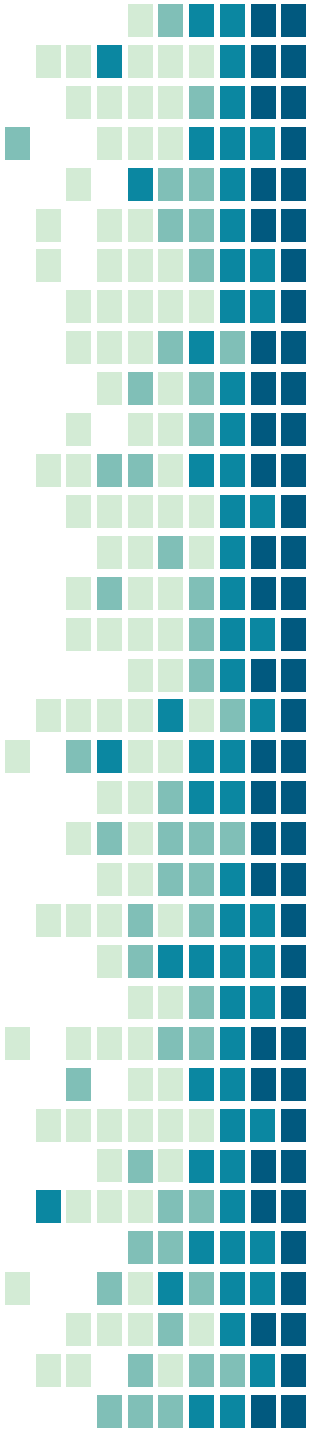
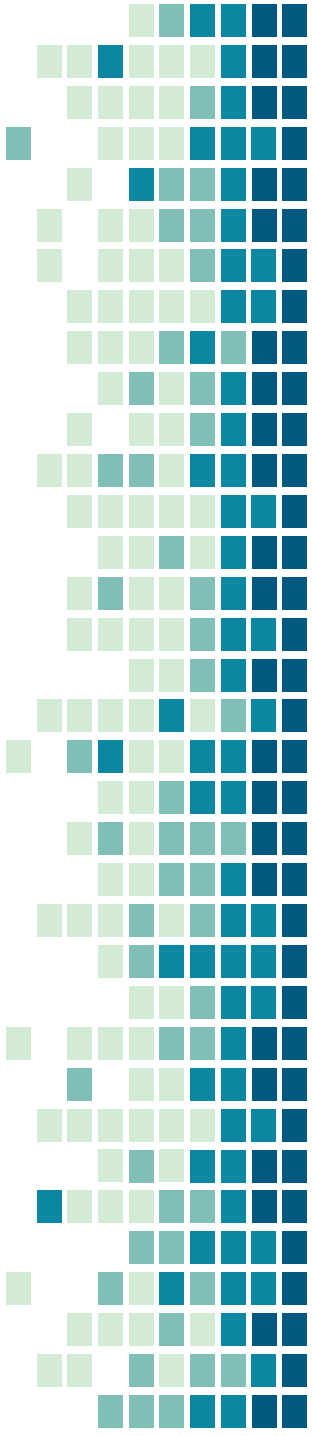
Arrays



