the UNIVERSITY of GREENWICH

COMP1618
Lecture 8 Arrays

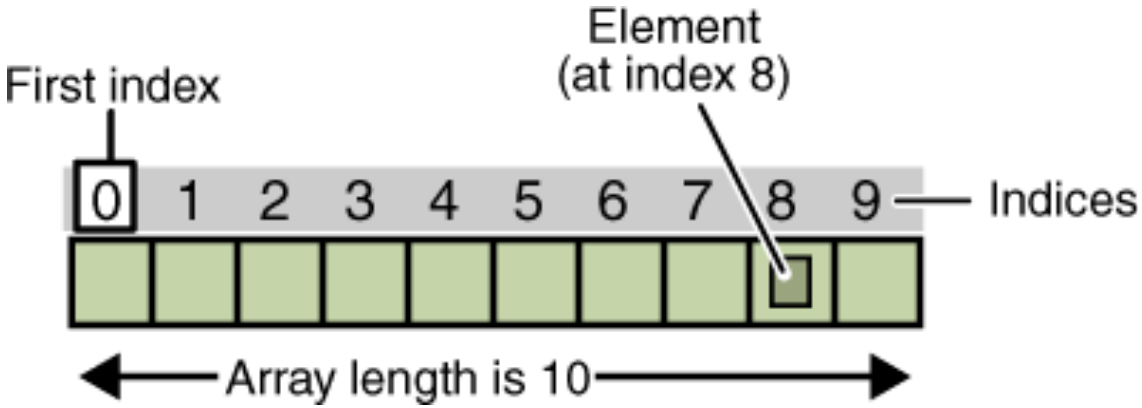
the UNIVERSITY of GREENWICH

Lecture Objectives

- Arrays
- Passing Arrays as Arguments to Methods
- Some Useful Array Operations
- Copy Arrays
- The Sort/Search Algorithm

Introduction to Arrays

- Arrays allow us to create a collection of objects that are indexed.
- An array can store any type of data but only one type of data at a time.
- An array is a list of data elements.



Sample

create an array with random values	`double[] a = new double[n];`
	`for (int i = 0; i < n; i++)`
	`  a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < n; i++)` `  System.out.println(a[i]);`

find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < n; i++)`
	`  if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < n; i++)`
	`  sum += a[i];`
	`double average = sum / n;`

reverse the values within an array	`for (int i = 0; i < n/2; i++)`
	`{`
	`  double temp = a[i];`
	`  a[i] = a[n-1-i];`
	`  a[n-i-1] = temp;`
	`}`

copy sequence of values to another array	`double[] b = new double[n];`
	`for (int i = 0; i < n; i++)`
	`  b[i] = a[i];`

Array Declaration

- Arrays are declared as:

```
int[] numbers or int numbers[];
```

- Multiple arrays can be declared on the same line.

```
int[] numbers, codes, scores;
```

- Array variable must have brackets.

```
int numbers[], codes[], scores;
```

  - The `scores` variable in this instance is simply an `int` variable.

Creating Arrays

```
int[] numbers = new int[6];
```

- Create a array and assigns its address to the `numbers` variable

0	0	0	0	0	0
index 0	index 1	index 2	index 3	index 4	index 5

- More examples:

```
float[] temperatures = new float[100];
char[] letters = new char[41];
long[] units = new long[50];
double[] sizes = new double[1200];
```

Creating Arrays

- The array may be a literal value, a constant, or variable (non-negative)

```
final int ARRAY_SIZE = 6;
int[] numbers = new int[ARRAY_SIZE];
```

- Once created, an array size is fixed and cannot be changed.

Array Initialization

- When relatively few items need to be initialized, an initialization list can be used to initialize the array.

```
int[]days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

- The numbers in the list are stored in the array in order:
  - `days[0]` is assigned 31,
  - `days[1]` is assigned 28,
  - `days[2]` is assigned 31,
  - `days[3]` is assigned 30,
  - etc.
- See example: ArrayInitialization.java

Accessing the Elements of an Array

- An element of an array is accessed by:
  - the array name with an index

```
numbers[0] = 20;
```

20	0	0	0	0	0
numbers[0]	numbers[1]	numbers[2]	numbers[3]	numbers[4]	numbers[5]

- See more example: ArrayDemo1.java; ArrayDemo2.java

Array Length

- The length of an array can be obtained via its `length` constant.

