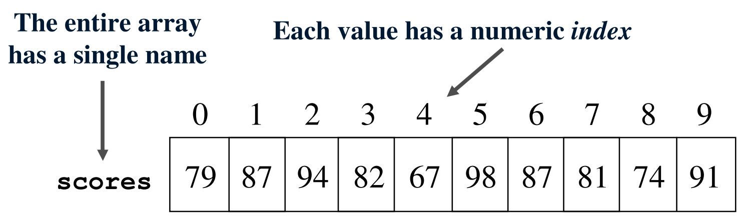
COMP 202 – Week 9

- **Arrays are objects that help us organize large amounts of information**

- **This week we focus on:**
  - **array declaration and use**
  - **arrays of objects**
  - **multidimensional arrays**
  - **the `ArrayList` class**
  - **the foreach statement**
  - **methods with variable length parameter list**

Arrays

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


The entire array has a single name
Each value has a numeric *index*
0 1 2 3 4 5 6 7 8 9
scores
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 brackets
- For example, the expression

```
scores[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 `scores` array could be declared as follows:

```
int[] scores = 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 `scores` is `int[]` (an array of integers)
- It is set to a new array object that can hold 10 integers
- See `BasicArray.java`

Declaring Arrays

- Some examples of array declarations:

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

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

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

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):

```
scores.length
```

- Note that `length` holds the number of elements, not the largest index
- See `ReverseNumbers.java`
- See `TravelLog.java`
- See `LetterCount.java`

Array Declarations Revisited

- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore the following declarations are equivalent:

```
float[] prices;
float prices[];
```

- The first format is generally more readable

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
- See `Primes.java`

Arrays as Parameters

- An entire array can be passed to a method as a parameter
  `convert(int[] aList) {…}` → `convert(units);`
- 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)`

Arrays of Objects

- The elements of an array can be object references
- The following declaration reserves space to store 25 references to `String` objects

```
String[] words = new String[25];
```

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

Command-Line Arguments

- The signature of the `main` method indicates that it takes an array of `String` objects as a parameter
- These values come from command-line arguments that are provided when the interpreter is invoked
- For example, the following invocation of the interpreter passes an array of three `String` objects into main:

```
> java DriverProg rain dogs cats
```

- These strings are stored at indexes 0-2 of the parameter
- See `NameTag.java`

Arrays of Objects

- 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
- See `FeedTheLitter.java`
- See `Tunes.java`
- See `CDCollection.java`
- See `CD.java`

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
- See `TwoDArray.java`

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

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
- See **`Beatles.java`**

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()`

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
- Therefore, the implementation of `ArrayList` in the API is not very efficient for inserting elements

The Foreach statement

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

is equivalent to

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

Foreach statement

- **The *foreach* statement works on arrays and on any object whose class implements the `Iterable` interface (which consists of only one method which returns an implementation of the `Iterator` interface), such as the `ArrayList` 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);
```

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;
}
```

- 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