

Chapter 6:

Arrays and the ArrayList Class

**Starting Out with Java:
From Control Structures through Objects**

Fifth Edition

by Tony Gaddis

Chapter Topics

Chapter 6 discusses the following main topics:

- Introduction to Arrays
- Processing Array Contents
- Passing Arrays as Arguments to Methods
- Some Useful Array Algorithms and Operations
- Returning Arrays from Methods
- String Arrays
- Arrays of Objects

Chapter Topics

Chapter 6 discusses the following main topics:

- The Sequential Search Algorithm
- Parallel Arrays
- Two-Dimensional Arrays

Introduction to Arrays

- Primitive variables are designed to hold only one value at a time.
- Arrays allow us to create a collection of like values 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.

Creating Arrays

- An array is an object so it needs an object reference.

```
// Declare a reference to an array that will hold integers.  
int[] numbers;
```

- The next step creates the array and assigns its address to the `numbers` variable.

```
// Create a new array that will hold 6 integers.  
numbers = new int[6];
```

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

Array element values are initialized to 0.

Array indexes always start at 0.

Creating Arrays

- It is possible to declare an array reference and create it in the same statement.

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

- Arrays may be of any type.

```
float[] temperatures = new float[100];
```

```
char[] letters = new char[41];
```

```
long[] units = new long[50];
```

```
double[] sizes = new double[1200];
```

Creating Arrays

- The array size must be a non-negative number.
- It may be a literal value, a constant, or variable.

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

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

Accessing the Elements of an Array

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

- An array is accessed by:
 - the reference name
 - a subscript that identifies which element in the array to access.

`numbers[0] = 20; //pronounced "numbers sub zero"`

Inputting and Outputting Array Elements

- Array elements can be treated as any other variable.
- They are simply accessed by the same name and a subscript.
- See example: [ArrayDemo1.java](#)
- Array subscripts can be accessed using variables (such as for loop counters).
- See example: [ArrayDemo2.java](#)

Bounds Checking

- Array indexes always start at zero and continue to (array length - 1).

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

- This array would have indexes 0 through 9.
- See example: [InvalidSubscript.java](#)
- In `for` loops, it is typical to use *i*, *j*, and *k* as counting variables.
 - It might help to think of *i* as representing the word *index*.

Off-by-One Errors

- It is very easy to be off-by-one when accessing arrays.

```
// This code has an off-by-one error.  
int[] numbers = new int[100];  
for (int i = 1; i <= 100; i++)  
    numbers[i] = 99;
```

- Here, the equal sign allows the loop to continue on to index 100, where 99 is the last index in the array.
- This code would throw an `ArrayIndexOutOfBoundsException`.

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](#)

Alternate Array Declaration

- Previously we showed arrays being declared:

```
int[] numbers;
```

- However, the brackets can also go here:

```
int numbers[];
```

- These are equivalent but the first style is typical.

- Multiple arrays can be declared on the same line.

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

- With the alternate notation each variable must have brackets.

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

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

Processing Array Contents

- Processing data in an array is the same as any other variable.

```
grossPay = hours[3] * payRate;
```

- Pre and post increment works the same:

```
int[] score = {7, 8, 9, 10, 11};
```

```
++score[2]; // Pre-increment operation
```

```
score[4]++; // Post-increment operation
```

- See example: [PayArray.java](#)

Processing Array Contents

- Array elements can be used in relational operations:

```
if(cost[20] < cost[0])  
{  
    //statements  
}
```

- They can be used as loop conditions:

```
while(value[count] != 0)  
{  
    //statements  
}
```

Array Length

- Arrays are objects and provide a public field named `length` that is a constant that can be tested.

```
double[] temperatures = new double[25];
```

- The length of this array is 25.

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

```
int size = temperatures.length;
```

- The variable `size` will contain 25.

The Enhanced `for` Loop

- Simplified array processing (read only)
- Always goes through all elements
- General format:

```
for (datatype elementVariable : array)  
    statement;
```

The Enhanced `for` Loop

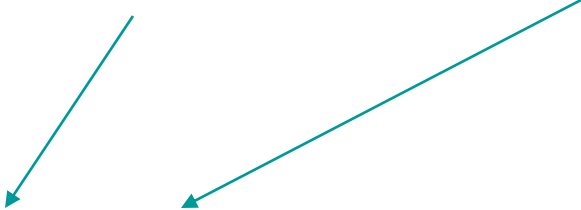
Example:

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

Array Size

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

Index subscripts start at 0 and end at one *less than* the array length.



