

Java Class 20

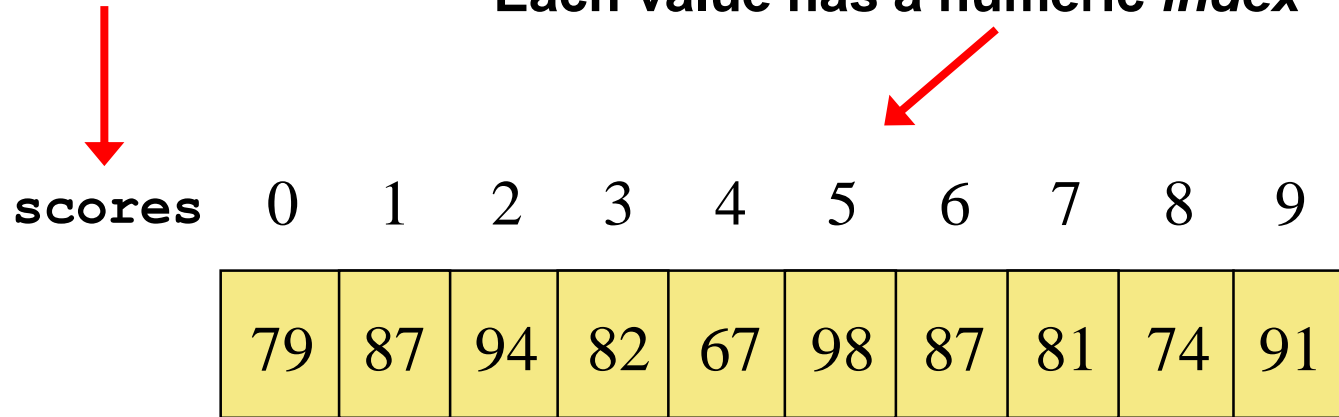


Arrays

An array is an **ordered** list of values:

**The entire array
has a single name**

Each value has a numeric *index*



scores	0	1	2	3	4	5	6	7	8	9
	79	87	94	82	67	98	87	81	74	91

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 **square brackets**

For example, the expression

```
scores[2]
```

refers to the value 94 (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 be used

Arrays

For example, an array element can be assigned a value, printed, or used in a calculation:

```
scores[2] = 89;  
scores[first] = scores[first] + 2;  
mean = (scores[0] + scores[1])/2;  
System.out.println ("Top = " + scores[5]);  
pick = scores[rand.nextInt(10)];
```

Declaring Arrays

The `scores` array could be declared as follows:

