COMP 202

Programming With Arrays

CONTENTS:
- Arrays, 2D Arrays, Multidimensional Arrays
- The Array List
- Variable Length parameter lists
- The For-each Statement

Thinking Like A Programmer:
Designing for arrays

Part 1

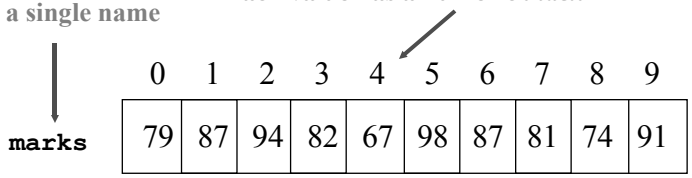
Array Basics

Arrays

- An *array* is an ordered list of values

The entire array has a single name

Each value has a numeric *index*



An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays

- A particular value in an array is referenced using the array name followed by the index in brackets
- For example, the expression

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

  refers to the value 94 (which is the 3rd value in the array)
- That expression represents a place to store a single integer, and can be used wherever an integer variable can
- For example, it can be assigned a value, printed, or used in a calculation

Arrays

- An array stores multiple values of the same type
  - That type can be primitive types or objects
  - Therefore, we can create an array of integers, or an array of characters, or an array of String objects, etc.
- In Java, the array itself is an object
  - Therefore the name of the array is an object reference variable, and the array itself is instantiated separately

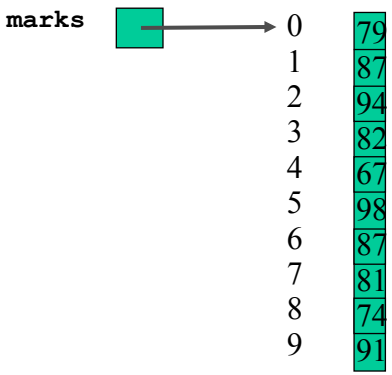
Declaring Arrays

- The `marks` array could be declared as follows:

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

- Note:
  - that the type of the array does not specify its size, but each object of that type has a specific size
  - The type of the variable `marks` is `int[]` (an array of integers)
  - It is set to a new array object that can hold 10 integers

Array Object with elements of primitive type (e.g., integer)



Why Arrays?

- Imagine you want to write a program that asks the user to enter in 50 student midterm marks. The program would then display each grade with the student's ID number and then the average for that assignment in a nice report.
- How would you do it?
  - One way would be to provide 100 variables:
    - `int mark1, mark2, mark3, (until 50)`
    - `int ID1, ID2, ID3, (until 50)`
  - The array provides for a way to create only two variables:
    - `int marks[]  = new int[50];`
    - `int IdNumbers[] = new int[50];`
  - Isn't this nicer?
  - Imagine trying to calculate the average with 50 variables…

MarksArray.java

```
import java.util.Scanner;
public class MarksArray {
   final static int LIMIT = 50;

   public static void main (String[] args){
      int[] marks = new int[LIMIT]; // array with marks
      double average = 0.0;  // average grade
      int sum, lowmark, position; // helpers
      int index; // for loop
      Scanner scan = new Scanner(System.in);

      //  Initialize the array values
      for (index = 0; index < LIMIT; index++){
         System.out.println("mark " + (index+1) + ": ");
         marks[index] = scan.nextInt();
      }
```


MarksArray.java

```
   //  calculate average
   for (index = 0, sum=0; index < LIMIT; index++)
       sum += marks[index];
   average = (double) sum / LIMIT;
   System.out.println("The average mark is " + average);

   // gets students with lowest grade;
   for (index = 0, position = 0; index < LIMIT; index++)
   {
       if (marks[index] < marks[position])
           position = index;
   }
   System.out.println("Lowest mark " + marks[position] +
                      " by student " + (position+1));
}
```


Declaring Arrays

- Some examples of array declarations:

```
float[] prices = new float[500];

boolean[] flags;
flags = new boolean[20];

char[] productID = new char[1750];
```


Declaring Array Variations

- `int[] x, y, z;`
- `int x[], y, z;`

In the first example: `int[]` identifies that x, y and z are all arrays.