Arrays

Arrays are objects that help us organize large amounts of information

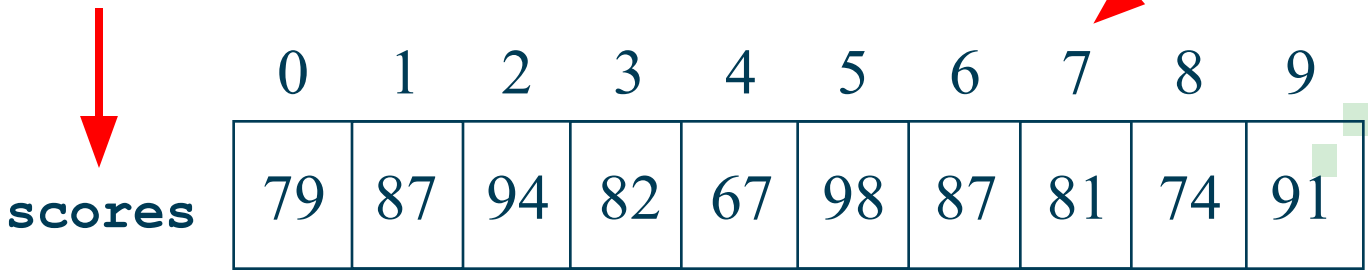


Arrays

The entire array has a single name

An *array* is an ordered list of values

Each value has a numeric *index*

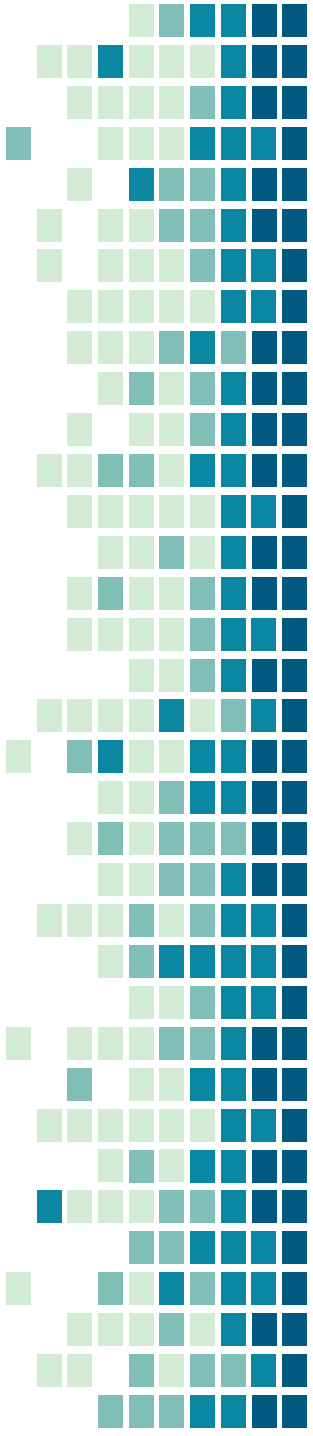


An array of size N is indexed from zero to N-1

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



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]);
```

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

In Java, the array itself is an object

Therefore the name of the array is a object reference variable, and the array itself must be instantiated

Declaring Arrays

The `scores` array could be declared as follows:

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

The type of the variable `scores` is `int[]` (an array of integers)

Note that the type of the array does not specify its size, but each object of that type has a specific size

The reference variable `scores` is set to a new array object that can hold 10 integers

Declaring Arrays

Some examples of array declarations:

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

boolean[] flags;

flags = new boolean[20];

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

Bounds Checking

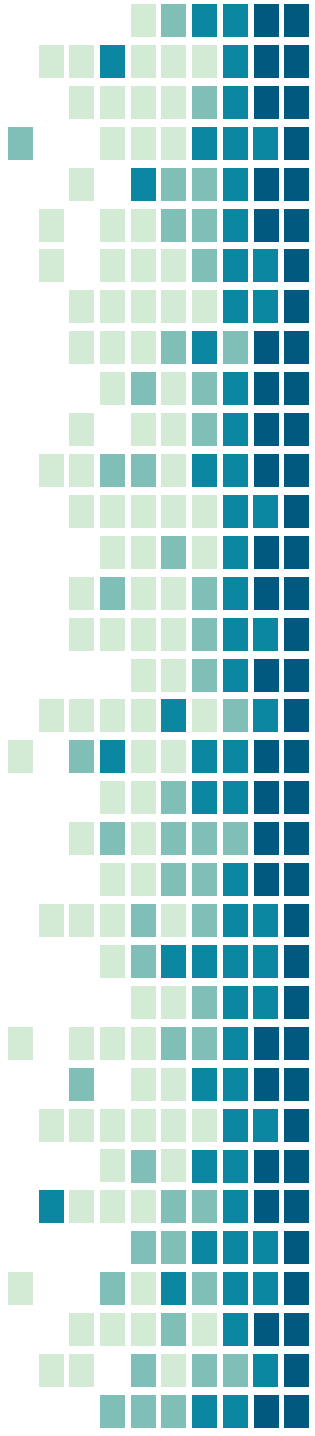
Once an array is created, it has a fixed size

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 throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds

This is called *automatic bounds checking*



Alternate Array Syntax

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 generally is 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 only in the array declaration

See Primes.java (page 330)

Arrays as Parameters

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

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 within the method changes the original

An array element can be passed to a method as well, and follows the parameter passing rules of that element's type

Arrays of Objects

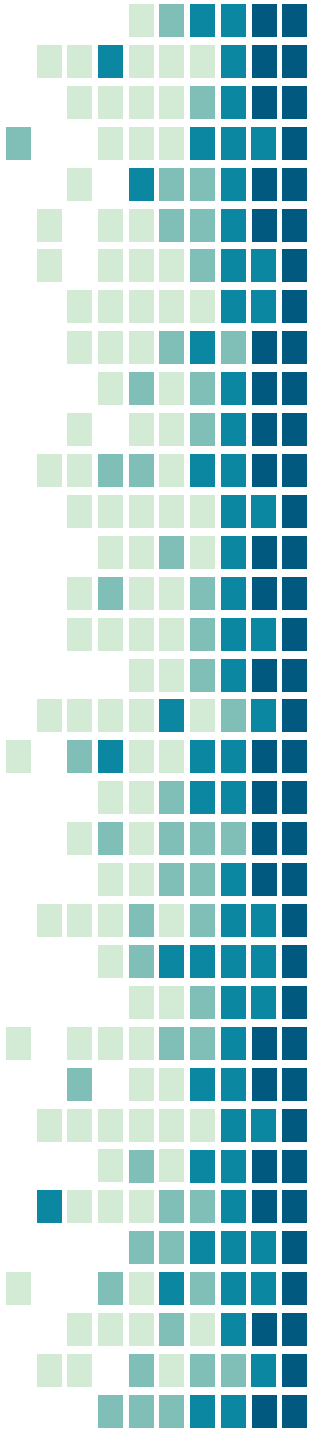
The elements of an array can be object references

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



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 StateEval pennsylvania texas arizona
```