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

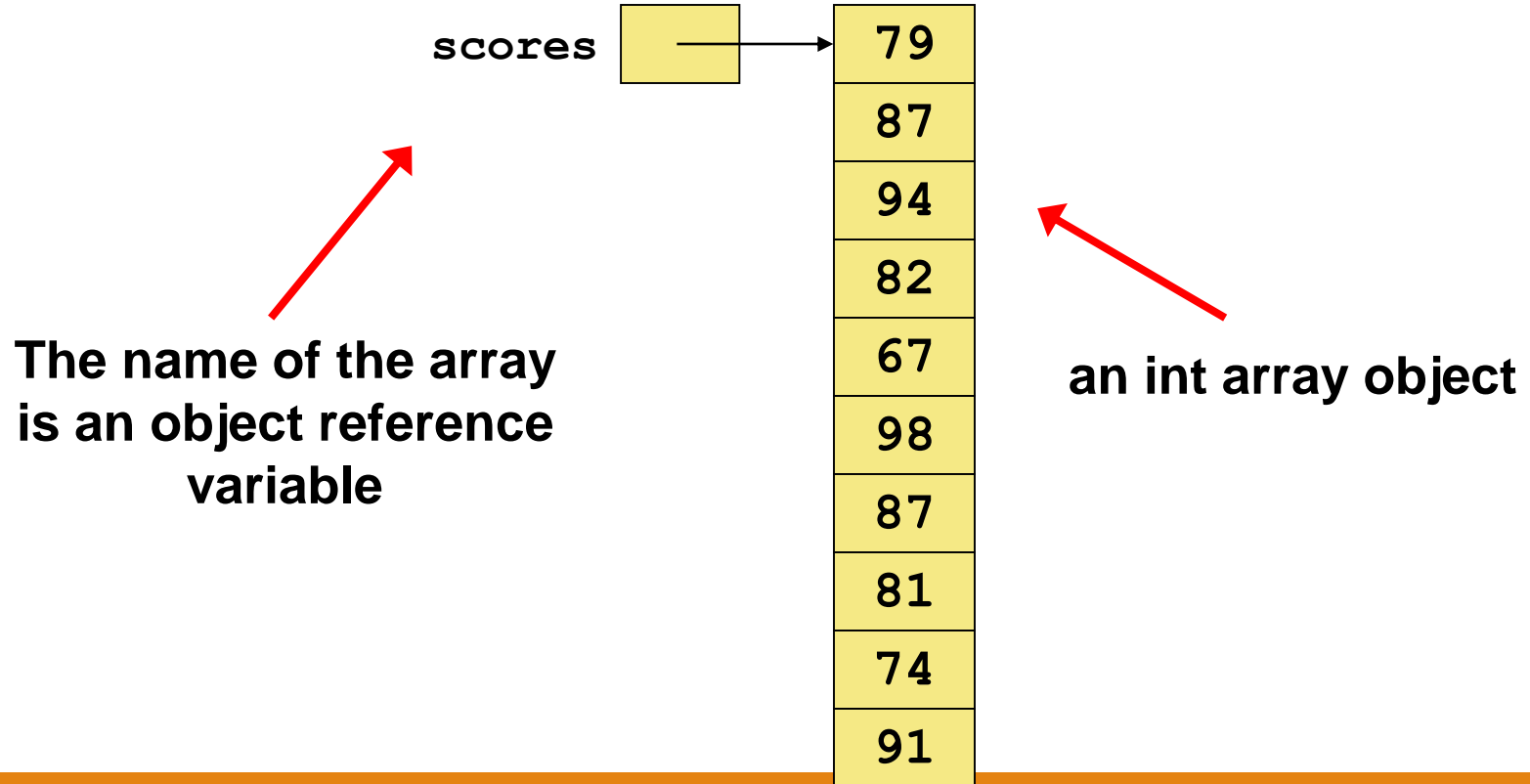
- The type of the variable `scores` is `int[]` (a reference to an array of integers)
- The **reference variable** `scores` is declared to refer to a **new array object** that can hold 10 integers (**with initial values 0s**)
- Once an array is declared to be a certain size, its size **cannot** be changed

```
scores[0] = 79;
```

```
scores[1] = 87;
```

Declaring Arrays

Another way to depict the `scores` array:



Declaring Arrays

The values held in an array are called *array elements*

An array stores multiple values of **the same type** – the *element type*

The element type can be a primitive type or an object reference

Therefore, we can create an array of integers, an array of characters, an array of `String` **references**, an array of `Dog` **references** (next lecture), etc.

Declaring Arrays

Some other examples of array declarations:

```
int[] weights = new int[2000];
```

```
int N = 500;
```

```
double[] prices = new double[N];
```

```
boolean[] flags;
```

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

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


Using Arrays

The **for-each version** of the `for` loop can be used when processing array elements:

```
for (int score : scores)
    System.out.println(score);
```

It is equivalent to

```
for (int index = 0; index < 10; index++)
    System.out.println(scores[index]);
```

Quick Check

Write an array declaration to represent the ages of 100 children.