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

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](#)

Initiate a random 1D-array from 1 to 99

```
static void PhatSinh(int[] a, int n)
{
    Random rd = new Random();
    for (int i = 0; i < n; i++)
    {
        a[i] = rd.Next(100);
    }
}
```

List elements based on predefined condition

Example 1:

```
static void LietKeXXX(int []a, int n)
{
    for (int i = 0; i < n; i++)
        if (a[i] thỏa điều kiện)
            List a[i];
}
```

Example 2:

```
static void LietKeXXX(int []a, int n, int x)
{
    for (int i = 0; i < n; i++)
        if (a[i] thỏa điều kiện so với x)
            List a[i];
}
```

Example 1: List even elements

```
static void LietKeChan(int []a, int n)  
{  
    for (int i = 0; i<n; i++)  
        if (a[i] %2 ==0)  
            System.out.print (a[i] + “\t”);  
}
```

Example 2: List elements greater than x

```
static void LietKeLonHonX(int []a, int n, int x)  
{  
    for (int i = 0; i<n; i++)  
        if (a[i] > x)  
            System.out.print(a[i] + “\t”);  
}
```

Reassigning Array References

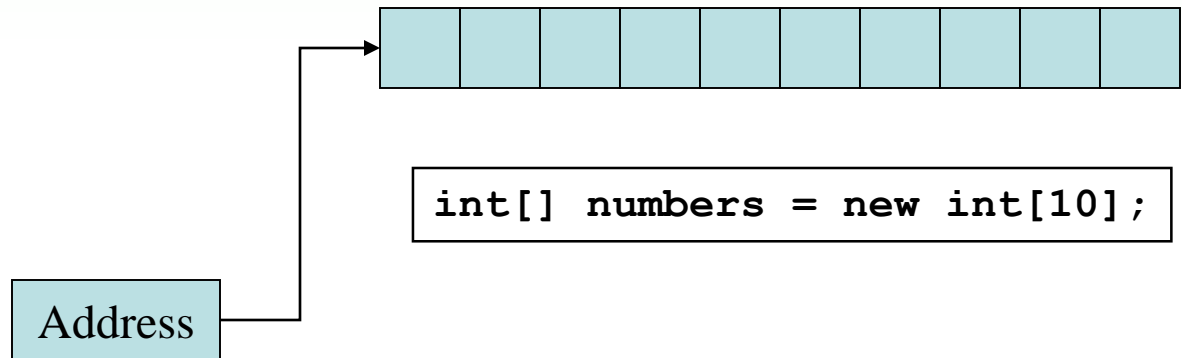
- An array reference can be assigned to another array of the same type.

```
// Create an array referenced by the numbers variable.  
int[] numbers = new int[10];  
// Reassign numbers to a new array.  
numbers = new int[5];
```

- If the first (10 element) array no longer has a reference to it, it will be garbage collected.

