CSEN202 – Introduction to Computer Programming

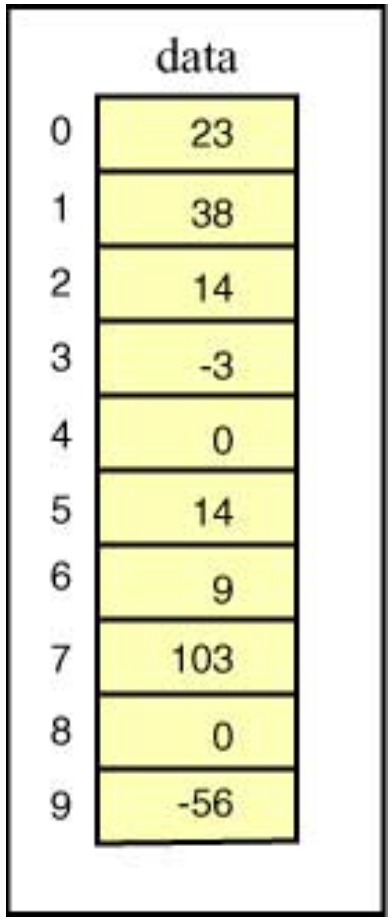
Topics:

Arrays
Common Array Algorithms

Prof. Dr. Slim Abdennadher

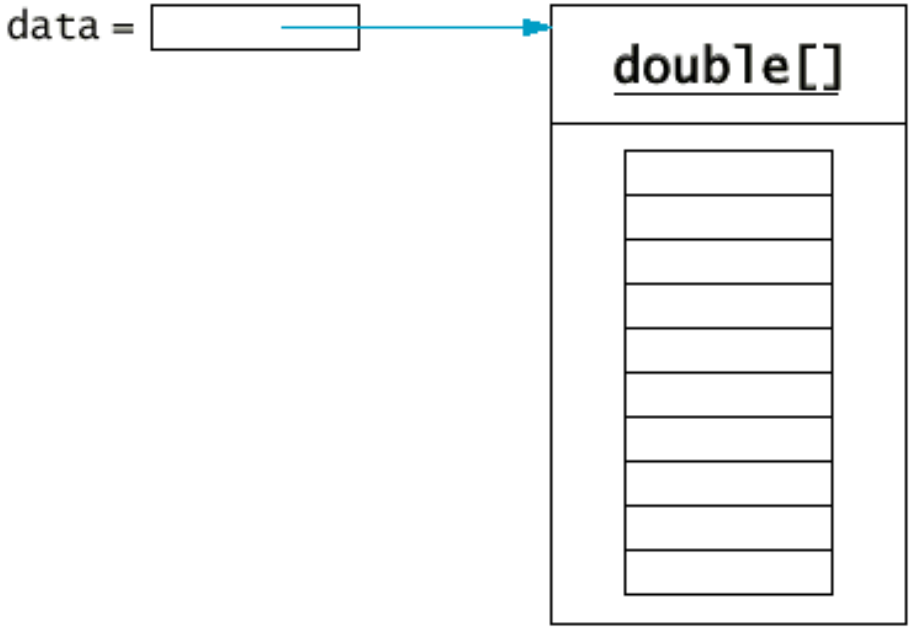
Scalar Variables

- Individual variables are classified as scalars
- A scalar can assume exactly one value at a time
- As we have seen, individual variables can be used to create some interesting and useful programs
- A scalar cannot represent a collection of data elements.



Arrays

- An array is a non-scalar reference variable



- Like any other variable, an array
  - can be local, class, or instance variable
  - must be declared before it is used

Array Declarations

- **Declaring a single array variable:**

```
type[] name;
```

- **Examples:**

```
int[] a;              // a is an array of integers
double[] list;        // list is  an array of floating-point numbers
```

- **Declaring multiple array variables in one statement:**

```
type[] name1, name2, ..., nameN;
```

- **Examples:**

```
char[] a, letters, cList;
boolean[] answerList, selections;
```

Allocating Arrays

- **The number of elements in an array is not determined until the array is created.**
- **An array is special kind of object**
  - **An array variable is therefore an object reference**
  - **Declaring an array does not allocate space for its contents**
  - **The new operator must be used to create the array with the proper size**