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

Write code that prints each value in array `ages`.

```
for (int age : ages)  
    System.out.println(age);
```

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 range 0 to N-1

For example, if the array `codes` can hold 100 values, it can be indexed from 0 to 99

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:

```
scores.length
```

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

```
//*****  
//  ReverseOrder.java          Author: Lewis/Loftus  
//  
//  Demonstrates array index processing.  
//*****  
  
import java.util.Scanner;  
  
public class ReverseOrder  
{  
    //-----  
    //  Reads a list of numbers from the user, storing them in an  
    //  array, then prints them in the opposite order.  
    //-----  
    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);  

```

continue

continue

```
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 order:");

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

Sample Run

The size of the array: 10

Enter number 1: 18.36

Enter number 2: 48.9

Enter number 3: 53.5

Enter number 4: 29.06

Enter number 5: 72.404

Enter number 6: 34.8

Enter number 7: 63.41

Enter number 8: 45.55

Enter number 9: 69.0

Enter number 10: 99.18

The numbers in reverse order:

99.18 69.0 45.55 63.41 34.8 72.404 29.06 53.5 48.9 18.36

```
//*****  
// LetterCount.java          Author: Lewis/Loftus  
//  
// Demonstrates the relationship between arrays and strings.  
//*****
```

```
import java.util.Scanner;
```

```
public class LetterCount  
{
```

```
    //-----  
    // Reads a sentence from the user and counts the number of  
    // uppercase and lowercase letters contained in it.  
    //-----
```

```
    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;    // the current character being processed
```

```
        int other = 0;    // counter for non-alphabetics
```

continue

continue

```
System.out.println ("Enter a sentence:");
String line = scan.nextLine();

// Count the number of each letter occurrence
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++;
}
```

continue

Characters Unicode Values

'0' - '9'	48 through 57
'A' - 'Z'	65 through 90
'a' - 'z'	97 through 122

continue

```
// Print the results
System.out.println ();
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 ();
System.out.println ("Non-alphabetic characters: " + other);
}
}
```

Sample Run

Enter a sentence:

In Casablanca, Humphrey Bogart never says "Play it again, Sam."

A: 0	a: 10
B: 1	b: 1
C: 1	c: 1
D: 0	d: 0
E: 0	e: 3
F: 0	f: 0
G: 0	g: 2
H: 1	h: 1
I: 1	i: 2
J: 0	j: 0
K: 0	k: 0
L: 0	l: 2
M: 0	m: 2
N: 0	n: 4
O: 0	o: 1
P: 1	p: 1
Q: 0	q: 0

continue

Sample Run (continued)

R: 0	r: 3
S: 1	s: 3
T: 0	t: 2
U: 0	u: 1
V: 0	v: 1
W: 0	w: 0
X: 0	x: 0
Y: 0	y: 3
Z: 0	z: 0

Non-alphabetic characters: 14

Alternate Array Syntax

The square brackets of the array type in a declaration can be associated with the element type or with the name of the array

Therefore the following two declarations are equivalent:

```
double[] prices;  
double prices[];
```

The **first format generally is more readable** and should be used

Initializer Lists

An *initializer list* can be used to instantiate and fill an array in one step

The values are delimited by **braces** and separated by commas

Examples:

```
int[] spades = {1,2,3,4,5,6,7,8,9,10,11,12,13};
```

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

```
String[] strs = {"asv", "asdas", "Wrq"};
```

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 list

An initializer list can be used only in the array declaration

Arrays as Parameters

An entire array can be passed as a parameter to a method

Because an array is an object, when an entire array is passed as a parameter, a copy of the reference to the original array is passed

A method that receives an array as a parameter can permanently change an element of the array because it is referring to the original element value

The method cannot permanently change the reference to the array itself because a copy of the original reference is sent to the method. These rules are consistent with the rules that govern any object type

Arrays of Objects

The elements of an array can be object references

The following declaration reserves space to store 5 references to `String` objects