In the second example `x[]` identifies that only x is an array, y and z are simply integers.

Bounds Checking

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in bounds (0 to N-1)
- The Java interpreter will throw an exception if an array index is out of bounds
- This is called automatic *bounds checking*

Bounds Checking

- For example, if the array `codes` can hold 100 values, it can only be indexed using the numbers 0 to 99
- If `count` has the value 100, then the following reference will cause an `ArrayOutOfBoundsException`:

```
System.out.println (codes[count]);
```

- It's common to introduce *off-by-one errors* when using arrays

problem

```
for (int index=0; index <= 100; index++)
   codes[index] = index*50 + epsilon;
```


Bounds Checking

- Each array object has a public constant called `length` that stores the size of the array
- It is referenced using the array name (just like any other object):

```
if (scores.length < 10)
```

- Note that `length` holds the number of elements, not the largest index

ReverseNumbers.java

```
import java.util.Scanner;
public class ReverseNumbers {
   public static void main (String[] args) {
      Scanner scan     = new Scanner (System.in);
      double[] numbers = new double[10];

      System.out.println ("The size of the array: " + numbers.length);

      for (int index = 0; index < numbers.length; index++)
      {
         System.out.print ("Enter number " + (index+1) + ": ");
         numbers[index] = scan.nextDouble();
      }

      System.out.println ("The numbers in reverse:");

      for (int index = numbers.length-1; index >= 0; index--)
         System.out.print (numbers[index] + "  ");

      System.out.println ();
   }
}
```

How does this work?

TravelLog.java

```
public class TravelLog {
  public static void main (String[] args) {
      Scanner scan    = new Scanner (System.in);
      int[] travelLog = new int[8];
      int sum = 0, entry; double average;

      System.out.println ("Let's fill the travel log for the first "
                          + travelLog.length + " hours.");

      for (int i=0; i<travelLog.length; i++) {
          System.out.print ("Enter the number of kilometers covered "
                            + " in hour " + (i+1) + " : ");
          travelLog[i] = scan.nextInt();
      }
      for (int i=0; i<travelLog.length; i++) sum += travelLog[i];

      System.out.println ("The total distance covered is " + sum + " km.");
      average = (double)sum / travelLog.length;

      DecimalFormat fmt = new DecimalFormat ("0.###");
      System.out.println ("Average speed " + fmt.format(average) + " km/h.");
      System.out.print ("Review Trip (-1 to exit)? ");

      entry = scan.nextInt();
      while (entry>0 && entry<=travelLog.length) {
         System.out.println ("Kilometres covered " + travelLog[entry-1]);
         System.out.print("Review Trip (-1 to exit)? ");
         entry = scan.nextInt();
      }
   } }
```

Run through by hand/eye…

LetterCount.java

```
import java.util.Scanner;
public class LetterCount {
   public static void main (String[] args) {
      final int NUMCHARS = 26;
      Scanner scan = new Scanner (System.in);
      int[] upper  = new int[NUMCHARS];
      int[] lower  = new int[NUMCHARS];
      char current; int other = 0;

      System.out.println ("Enter a sentence:");
      String line = scan.nextLine();
      for (int ch = 0; ch < line.length(); ch++) {
         current = line.charAt(ch);
         if (current >= 'A' && current <= 'Z')
            upper[current-'A']++;
         else if (current >= 'a' && current <= 'z')
            lower[current-'a']++;
         else other++;
      }
      for (int letter=0; letter < upper.length; letter++) {
         System.out.print ( (char) (letter + 'A') );
         System.out.print (": " + upper[letter]);
         System.out.print ("\t\t" + (char) (letter + 'a') );
         System.out.println (": " + lower[letter]);
      }
      System.out.println ("Non-alphabetic characters: " + other);
   }}
```