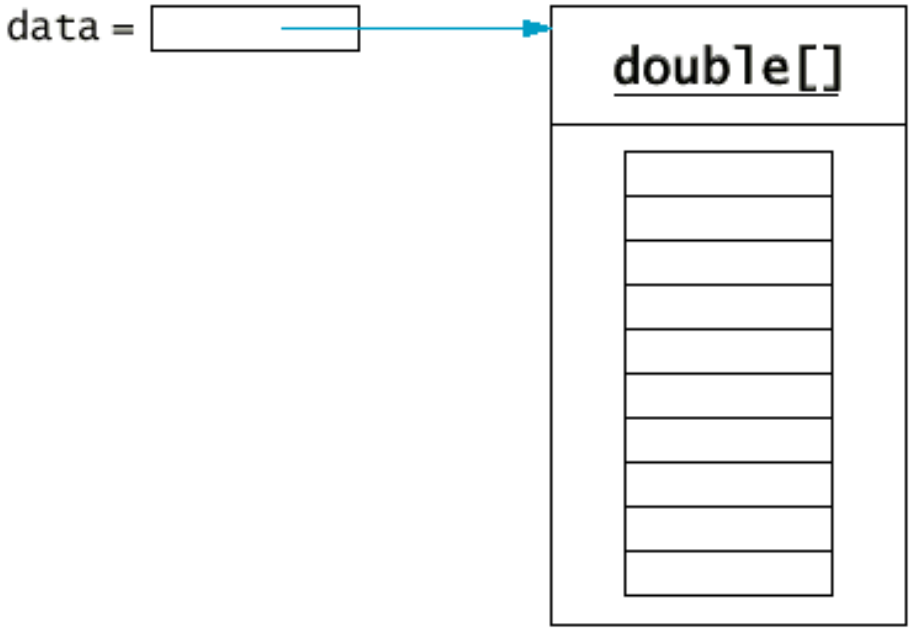
```
a = new char[10];                              \\ a holds ten characters
letters = new char[Console.In.readInt()]; \\ Number of characters
                                                 entered by user

answerList = new boolean[numValues];        \\ Size of answerList
                                                 depends on the value
                                                 of a variable
```

Creating an Array at Declaration

- **Arrays may be created when they are declared.**
- **Example**

```
double[] data = new double[10];
```

This statement declares an array of floating-point numbers and creates it with a capacity of ten elements.



Homogeneous Elements

- **Each array has a declared type.**
- **All elements stored in an array must be compatible with that array's declared type.**
- **Arrays are said to be homogeneous data structures.**

Initializing Arrays

- **It is possible to both create an array and initialize its contents simultaneously**
- **The elements are provided in a comma separated list within curly braces:**

```
type[] name = { value1, value2, ..., valueN };
```

- **Examples:**

```
int[] list = {20, 30, 40, 50};
double x = 10.5, y = 20.0, z = 0.25;          double[] a = {x, y, z}
```

- **Notice that new is not used here even though an array object is being created.**
- **It is possible to create an array with an initialization list after it has been declared. Notice that new must be used in this case.**

```
int[] a;
/* ... Do some stuff ... */
a = new int[] {10, 20, 30};
```

Using Arrays

- **Once an array has been properly declared and created it can be used within a program**
- **A programmer can use an array in one of two ways:**
  - **An element of the array can be used within a statement**
  - **A reference to the whole array itself can be used within a statement**

Accessing an Element

- One an array has been created, its elements can be accessed with `[]` operator
- The general form of an array access expression is:

```
name[ expression ]
```

- `expression` must evaluate to an integer:
  - an integer literal: `a[45]`
  - an integer variable: `a[x]`
  - an integer arithmetic expression: `a[x+2]`
  - an integer result of a method call that return `int`: `a[find(2)]`
  - an access to an integer array: `a[b[1]]`
- The square brackets and enclosed expression is sometimes called a subscript or index

Array Subscripts

If array `a` has been allocated to hold `n` elements, then

- **The first element in `a` is `a[0]`**
- **The last element in `a` is `a[n-1]`**

Example:

- **Create the array: `a = new int[5];`**

0	0	0	0	0
0	1	2	3	4

- **Reassign one of its elements: `a[2] = 6;`**

0	0	6	0	0
0	1	2	3	4

Out-of-bounds Access

- **The programmer must ensure that the subscript is within the bounds of the array**
- **Since the subscript can consist of an arbitrary integer expression whose value cannot be determined until run time, the compiler cannot check for out-of-bound array accesses**
- **A run-time error will occur if a program attempts an out-of-bounds array access**

Loops and Arrays

An array is most naturally traversed with a `for` loop

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

All arrays have a public constant attribute named `length` that returns the number of elements allocated for the array.

Loops and Arrays: Constructing an Array

```
import java.io.* ;
class InputArray
{

  public static void main ( String[] args ) throws IOException
  {
    BufferedReader inData =
        new BufferedReader ( new InputStreamReader( System.in ) );
    int[] array;

    // determine the array size and construct the array
    System.out.println( "What length is the array?" );
    int size  = Integer.parseInt( inData.readLine() );
    array     = new int[ size];
    // input the data
    for ( int index=0; index < array.length; index++)
    {
      System.out.println( "enter an integer: " );
```

Loops and Arrays: Constructing an Array

```
      array[ index ] = Integer.parseInt( inData.readLine() );
    }

    // write out the data
    for ( int index=0; index < array.length; index++ )
    {
      System.out.println( "array[ " + index + " ] = " + array[ index ] )
    }

  }
}
```

Loops and Arrays: Adding Elements of an Array

```
class SumArray
{
  public static void main ( String[] args )
  {
    double[] array =  { -47.39, 24.96, -1.02, 3.45, 14.21, 32.6} ;

    // declare and initialize the total
    double   total =   0.0 ;

    // add each element of the array to the total
    for ( int index=0; index < array.length; index++ )
    {
      total =  total + array[ index ]  ;
    }
    System.out.println("The total is: " + total );
  }
}
```

Array Parameters and Return Values

An array is simply an object reference; therefore, it can be used like any object reference.