```
double[] temperatures = new double[25];
int size = temperatures.length;
```

- The `length` can be used in a loop to provide automatic bounding.

```
for(int i = 0; i<temperatures.length; i++)
{
   System.out.println("Temperature " + i ": "+
   temperatures[i]);
}
```

Bounds Checking

- Array index starts from 0 till length - 1.

```
int values = new int[10];   where would the index end?
```

- Off-by-one error

```
int[] numbers = new int[100];
for (int i = 0; i <= 100;i++)
  numbers[i] = 99;
```

  - There is NO `numbers[100]` ✗

This code would throw an exception:

`ArrayIndexOutOfBoundsException`

Another way to iterate array elements

- Another way to iterate the elements in an array

```
for(datatype elementVariable : arrayName)
   Sstatements;
```

- Example:

```
int[] numbers = {3, 6, 9};
for(int num : numbers)
{
  System.out.println("The next value is" + num);
}
```

Specifying Array Size

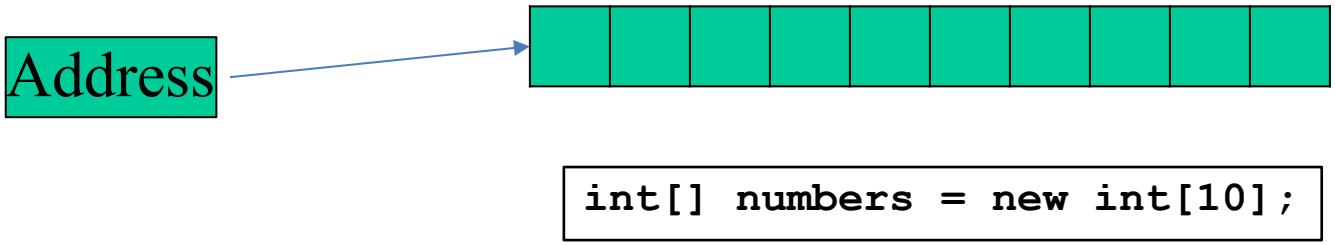
- You can let the user specify the size of an array:

