithm: Binary Search

```
/** Binary Search searches a sorted array
Assumes the array is sorted in ascending order. If
search is successful, an index of the array where
target key is found will be returned, otherwise the
value -1 will be returned */
public int binarySearch(int arr[], int key) {
   int first = 0;
   int last = ar.length - 1;
   int mid;
   while (first <= last) {
      mid = (first + last) / 2;
```

Another Searching Algorithm: Binary Search

```
        if (key == arr[mid])     // match found
            return mid;                 // exit
        else if(key < arr[mid]) // search low end
            last = mid - 1;
        else                    //search high end of array
            first = mid + 1;
    }// end while
    return -1;    // match not found
}// end binarySearch
```

Multi-Dimensional Arrays

- These are arrays with more than one index
  - The number of dimensions = number of indexes
- Arrays with more than two dimensions are a simple extension of two-dimensional (2-D) arrays
- Java allows multi-dimensional arrays to be used.
  - An array is n-dimensional if it uses n indexes
  - Up to now we have been studying one-dimensional arrays

Multi-Dimensional Arrays

- A 2-D array corresponds to a table or grid
  - One dimension is the row
  - The other dimension is the column
  - Cell = an intersection of a row and column
  - An array element corresponds to a cell in the table
- The syntax for 2-D arrays is similar to 1-D arrays:

```
Base_Type[][] arrayName = new
                Base_Type[intExp1][intExp2];
```

Multi-Dimensional Arrays

- Eg: declare a 2-D array of `ints` named `table`, the table is to have ten rows and six columns:

```
int[][]table = new int[10][6];
```

- To access a component of a 2-dimensional array:

```
arrayName[indexExp1][indexExp2];
// where:
intExp1, intExp2 >= 0
indexExp1 = row position
indexExp2 = column position
```

Multi-Dimensional Arrays

- Eg:

```
table[1][2] = 0;
```

// initialises the cell (element) in the second row and
// third column of table to zero

- **Usage:** Often we want to store values according to a more complex indexing system. Eg:
  - The time of the `ith` competitor in the `jth` race
  - The rainfall on the `dth` day of the `mth` month of the `yth` year
  - The value in the `rth` row and `cth` column of the `tth` table in the `bth` book of tables

Multi-Dimensional Arrays

- The examples above are 2, 3 and 4 dimensional respectively
- Eg:

```
int[][] matrix = new int[20][10];
for (int row=0;row<20;row++)
  for(int column=0;column <10;column++)
     matrix[row][column] = row*column;
```

Multi-Dimensional Arrays

- Eg:

```
int[][][] rain = new int[31][12][100];
// initialise all elements of array rain to zero
for (int i=0;i < 31;i++)
  for (int j=0;j < 12;j++)
     for (int k=0;k < 100;k++)
        rain[i][j][k] = 0;
// display the value of one element of array rain
System.out.println("Rainfall on the second of
   January 26 is " + rain[1][0][62] + " cm");
```

Multi-Dimensional Arrays

- The indexed variables (array elements) for multi-dimensional arrays are just like indexed variables for 1-d arrays, except that they have multiple indexes

Example

```
/** Displays a 2-D table showing how interest rates
affect bank balances. From Savitch (7th ed) pp.579-583
   */
public class InterestTable2 {
  public static final int ROWS = 10;
  public static final int COLUMNS = 6;
  public static void main(String[] args) {
     int[][] table = new int[ROWS][COLUMNS];
     int row, col;
     for (row = 0; row < ROWS; row++)
        for (col = 0; col < COLUMNS; col++)
```

Example

```
            table[row][col]=
                                getBalance(1000.00,row+1,
                                           (5 + 0.5 * col));
    System.out.println("Balances for Various
                                             Interest Rates");
    System.out.println("Compounded Annually");
    System.out.println("(Rounded to Whole Dollar

 Amounts)");
    System.out.println("Years 5.00% 5.50% 6.00%
                                          6.50% 7.00% 7.50%");
    System.out.println();
    showTable(table);
}// end main
```

Example

```
/** Pre-condition: Array displayArray has ROWS rows and
    COLUMNS columns
Post-condition: Array contents displayed with $ signs
    */
public static void showTable(int[][] anArray) {
   int row, column;
   for (row = 0;row < ROWS;row++) {
      System.out.print((row + 1) + "       ");
      for (col = 0;col < COLUMNS;col++)
         System.out.print("$" +
                          anArray[row][column] + "  ");
      System.out.println();
   }
}// end showTable
```

Example

```
/** Returns the balance in an account after a given
    number of years and interest rate with an initial
    balance of startBalance. Interest is compounded
    annually. The balance is rounded to a whole
    number.*/
public static int getBalance(double
    startBalance, int years, double rate) {
   double runningBalance = startBalance;
   int count;
   for (count = 1;count <= years;count++)
      runningBalance=runningBalance*(1+rate/100);
   return (int)(Math.round(runningBalance));
}// end getBalance
}// end class InterestTable2
```

Example

OUTPUT (from InterestTable2):

Balances for Various Interest Rates
Compounded Annually
(Rounded to Whole Dollar Amounts)

Years	5.00%	5.50%	6.00%	6.50%	7.00%	7.50%
1	$1050	$1055	$1060	$1065	$1070	$1075
2	$1103	$1113	$1124	$1134	$1145	$1156
3	$1158	$1174	$1191	$1208	$1225	$1242
4	$1216	$1239	$1262	$1286	$1311	$1335
5	$1276	$1307	$1338	$1370	$1403	$1436
6	$1340	$1379	$1419	$1459	$1501	$1543
7	$1407	$1455	$1504	$1554	$1606	$1659
8	$1477	$1535	$1594	$1655	$1718	$1783
9	$1551	$1619	$1689	$1763	$1838	$1917
10	$1629	$1708	$1791	$1877	$1967	$2061