Initializer Lists

- An *initializer list* can be used to instantiate and initialize an array in one step
- The values are delimited by braces and separated by commas
- Examples:

```
int[] units = {147, 323, 89, 933, 540,
               269, 97, 114, 298, 476};

char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```


Initializer Lists

- Note that when an initializer list is used:
  - the `new` operator is not used
  - no size value is specified
- The size of the array is determined by the number of items in the initializer list
- An initializer list can only be used in the declaration of an array

Primes.java

```
public class Primes
{
   public static void main (String[] args)
   {
      int[] primes = {2, 3, 5, 7, 11, 13, 17, 19};

      System.out.println ("Array length: " + primes.length);

      System.out.println ("The first few prime numbers are:");

      for (int prime = 0; prime < primes.length; prime++)
         System.out.print (primes[prime] + "  ");

      System.out.println ();
   }
}
```


Part 2

Arrays and Parameters

Arrays as Parameters

- An entire array can be passed to a method as a parameter
  `convert(int[] aList) {…}` → `convert(units);`
- Note:
  - Like any other object, the reference to the array is passed, making the formal and actual parameters aliases of each other
  - Changing an array element in the method changes the original
- An array element can be passed to a method as well, and will follow the parameter passing rules of that element's type
  `convertOne(int i) {…}` → `convertOne (units[3]);`
- Arrays can also be returned: `int[] oddList(int limit)`

Problem

- Write a program that asks the user for positive integer numbers and the amount of numbers they want to enter.
  - Create an array to be the exact size as the number of values they want to enter
  - If the user does not input a positive value the program asks them to input again
  - The program will then call a function called average that will accept an array as parameter and return a double value representing the average of all the values in the array.

Part 3

Arrays and Objects

Arrays of Objects

- The elements of an array can be object references
- The following declaration reserves space to store 10 references to `String` objects

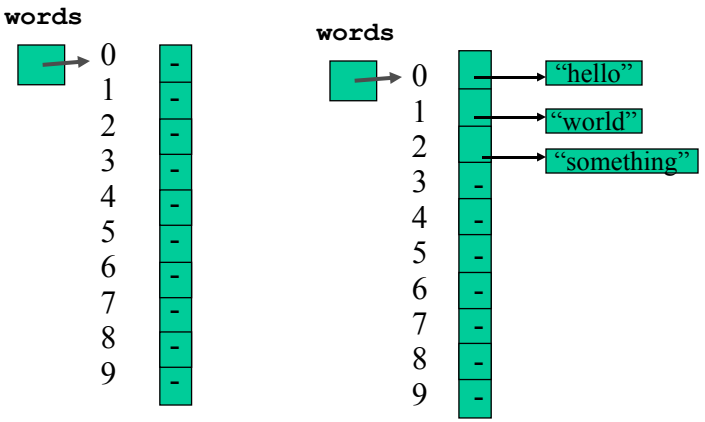
```
String[] words = new String[10];
```

- It does NOT create the `String` objects themselves
- Each object stored in an array must be instantiated separately

Array of String Objects

At declaration:

At a few string object created and put into the array



GradeRange.java

```
public class GradeRange
{
   public static void main (String[] args)
   {
      String[] grades = {"A", "A-", "B+", "B", "B-", "C+", "C",
                         "D", "F"};

      float[] gradePoint = { 4.0f, 3.7f, 3.3f, 3.0f, 2.7f, 2.3f,
                             2.0f, 1.0f, 0.0f};

      int[] cutoff = {85, 80, 75, 70, 65, 60, 55, 50, 0};

      System.out.println ("GRADE" + "\t" + "GP" + "\t" + "CUTOFF");

      for (int level = 0; level < cutoff.length; level++)
         System.out.println (grades[level] + "\t"
                             + gradePoint[level] + "\t"
                             + cutoff[level]);