```
int numTests;
int[] tests;
Scanner keyboard = new Scanner(System.in);
System.out.print("How many tests " +
                 "do you have? ");
numTests = keyboard.nextInt();
tests = new int[numTests];
```

- See example: DisplayTestScores.java

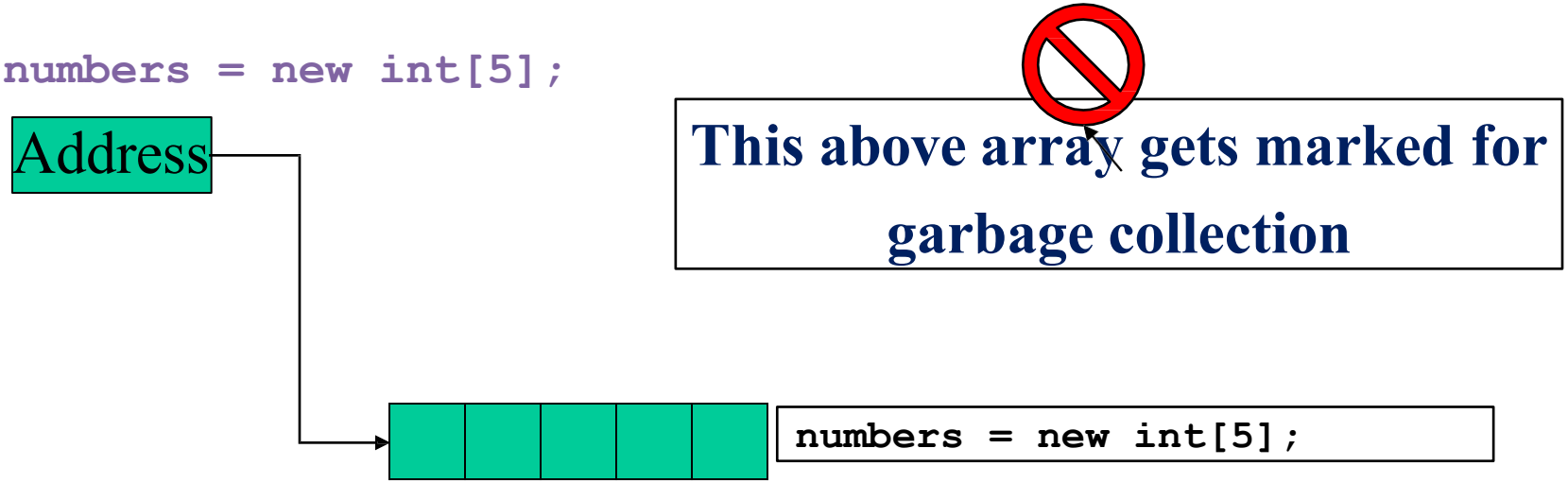
Reassigning Array References

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



An array reference can be assigned to another array of the same type.

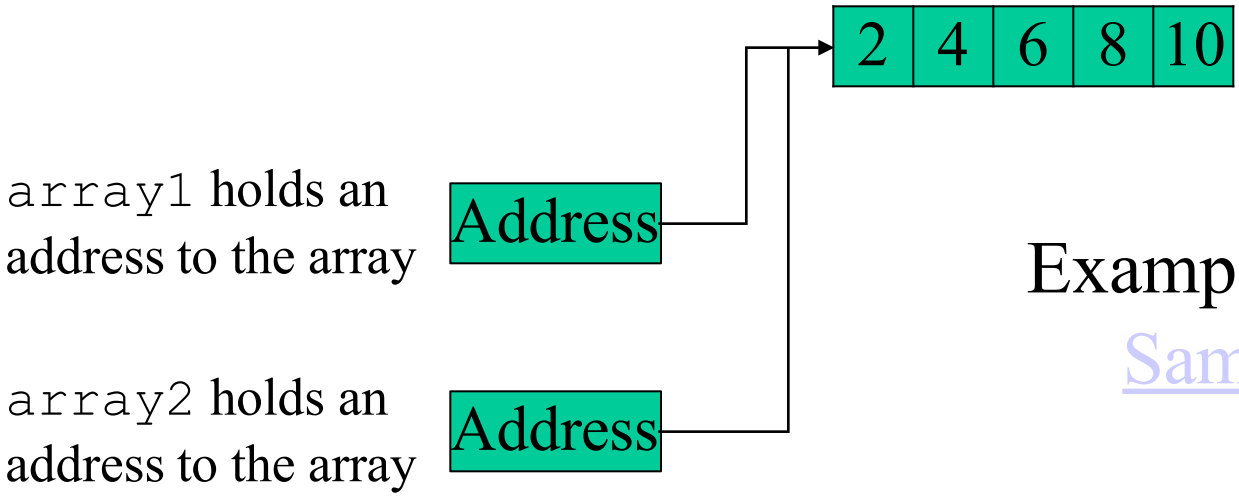
```
numbers = new int[5];
```



Copying Arrays

- This is **NOT** the way we copy the content of an array.

```
int[] array1 = { 2, 4, 6, 8, 10 };
int[] array2 = array1; // This does not copy array1.
```



Example:

SameArray.java

The right way to Copy Arrays

- You cannot copy an array by assigning one reference variable to another.
- You need to copy the individual elements of one array to another.