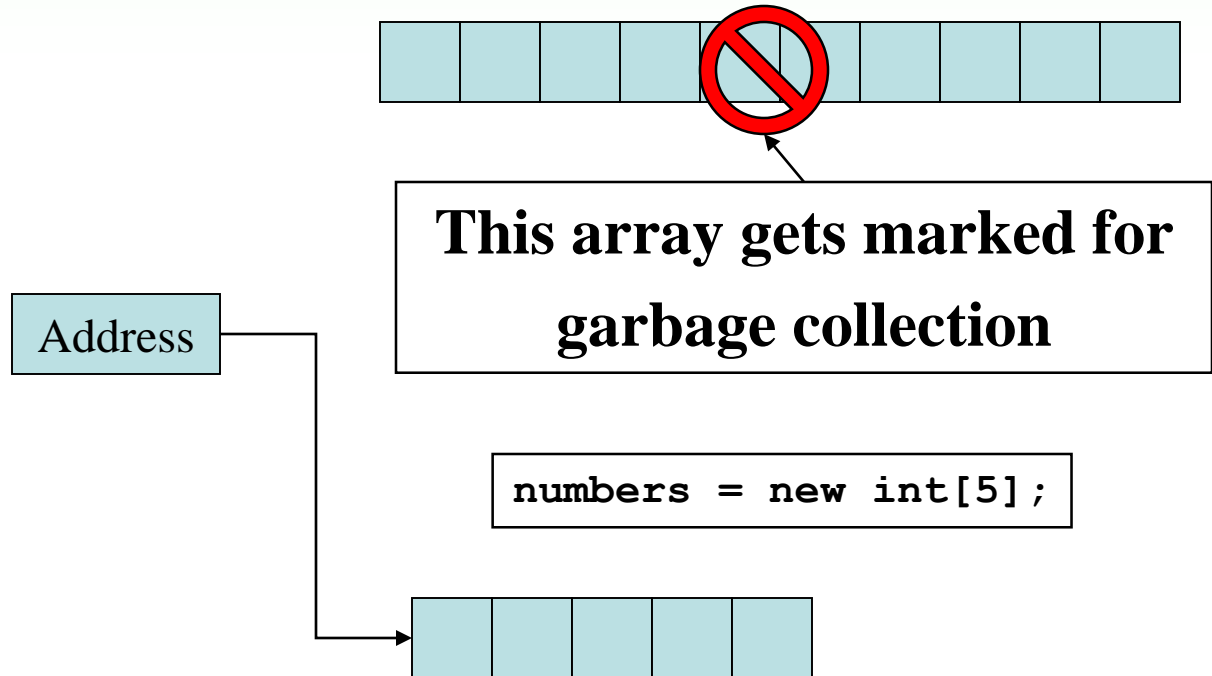
Reassigning Array References

The `numbers` variable holds the address of an `int` array.



Reassigning Array References

The `numbers` variable holds the address of an `int` array.

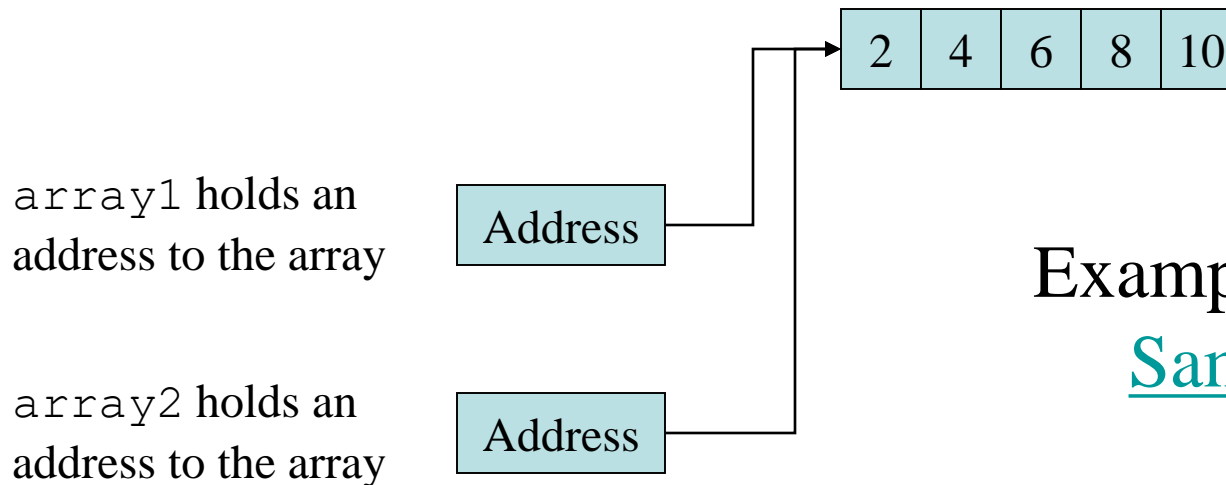


Copying Arrays

- This is *not* the way to copy an array.

```
int[] array1 = { 2, 4, 6, 8, 10 };
```

```
int[] array2 = array1; // This does not copy array1.
```



Example:

[SameArray.java](#)