```
String[] words = new String[5];
```

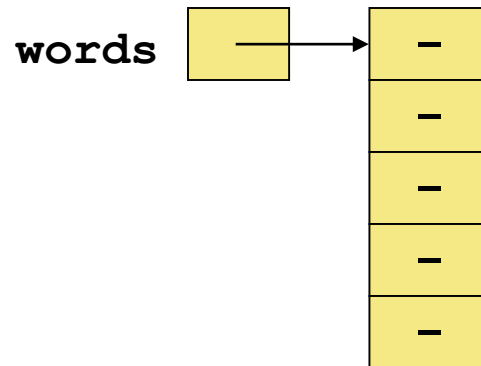
It does NOT create the `String` objects themselves

Initially an array of objects holds `null` references

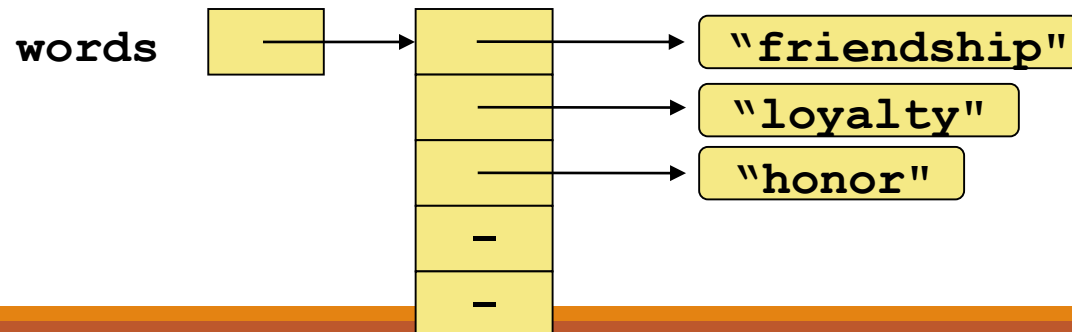
Each object stored in an array must be instantiated separately

Arrays of Objects

The `words` array when initially declared:



After some `String` objects are created and stored in the array:



Arrays of Objects

Keep in mind that `String` objects can be created using literals

The following declaration creates an array object called `verbs` and fills it with four `String` objects created using string literals

```
String[] verbs = {"play", "work", "eat",  
                  "sleep", "run"};
```

Variable Length Parameter Lists

Suppose we wanted to create a method that processed a different amount of data from one invocation to the next

For example, let's define a method called `average` that returns the average of a set of integer parameters

```
// one call to average three values
```

```
mean1 = average (42, 69, 37);
```

```
// another call to average seven values
```

```
mean2 = average (35, 43, 93, 23, 40, 21, 75);
```

Variable Length Parameter Lists

We could define overloaded versions of the `average` method

- Downside: we would need a separate version of the method for each additional parameter

We could define the method to accept an array of integers

- Downside: we would have to create the array and store the integers prior to calling the method each time

Instead, Java provides a convenient way to create *variable length parameter lists*

Variable Length Parameter Lists

Using special syntax in the formal parameter list, we can define a method to accept any number of parameters of the same type

For each call, the parameters are automatically put into an array for easy processing in the method

Indicates a variable length parameter list

```
public double average (int ... list)
{
    // whatever
}
```

**element
type**

**array
name**

Variable Length Parameter Lists

```
public double average (int ... list)
{
    double result = 0.0;

    if (list.length != 0)
    {
        int sum = 0;
        for (int num : list)
            sum += num;
        result = (double)sum / list.length;
    }

    return result;
}
```

Variable Length Parameter Lists

The type of the parameter can be any primitive or object type

```
public void printGrades (Grade ... grades)
{
    for (Grade letterGrade : grades)
        System.out.println (letterGrade);
}
```

Quick Check

Write method called `distance` that accepts a variable number of integers (which each represent the distance of one leg of a trip) and returns the total distance of the trip.

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


Variable Length Parameter Lists

A method that accepts a variable number of parameters can also accept other parameters

The following method accepts an `int`, a `String` object, and a variable number of `double` values into an array called `nums`