   }
}
```

What does this do?

FeedTheLitter.Java

```
public class FeedTheLitter {
   public static void main (String[] args)
   {
        final int NUM_CATS = 3;
        Cat[] litter = new Cat[NUM_CATS];
        for (int i=0; i<NUM_CATS; i++)
        {
           litter[i]=new Cat();
           System.out.println("\nCat " + (i+1) + " :\n" + litter[i]);
        }

        System.out.println("\n******** Let's feed our cats ********\n");

        for (int i=0; i<NUM_CATS; i++)
        {
           System.out.println();
           litter[i].feed(i*2.0f);
           System.out.println("\nCat " + (i+1) + " :\n" + litter[i]);
        }
   }
}
```


Command Line Arguments

- public static void main(String [] args)
  - an array of Strings, provided by the command line
  - `java MyProgram`
  - `java MyProgram a 123 foo`

```
public class MyProgram {
   public static void main(String args[]) {
      System.out.println("Num arguments: "+args.length);
      for (int i=0;i<args.length;i++)
         System.out.println("Arg["+i+"]: "+args[i]);
   }
}
```


Objects with Arrays

- Objects can have arrays as instance variables
- Therefore, fairly complex structures can be created simply with arrays and objects
- The software designer must carefully determine an organization of data and objects that makes sense for the situation

Problem

Create a class called MyCD storing information about an individual CD. Then create another class called MyInventory that has a main method that creates an array of MyCD objects. The program will then display a menu: Add CD, List CDs, Quit.

Add CD adds a CD to the next available cell.
List CD lists all the filled cells
Quit ends the program

Possible Other Problem

- Define a student with a name, an id, marks for final, midterm and 5 assignments.
  - define proper setter and getter methods
  - define toString method
- Define a class that supports a group of students
  - allows to enter students to group
  - allows to calculate average for midterm, final, etc.
- Define a starter class that creates the student group for 202.

Part 4

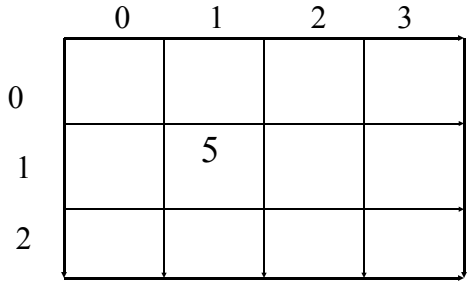
Multidimensional Arrays

Two-Dimensional Arrays

- A *one-dimensional array* stores a simple list of values
- A *two-dimensional array* can be thought of as a table of values, with rows and columns
- A two-dimensional array element is referenced using two index values
- To be precise, a two-dimensional array in Java is an array of arrays

For Example…

```
int x[][] = new int[3][4];
x[1][1] = 5;
```



TwoDArray.java

```
public class TwoDArray
{
   public static void main (String[] args)
   {
      int[][] table = new int[5][10];

      // Load the table with values
      for (int row=0; row < table.length; row++)
         for (int col=0; col < table[row].length; col++)
            table[row][col] = row * 10 + col;

      // Print the table
      for (int row=0; row < table.length; row++)
      {
         for (int col=0; col < table[row].length; col++)
            System.out.print (table[row][col] + "\t");
         System.out.println();
      }
   }
}
```

What does this do and how?

Multidimensional Arrays

- An array can have as many dimensions as needed, creating a multidimensional array
- Each dimension subdivides the previous one into the specified number of elements
- Each array dimension has its own `length` constant
- Because each dimension is an array of array references, the arrays within one dimension could be of different lengths

For Example…