Copying Arrays

- You cannot copy an array by merely 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];
```

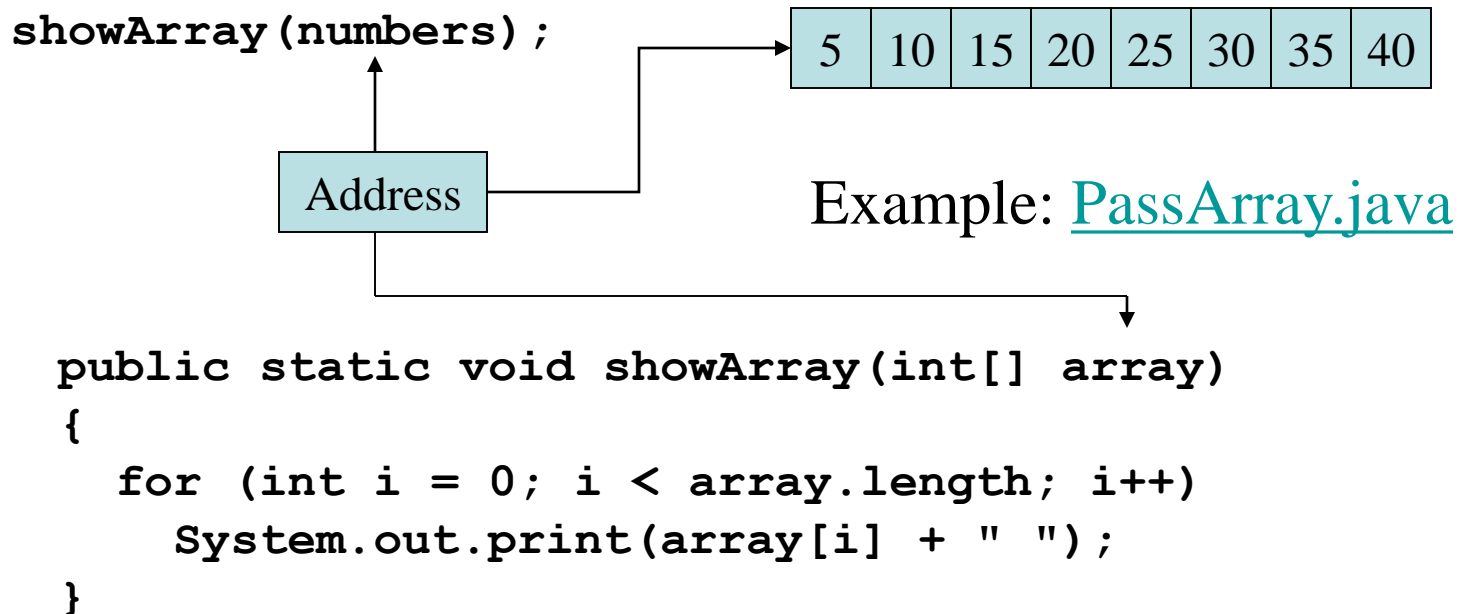
- This code copies each element of `firstArray` to the corresponding element of `secondArray`.

Passing Array Elements to a Method

- When a single element of an array is passed to a method it is handled like any other variable.
- See example: [PassElements.java](#)
- More often you will want to write methods to process array data by passing the entire array, not just one element at a time.

Passing Arrays as Arguments

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



Comparing Arrays

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

```
int[] firstArray = { 5, 10, 15, 20, 25 };  
int[] secondArray = { 5, 10, 15, 20, 25 };  
  
if (firstArray == secondArray) // This is a mistake.  
    System.out.println("The arrays are the same.");  
else  
    System.out.println("The arrays are not the same.");
```

Comparing Arrays: Example

```
int[] firstArray = { 2, 4, 6, 8, 10 };
int[] secondArray = { 2, 4, 6, 8, 10 };
boolean arraysEqual = true;
int i = 0;

// First determine whether the arrays are the same size.
if (firstArray.length != secondArray.length)
    arraysEqual = false;

// Next determine whether the elements contain the same data.
while (arraysEqual && i < firstArray.length)
{
    if (firstArray[i] != secondArray[i])
        arraysEqual = false;
    i++;
}

if (arraysEqual)
    System.out.println("The arrays are equal.");
else
    System.out.println("The arrays are not equal.");
```


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[] array = { 1.2, 2.3, 4.5, 6.7, 8.9 };  
    return array;  
}
```