```
public void test (int count, String name,  
                  double ... nums)  
{  
    // whatever  
}
```

Variable Length Parameter Lists

The varying number of parameters must come last in the formal arguments

A method cannot accept two sets of varying parameters

Constructors can also be set up to accept a variable number of parameters

Two-Dimensional Arrays

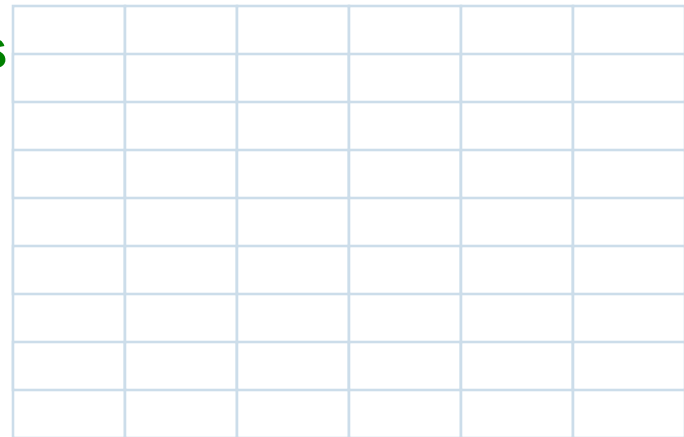
A *one-dimensional array* stores a list of elements

A *two-dimensional array* can be thought of as a table of elements, with rows and columns

**one
dimension**



**two
dimensions**



Two-Dimensional Arrays

To be precise, in Java a two-dimensional array is an array of arrays

A two-dimensional array is declared by specifying the size of each dimension separately:

```
int[][] table = new int[12][50];
```

A array element is referenced using two index values:

```
value = table[3][6]
```

```
//*****  
// SodaSurvey.java          Author: Lewis/Loftus  
//  
// Demonstrates the use of a two-dimensional array.  
//*****
```

```
import java.text.DecimalFormat;
```

```
public class SodaSurvey
```

```
{  
    //-----  
    // Determines and prints the average of each row (soda) and each  
    // column (respondent) of the survey scores.  
    //-----  
    public static void main (String[] args)  
    {  
        int[][] scores = { {3, 4, 5, 2, 1, 4, 3, 2, 4, 4},  
                             {2, 4, 3, 4, 3, 3, 2, 1, 2, 2},  
                             {3, 5, 4, 5, 5, 3, 2, 5, 5, 5},  
                             {1, 1, 1, 3, 1, 2, 1, 3, 2, 4} };  
  
        final int SODAS = scores.length;  
        final int PEOPLE = scores[0].length;  
  
        int[] sodaSum = new int[SODAS];  
        int[] personSum = new int[PEOPLE];  
    }  
}
```

continue

continue

```
for (int soda=0; soda < SODAS; soda++)
    for (int person=0; person < PEOPLE; person++)
    {
        sodaSum[soda] += scores[soda][person];
        personSum[person] += scores[soda][person];
    }

DecimalFormat fmt = new DecimalFormat ("0.##");
System.out.println ("Averages:\n");

for (int soda=0; soda < SODAS; soda++)
    System.out.println ("Soda #" + (soda+1) + ": " +
        fmt.format ((float)sodaSum[soda]/PEOPLE));

System.out.println ();
for (int person=0; person < PEOPLE; person++)
    System.out.println ("Person #" + (person+1) + ": " +
        fmt.format ((float)personSum[person]/SODAS));
}
```

continue

```
for (int soda=0;
    for (int person
    {
        sodaSum[soda]
        personSum[p
    }
}
```

```
DecimalFormat fmt
System.out.printl
```

```
for (int soda=0;
    System.out.pri
        fmt
```

```
System.out.printl
```

```
for (int person=0
    System.out.pri
        fmt
```

```
}
```

```
}
```

Output

Averages:

Soda #1: 3.2
Soda #2: 2.6
Soda #3: 4.2
Soda #4: 1.9

Person #1: 2.2
Person #2: 3.5
Person #3: 3.2
Person #4: 3.5
Person #5: 2.5
Person #6: 3
Person #7: 2
Person #8: 2.8
Person #9: 3.2
Person #10: 3.8