```
int[] firstArray = {5, 10, 15, 20, 25 };
int[] secondArray = new int[5];
for (int i = 0; i < firstArray.length; i++)
  secondArray[i] = firstArray[i]
```

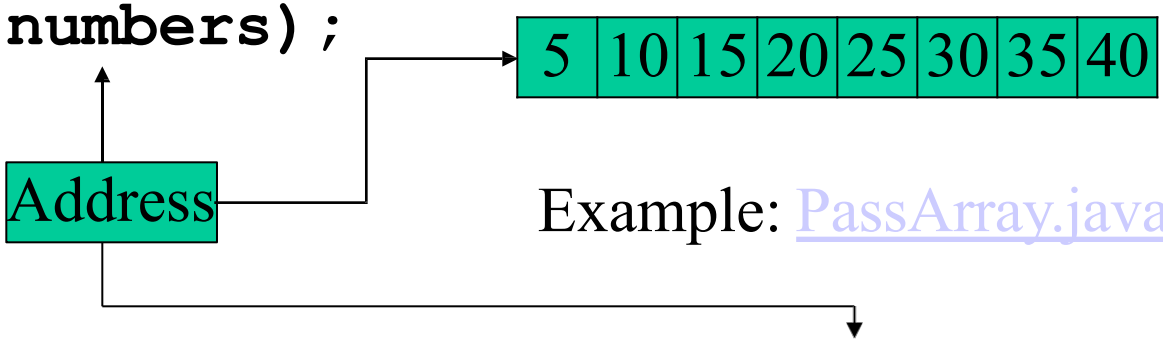
Or

```
int[] secondArray = firstArray.clone();
```

Passing Arrays as Arguments

- Arrays are objects.
- Their references can be passed to methods like any other object reference variable.

```
showArray(numbers);
```



Example: PassArray.java

```
public static void showArray(int[] array)
{
   for (int i = 0; i < array.length; i++)
      System.out.print(array[i] + " ");
}
```

Deep Comparison of Two Arrays

- The == operator determines only whether two array references point to the same array object.

```
int inarr1[] = { 1, 2, 3 };
int inarr2[] = { 1, 2, 3 };
    if (arr1 == arr2))
            System.out.println("Same");
```

The output will NOT be "Same"

- To compare the contents of two arrays:

```
// Importing required classes
import java.util.Arrays;

int inarr1[] = { 1, 2, 3 };
int inarr2[] = { 1, 2, 3 };
    if (Arrays.deepEquals(arr1, arr2))
            System.out.println("Same");
```

The output will be Same

Useful Array Operations

- **Finding the Highest Value**

```
int [] numbers = new int[50];
int highest = numbers[0];
for (int i = 1; i < numbers.length; i++)
{
      if (numbers[i] > highest)
              highest = numbers[i];
}
```

- **Finding the Lowest Value**

```
int lowest = numbers[0];
for (int i = 1; i < numbers.length; i++)
{
      if (numbers[i] < lowest)
              lowest = numbers[i];
}
```

Useful Array Operations

- Summing Array Elements:

```
int total = 0; // Initialize accumulator
for (int i = 0; i < units.length; i++)
  total += units[i];
```

- Averaging Array Elements:

```
double total = 0; // Initialize accumulator
double average; // Will hold the average
for (int i = 0; i < scores.length; i++)
 total += scores[i];
average = total / scores.length;
```

- Example: SalesData.java, Sales.java

Partially Filled Arrays

- Typically, if the amount of data that an array must hold is unknown:
  - size the array to the largest expected number of elements.
  - use a counting variable to keep track of how much valid data is in the array.

```
…
int[] array = new int[100];
int count = 0;
…
   System.out.print("Enter a number or -1 to quit: ");
   number = keyboard.nextInt();
   while (number != -1 && count <= 99)
   {
      array[count] = number;
      count++;
      System.out.print("Enter a number or -1 to quit: ");
      number = keyboard.nextInt();
…  }
```

`input`, `number` and `keyboard` were previously declared and `keyboard` references a `Scanner` object

Arrays and Files

- Saving the contents of an array to a file:

```
int[] numbers = {10, 20, 30, 40, 50};

PrintWriter outputFile =
     new PrintWriter ("Values.txt");

for (int i = 0; i < numbers.length; i++)
  outputFile.println(numbers[i]);

outputFile.close();
```

Arrays and Files

- Reading the contents of a file into an array:

```
final int SIZE = 5; // Assuming we know the size.
int[] numbers = new int[SIZE];
int i = 0;
File file = new File ("Values.txt");
Scanner inputFile = new Scanner(file);
while (inputFile.hasNext() && i < numbers.length)
{
  numbers[i] = inputFile.nextInt();
  i++;
}
inputFile.close();
```

Returning an Array Reference

- A method can return a reference to an array.
- The return type of the method must be declared as an array of the right type.