An array can be

- **passed as an actual parameter to a method call and**
- **returned from a method as a return value.**

Formal Array Parameters

- **An array is specified as a formal parameter in a method definition with the same syntax as variable declaration**

```
public static int sumUp(int [] a)
     if (a != null) {
        int sum = 0;
        for (int i = 0; i < a.length; i++) {
            sum += a[i];
        }
     return sum;
     }
}
```

- **Calling code simply passes the array as it would any variable:**

```
int[] list = new int[100];
/* Intialize list (details omitted) */
// ... then call sumUp()
int sum = sumUp(list);
```

Array Return Value

- **An array can be returned from a method call**
- **This allows a method to return more than one value**
- **Example:**

```
public class Triangle {
     private int side1;
     private int side2;
     private int side3;

     public Triangle(int aSide1, int aSide2, int aSide3) {
          side1 = aSide1;
          side2 = aSide2;
          side3 = aSide;
     }

     public int[] getSides() {
          retrun new int[] {side1, side2, side};
     }
```

Array Return Value

```
    public void expand(int factor) {
        side1 *= factor;
        side2 *= factor;
        side3 *= factor;
    }

    public static void main(String[] args) {
        Triangle tri = new Triangle(3, 4, 5);
        tri.expand(2);
        // Print the new expanded sides
        int[] sides = tri.getSides();
        System.out.println("Side1: " + sides[0]);
        System.out.println("Side2: " + sides[1]);
        System.out.println("Side3: " + sides[2]);
        }
}
```

Nested Loops: Arrays

A Java method to print the result of the intersection of two arrays.

```
public class Intersect
{
    public static void intersect(int[] A, int[] B) {

     for(int i = 0; i < A.length; i++) {
                for(int j = 0; j < B.length; j++) {
                    if (A[i] == B[j])
                        System.out.print(A[i] + " ");
                }
            }
    }
}
```

Common Array Algorithms – Summing the numbers in an Array

- **Problem: Sum the numbers in an array**
- **Java Program:**

```
class SumArray {
  public static void main ( String[] args )  {
    double[] array =  { -47.39, 24.96, -1.02, 3.45, 14.21, 32.6, 19.

    // declare and initialize the total
    double   total =   0.0 ;

    // add each element of the array to the total
    for ( int index=0; index < array.length; index++ )
    {
      total =  total + array[ index ]  ;
    }
    System.out.println("The total is: " + total );
  }
}
```

Common Array Algorithms – Finding the Maximum of an Array

- **Problem:** **Find the largest integer in a list of integers.**
- **Algorithm:** **Initialize `max` to the first element in the array. A `for` loop is set up to look at every element in the array, starting with the beginning element, to see if that element "beats" the current maximum.**
- **Java Program:**

```
class MaxAlgorithm {
  public static void main ( String[] args ) {
    int[] array =  { -20, 19, 1, 5, -1, 27, 19, 5 } ;
    int   max;
    max = array[0];
    for ( int index=0; index < array.length; index++ )
    {
      if ( array[ index ] > max )     // examine the current element
        max = array[ index ];         // if it is the largest so far,
                                      // change max
    }
    System.out.println("The maximum of this array is: " + max ); } }
```

Common Array Algorithms – Copy Method

- **Problem: Copy the values from one array into another.**
- **Java Program:**

```
class ChangeArray
{
  static void  print ( int[] x )
  {
    for (int j=0; j < x.length; j++)
      System.out.print( x[j] + " " );
    System.out.println( );
  }

  // Copy source to target
  static void  copy (int[] source, int[] target)
  {
    for (int count=0; count<source.length; count++)
      target[ count ] = source[ count ];
  }

  public static void main(String[] args)
```

Common Array Algorithms – Copy Method

```
  {

    int[] s = {27, 19, 34, 5, 12} ;
    int[] t = new int[ s.length ];
    System.out.println( "Before copy:" );
    print( t );
    copy( s, t );
    System.out.println( "After copy:" );
    print( t );
  }
}
```

Common Array Algorithms – Linear Search

- **Problem: Find the target in the array of items, or report if the target is not present.**
- **Algorithm: Look through the slots of the array one by one, starting with the first slot and ending when either the target is found or every slot has been examined.**
- **Java Program:**

```
class Searcher {
  // return the index where found, or -1 if not found.
  public static int search( String[] array, String target )
  {
     for ( int j=0; j  < array.length; j++ )
       if ( array[j] != null )
         if ( array[j].equals( target ) ) return j ; // Target found.

     return -1 ; // Target not found
  }
}
```