```
double x[][][] = new double[5][5][5];
x[1][2][4] = 5.3;

int a[][] = new int[3][];
for(int i=0;i<3;i++) {
   a[i] = new int[i+1];
   a[i][i] = 5.3;
}
```

Actual use of non-rectangular arrays is rare...

Part 5

The ArrayList Object

The `ArrayList` Class

- Often programmers want the functionality of an array but the freedom to not have to manage an array.
- Often programmers want to insert and delete items from an array. In other words, the array size can grow or shrink.
- Arrays can do this but an `ArrayList` is a built-in object that is available for you to use.

The `ArrayList` Class

- An object of class `ArrayList` is similar to an array in that it stores multiple values
- However, an ArrayList
  - only stores objects
  - does not have the indexing syntax that arrays have
- The methods of the `ArrayList` class are used to interact with the elements of a vector
- The `ArrayList` class is part of the `java.util` package

The `ArrayList` Class

- `ArrayList()`
- `boolean add(Object obj)`
- `void add(int index, Object obj)`
- `Object remove(int index)`
- `Object set(int index, Object obj)`
- `void clear()`
- `boolean contains(Object obj)`
- `int indexOf(Object obj)`
- `Object get(int index)`
- `boolean isEmpty()`
- `int size()`

Beatles.java

```
import java.util.ArrayList;
public class Beatles {
   public static void main (String[] args) {
      ArrayList band = new ArrayList();

      band.add ("Paul");
      band.add ("Pete");
      band.add ("John");
      band.add ("George");

      System.out.println (band);

      int location = band.indexOf ("Pete");
      band.remove (location);

      System.out.println (band);
      System.out.println ("Size of the band: " + band.size());

      band.add (location,"Ringo");

      System.out.println (band);
      System.out.println ("Size of the band: " + band.size());
   }
}
```

What does this do?

The `ArrayList` Class

- An important difference between an array and an ArrayList is that a ArrayList can be thought of as dynamic, able to change its size as needed
- Each ArrayList initially has a certain amount of memory space reserved for storing elements
- If an element is added that doesn't fit in the existing space, more room is automatically acquired

The ArrayList Class

- The `ArrayList` class is implemented using an array
- Whenever new space is required, a new, larger array is created, and the values are copied from the original to the new array
- To insert an element, existing elements are first copied, one by one, to another position in the array
- The implementation of `ArrayList` in the API is not very efficient for inserting elements

Part 6

The For-each Statement

The Foreach statement

```
public int sum(int[] aList)
{
   int total = 0;
   for (int i=0; i<aList.length; i++) {
      int num = aList[i]
      total += num;
  }
   return total;
}
```

is equivalent to

```
public int sum(int[] aList)
{
   int total = 0;
   for (int num : aList)
       total += num;
   return total;
}
```

For-each Statement

- The *foreach* statement works on arrays and the `ArrayList` class.
  - (and on any object whose class implements the `Iterable` interface -- but we haven't covered interfaces in class)
- In the previous `Beatles` example, we could have printed out the members of the band using:

```
for (Object temp : band)
    System.out.println(temp);
```


Part 7

Variable length parameter list

Variable length parameter list

- A method can also have a *variable length parameter list* which automatically gets converted to an array inside the method:

```
public int sum(int ... aList)
{
    int total = 0;
    for (int num : aList)
        total += num;
    return total;
}
```

- This method could be called, e.g.,
  - sum(1, 4, 59) returning 64.
  - sum(2, 3) returning 5.
- A method can have only one variable length parameter and it must be after all other parameters in the parameter list
- An overloaded method with an exact number of parameters always has precedence over the variable length list method

Part 8

Thinking Like A Programmer

Designing for Arrays

- Arrays are a mass storage structure
- Design your program with these rules:
  - Identify large units of data
    - If the data is simple then use a regular array
    - If the data is complex use an array of objects
    - If the data is massive then use a file or database (arrays use too much memory when data is massive)
  - Construct your object to be used for the array
    - Use encapsulation to fully implement all the features of the data you want to store.

McGill
COMP 202 – Introduction to Computing 1

Try these problems…

Problem 1: Sum 50 integer numbers and print them out.

Problem 2: Implement a bank and bank account program.

Problem 3: Matrix Multiplication

Discuss for each the design issues from the previous slide.
Do this for each problem.
Then pick one to program.