```
public static double[] getArray()
{
  double[] myArray = { 1.2, 2.3, 4.5, 6.7, 8.9 };
  return myArray;
}
```

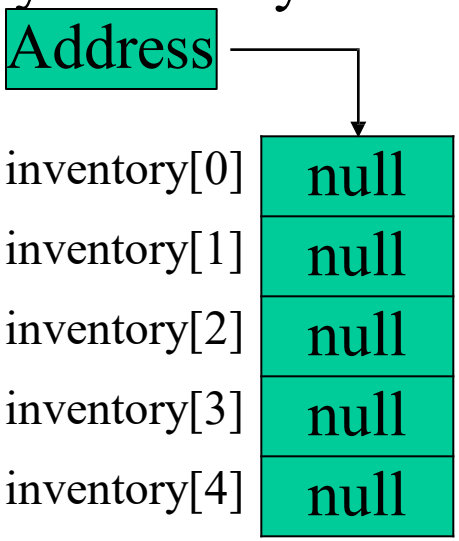
- The `getArray` method is a public static method that returns an array of doubles.
- See example: ReturnArray.java

Arrays of Objects

- Since Strings are objects, we know that arrays can contain objects.

```
InventoryItem[] inventory = new InventoryItem[5];
```

The *inventory* variable holds the address of an InventoryItem array.



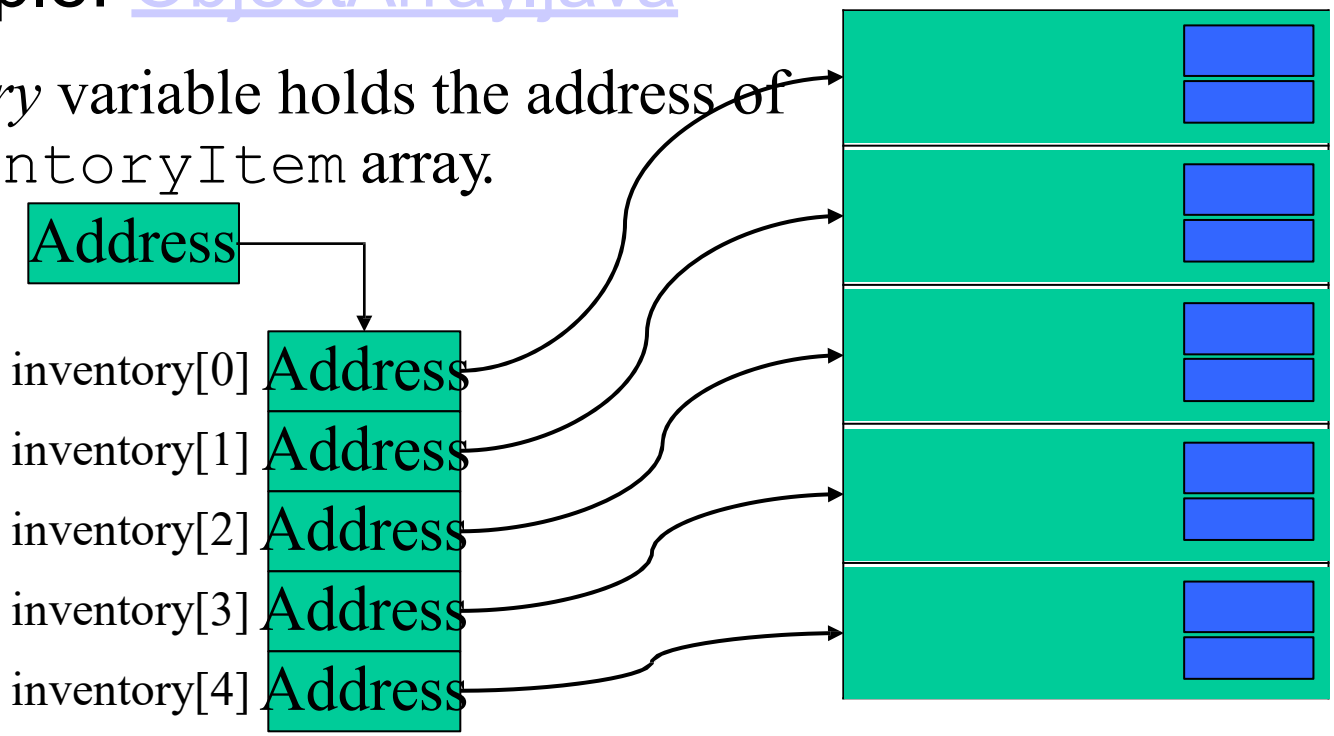
Arrays of Objects

- Each element needs to be initialized.