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

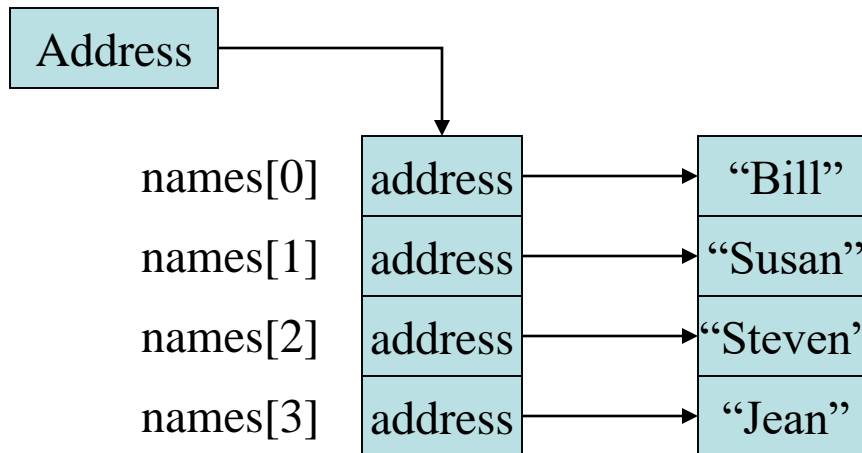
String Arrays

- Arrays are not limited to primitive data.
- An array of `String` objects can be created:

```
String[] names = { "Bill", "Susan", "Steven", "Jean" };
```

The `names` variable holds
the address to the array.

A `String` array is an array
of references to `String` objects.



Example:

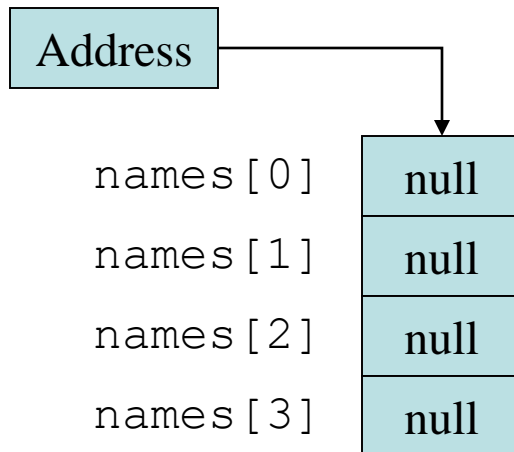
[MonthDays.java](#)

String Arrays

- If an initialization list is not provided, the `new` keyword must be used to create the array:

```
String[] names = new String[4];
```

The `names` variable holds
the address to the array.

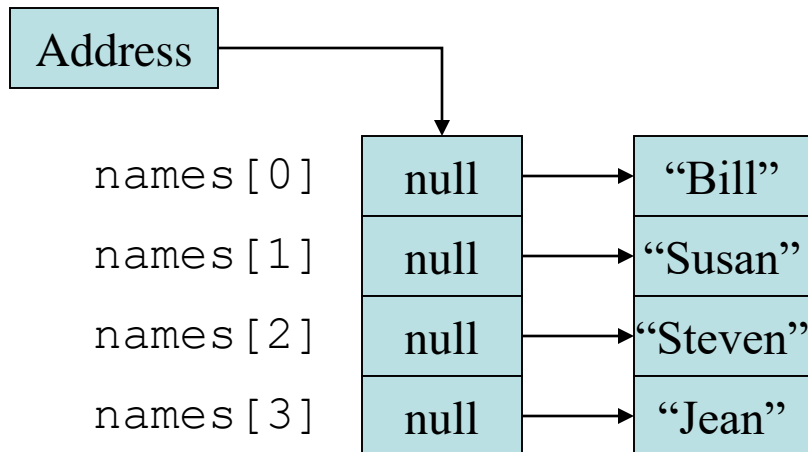


String Arrays

- When an array is created in this manner, each element of the array must be initialized.

The `names` variable holds the address to the array.

```
names[0] = "Bill";  
names[1] = "Susan";  
names[2] = "Steven";  
names[3] = "Jean";
```



Calling `String` Methods On Array Elements

- `String` objects have several methods, including:
 - `toUpperCase`
 - `compareTo`
 - `equals`
 - `charAt`
- Each element of a `String` array is a `String` object.
- Methods can be used by using the array name and index as before.

```
System.out.println(names[0].toUpperCase());  
char letter = names[3].charAt(0);
```

The length Field & The length Method

- Arrays have a **final field** named `length`.
- String objects have a **method** named `length`.
- To display the length of each string held in a `String` array:

```
for (int i = 0; i < names.length; i++)  
    System.out.println(names[i].length());
```

- An array's length is a **field**
 - You do not write a set of parentheses after its name.
- A `String`'s length is a **method**
 - You do write the parentheses after the name of the `String` class's `length` method.