Implementation of Multi-Dimensional Arrays

- In Java multi-dimensional arrays are implemented using 1-d arrays
  - That is, multidimensional arrays are arrays of arrays
- With this knowledge, we can use the `length` instance variable to process multi-dimensional arrays
- For example, the following code from the `main` method in class `InterestTable2`

Implementation of Multi-Dimensional Arrays

```
for (row = 0;row < ROWS;row++)
  for (col = 0;col < COLUMNS;col++)
     table[row][col] =
              computeBalance(1000.00, row+1,
                                  (5 + 0.5*column));
```

// can be written as

```
for (row = 0;row < table.length;row++)
  for(col = 0;col < table[row].length;col++)
     table[row][col] =
              computeBalance(1000.00, row+1,
                                  (5 + 0.5*column));
```

Murdoch UNIVERSITY

Implementation of Multi-Dimensional Arrays

- This means that `table` is actually a 1-d array of length 10, and each of the 10 indexed variables `table[0]` to `table[9]` is a 1-d array of length 6
- Thus the array `table` is an array of arrays

Implementation of Multi-Dimensional Arrays

- Its declaration

```
int[][] table = new int[10][6];
```

is equivalent to the following:

```
int[][] table;
table = new int[10][];
table[0] = new int[6];
table[1] = new int[6];
. . . . . .
table[9] = new int[6];
```

Implementation of Multi-Dimensional Arrays

- Since a 2-d array in Java is an array of arrays, there is no need for each row to have the same number of elements
- That is, rows can have different number of columns
- Such arrays are called **ragged arrays**

Implementation of Multi-Dimensional Arrays

- Eg:

```
int[][] raggedTable;
raggedTable = new int[3][];
raggedTable[0] = new int[2];
raggedTable[1] = new int[5];
raggedTable[2] = new int[7];
```

Another Example of 2-D Arrays


```
// TwoDimArrayV2.java modified from Deitel and Deitel
// First dimension (rows) represents number of students
// Second dimension (columns) represents number of
   scores
// per student
public class TwoDimArrayV2 {
  public static void main(String[] args) {
     int scores[][] = {{57,74,55,67},
                       {35,60,62,54},
                       {73,82,95,87}};
     // 3 students, 4 scores per student
     int students, minScore, maxScore;
```

Another Example of 2-D Arrays


```
    displayArray (scores); // output the array
    // find the minimum and the maximum scores
    minScore = findMinimum(scores);
    maxScore = findMaximum(scores);
    System.out.println("Lowest score: " + minScore
        + "\nHighest score: " + maxScore + "\n" );
    // find and display the average score
    students = scores.length; // no of students
    for (int i = 0;i < students;i++)
       System.out.println("Average for student "+i
              + " is " + findAverage(scores[i]) );
    System.out.println ("\nEnd of program - bye.");
} // end main
```

Another Example of 2-D Arrays


```
// find the minimum score
public static int findMinimum(int[][]
                                studentArray)
{
    int min = 100;
    int students, exams;
    students = studentArray.length;
    exams = studentArray[0].length;
    for(int i = 0;i<students;i++) // each student
       for (int j = 0;j < exams;j++) // each grade
          if (studentArray[i][j] < min)
             min = studentArray[i][j];
    return min;
}// end findMiniumum
```

Another Example of 2-D Arrays


```
// find the maximum grade
public static int findMaximum(int[][]
                                studentArray)
{
    int max = 0;
    for(int i = 0;i < studentArray.length;i++)
       // for each student
       for(int j = 0;j<studentArray[i].length;j++)
          // for each grade
          if (studentArray[i][j] > max)
             max = studentArray[i][j];
     return max;
  } // end findMaximum
```

Another Example of 2-D Arrays


```
// avg score for particular student (or set of scores)
public static double findAverage(int
                                 setOfScores[])
{
    int total = 0;
    double average;
    for (int i = 0;i < setOfScores.length;i++)
       total = total + setOfScores[ i ];
    average = (double)total/setOfScores.length;
    return average;
} // end findAverage
```

Another Example of 2-D Arrays


```
// builds up array in string variable and displays it
public static void displayArray(int[][]
                                        studentArray) {
   // used to align column heads
   String output = "              ";
   for (int i=0;i<studentArray[0].length;i++)
      // no of columns
      output = output + "[" + i + "]   ";
```

Another Example of 2-D Arrays


```
    for (int i=0;i<studentArray.length; i++) {
       // for each row
       output = output+"\nscores[" + i + "]  ";
      for(int j=0;j<studentArray[0].length;j++)
          output=output+studentArray[i][j]+"  ";
    } // end for
    System.out.println("\nThe array is:\n");
    System.out.println(output);
  } // end displayArray
} // end class TwoDimArray
```

Another Example of 2-D Arrays


```
/* OUTPUT
The array is:
             [0]   [1]   [2]   [3]
scores[0]     57    74    55    67
scores[1]     35    60    62    54
scores[2]     73    82    95    87
Lowest score: 35
Highest score: 95
Average for student 0 is 63.25
Average for student 1 is 52.75
Average for student 2 is 84.25
End of program - bye.
*/
```

Murdoch UNIVERSITY

End of Topic 6

Murdoch UNIVERSITY