```
for (int i = 0; i < inventory.length; i++)
  inventory[i] = new InventoryItem();
```

- Example: ObjectArray.java

The *inventory* variable holds the address of an `InventoryItem` array.



The Sequential Search Algorithm

- **Sequential Search** aims to locate a specific item in a larger collection of data.
- It uses a loop to:
  - sequentially to through an array,
  - compare each element with the search value, and
  - stop when
    - the value is found or
    - the end of the array is encountered.
- See example: SearchArray.java; TestSearch.java

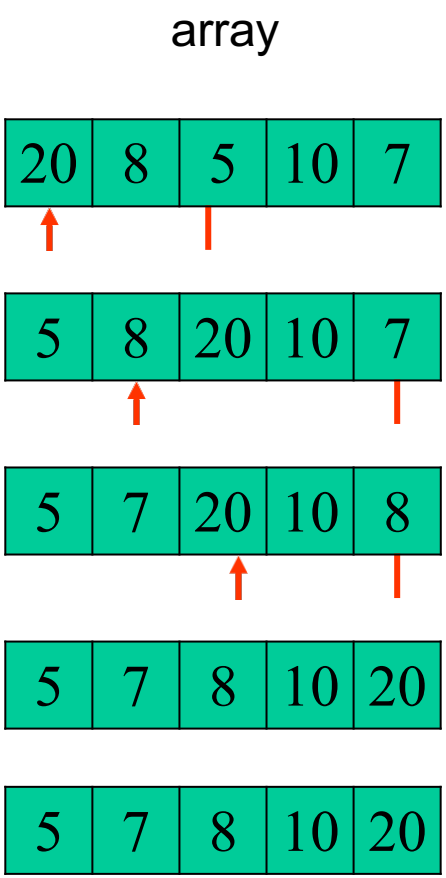
Selection Sort

Selection sort: arrange elements of an array in a specific order.

Process:

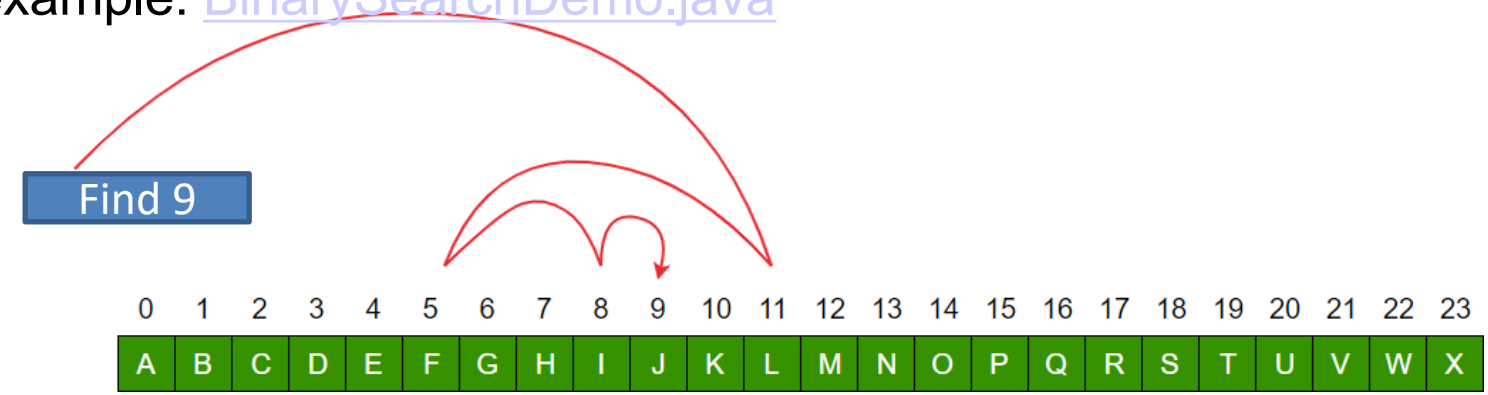
- Find the minimum value in the array and put it at beginning (array [0]) .
- Then the next minimum value is located and moved to (array [1]) .
- This process continues until all of the elements have been placed in their proper order.
- See example: SelectionSortDemo.java

  ArrayTools.java



Binary Search

- **Binary search**: find. the position of a target value within a sorted array
- **Process**:
  - It starts with the element in the middle of the array.
  - If that element is the desired value, the search is over.
  - Otherwise, the value in the middle element is either greater or less than the desired value
  - If it is greater than the desired value, search in the first half of the array.
  - Otherwise, search the last half of the array.
  - Repeat as needed while adjusting start and end points of the search.
- See example: BinarySearchDemo.java



Two-Dimensional Arrays

- A two-dimensional array is an array of arrays.
- It can be thought of as having rows and columns.

	column 0	column 1	column 2	column 3
row 0				
row 1				
row 2				
row 3				

Two-Dimensional Arrays

- Declaring a two-dimensional array requires two sets of brackets and two size declarators
  - The first one is for the number of rows
  - The second one is for the number of columns.