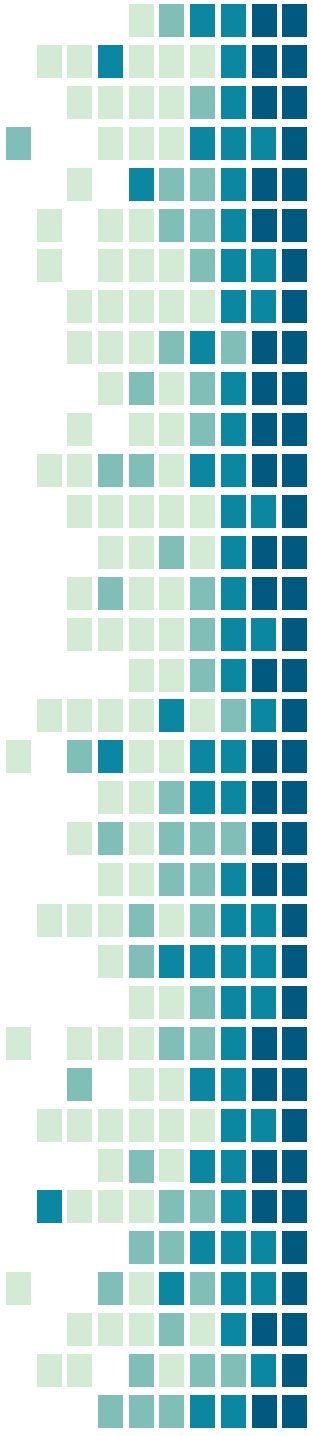
These strings are stored at indexes 0-2 of the parameter

Arrays of Objects

Objects can have arrays as instance variables

Many useful structures can be created with arrays and objects

The software designer must determine carefully an organization of data and objects that makes sense for the situation



Comparing Sorts

Both Selection and Insertion sorts are similar in efficiency

They both have outer loops that scan all elements, and inner loops that compare the value of the outer loop with almost all values in the list

Approximately n^2 number of comparisons are made to sort a list of size n

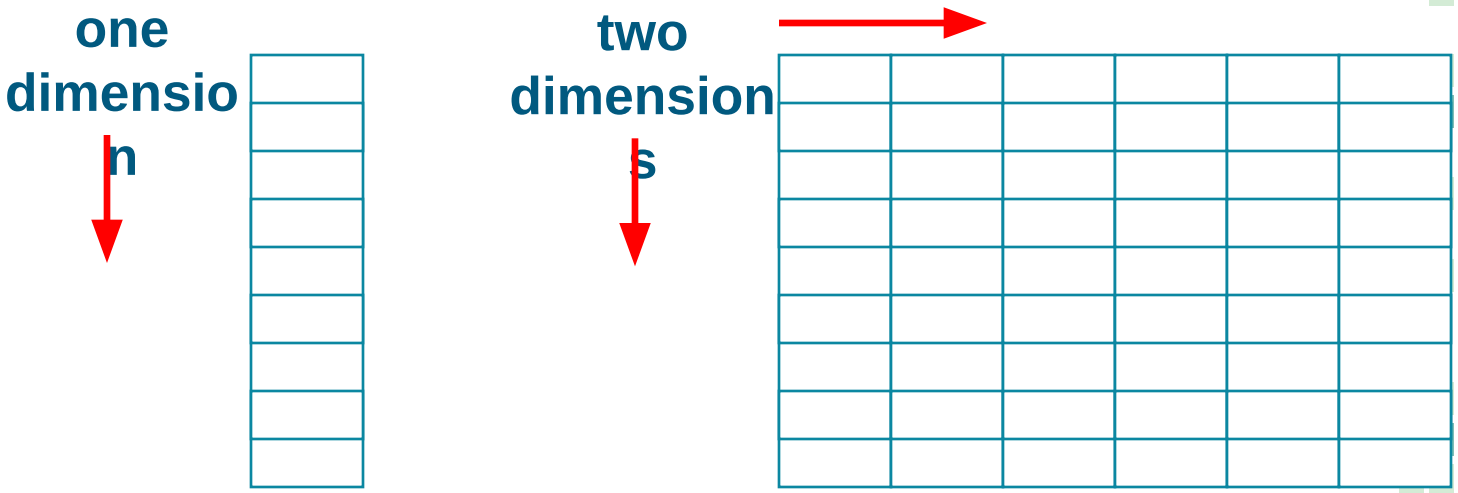
We therefore say that these sorts are of *order* n^2

Other sorts are more efficient: *order* $n \log_2 n$

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



Two-Dimensional Arrays

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

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

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

A two-dimensional array element is referenced using two index values

```
value = scores[3][6]
```

The array stored in one row or column can be specified using one index

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 array 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

The ArrayList Class

- The `ArrayList` class is part of the `java.util` package
- Like an array, it can store a list of values and reference them with an index
- Unlike an array, an `ArrayList` object grows and shrinks as needed
- Items can be inserted or removed with a single method invocation
- It stores references to the `Object` class, which allows it to store any kind of object