```
person++)
```

```
son];  
[person];
```

```
"0.#");
```

```
+1) + ": " +  
m[soda]/PEOPLE));
```

```
son++)  
rson+1) + ": " +  
Sum[person]/SODAS));
```

Multidimensional Arrays

An array can have many dimensions – if it has more than one dimension, it is called a *multidimensional array*

Each dimension subdivides the previous one into the specified number of elements

Each dimension has its own `length` constant

Because each dimension is an array of array references, the arrays within one dimension can be of different lengths

These are sometimes called *ragged arrays*



Text File I/O

- It is often useful to design programs that read input data from a text file or write output data to a text file
- Important classes for text file **input** (from the file):
 - BufferedReader
 - FileReader
- Important classes for text file **output** (to the file)
 - BufferedWriter
 - FileWriter
- Those classes are part of the java.io package, and must be imported into a program to be used: `import java.io.*;`

Read from a Text File

To read a text file `input.txt`: create a `BufferedReader` stream

```
BufferedReader reader = new BufferedReader(new  
    FileReader("input.txt"));
```

You need to put the text file under the current working directory of project (i.e., the same folder as "src" and "bin") to make the above line work

If it is located somewhere else, you should use the full path name

- Full path name: `"/Users/ota/java/TextIO/input.txt"` (mac) or `"D:\\Java\\Programs\\FileClassDemo.java"` (Windows)

Methods in BufferedReader

`readLine()`: read a line into a `String`

`close()`: **close** a `BufferedReader` **stream**

A typical usage (print each line one by one):

```
String line = null;
while( (line = reader.readLine()) != null)
    System.out.println(line);
reader.close();
```

Methods in BufferedReader

No methods to read numbers directly, so read numbers as Strings and then convert them

```
line = reader.readLine();  
int i = Integer.parseInt(line);
```

```
line = reader.readLine();  
double d = Double.parseDouble (line);
```

```
line = reader.readLine();  
boolean b = Boolean.parseBoolean(line);
```

Write to a Text File

To write to a text file `output.txt`: create a `BufferedWriter` stream

```
BufferedWriter writer = new BufferedWriter(new  
    FileWriter("output.txt"));
```

"output.txt" will be written to the current working directory of project (i.e., the same folder as "src" and "bin")

Can use full name to specify the destination as well

If a file with the same name already exists, it will be **overwritten** (data in the original file is lost)

Methods in BufferedWriter

`write(String s)`: write a String s to the file (without a new line)

`newline()`: write a new line to the file

`close()`: close a BufferedWriter stream

No methods to write numbers directly, so need to convert them into Strings before calling the method `write`

```
int i = 10;
```

```
writer.write(Integer.toString(i));
```

```
writer.newLine();
```

```
double d = 20.2;
```

```
writer.write(Double.toString(d));
```

LOGIC PUZZLE

During a recent police investigation, Chief Inspector Stone was interviewing five local villains to try and identify who stole Mrs Archer's cake from the mid-summers fayre. Below is a summary of their statements:

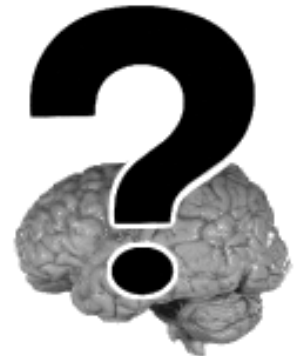
Arnold: it wasn't Edward
it was Brian

Brian: it wasn't Charles
it wasn't Edward

Charles: it was Edward
it wasn't Arnold

Derek: it was Charles
it was Brian

Edward: it was Derek
it wasn't Arnold



It was well known that each suspect told exactly one lie. Can you determine who stole the cake?

Hint: Look at Brian's statements first.

Answer: Charles committed the terrible crime.

Looking at Brian's statements, one of the statements was a lie and the other was the truth. Therefore it must have been either Charles or Edward.

Looking at Derek's statements, for the same reason, it was either Charles or Brian.

Therefore it must have been Charles who committed the crime. Double checking this against the other statements confirms this.

Group Exercises

Ex: 8.1

Ex: 8.2

Ex: 8.4

Ex: 8.5

Assignment for Class 21

Review BasicArray, ReverseOrder, LetterCount, Primes, GradeRange, Grade, Movies, DVDCollection, DVD, SodaSurvey

Read Chapter 7.1, 7.2, 7.3, 7.4, 7.9