```
double[][] scores = new double[3][4];
```

two dimensional array

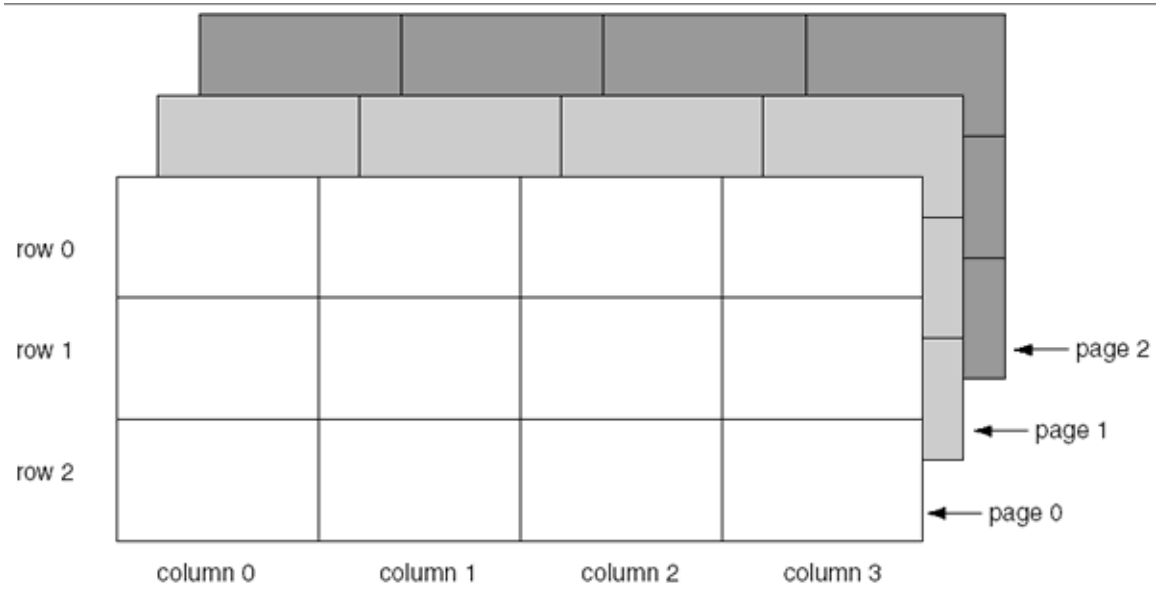
rows

columns

- The two sets of brackets in the data type indicate that the scores variable will reference a two-dimensional array.

More Than Two Dimensions

- Java does not limit the number of dimensions of an array.
- More than three dimensions is hard to visualize, but can be useful in some programming problems.



the UNIVERSITY of GREENWICH

Summary

- We have looked at:
  - Introduction to Arrays
  - Array Contents
  - Arrays as Arguments
  - Array Operations and Sort/Search Algorithm