Element counting

Sample 1:

```
static int DemXXX(int []a)  
{  
    int d = 0;  
    for (int i = 0; i < a.length; i++)  
        if (a[i] satisfy the conditions)  
            d++;  
    return d;  
}
```

Sample 2:

```
static int DemXXX(int []a, int x)  
{  
    int d = 0;  
    for (int i = 0; i < a.length; i++)  
        if (a[i] satisfy with x)  
            d++;  
  
    return d;  
}
```

Example 1: Count prime element

```
bool isPrime(int k){  
  
}
```

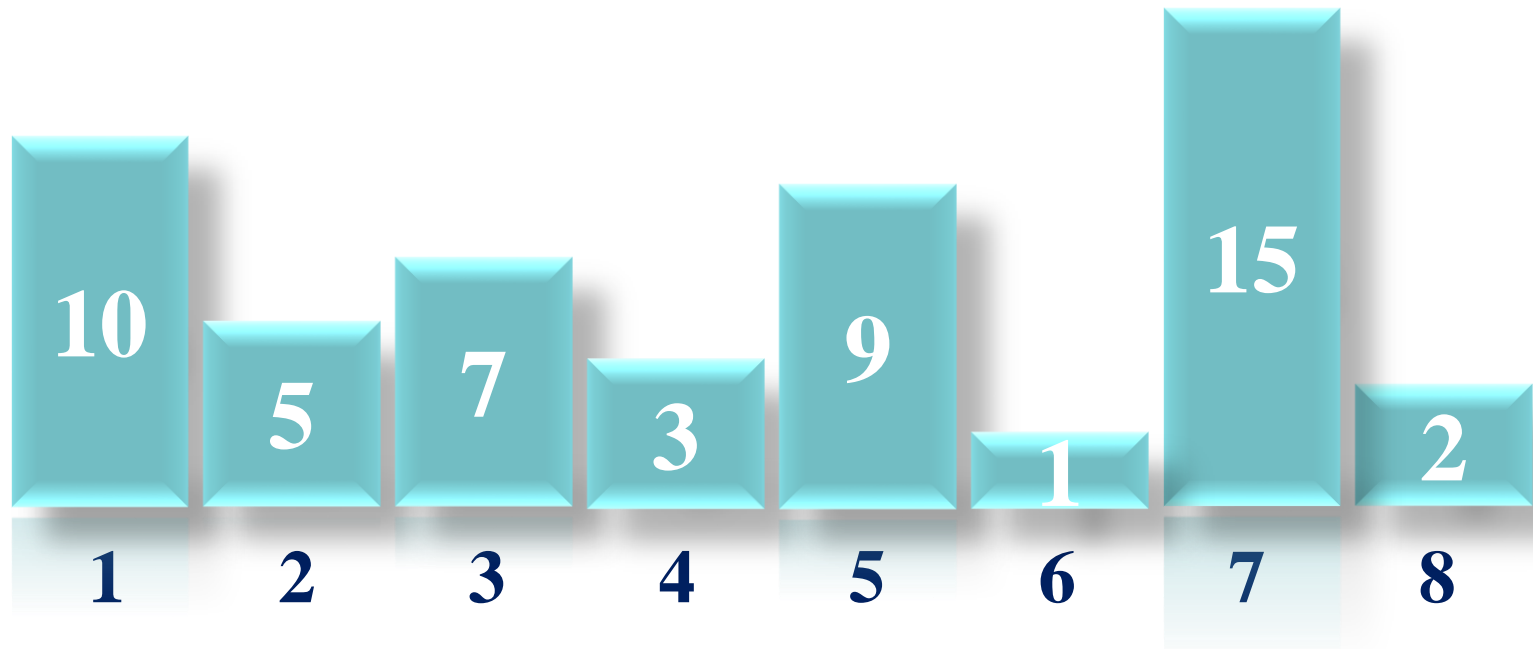
```
int countPrime(int []a){  
    int d = 0;  
    for (int i = 0; i<a.length; i++)  
    {  
        if (LaSNT(a[i]) ==true)  
        {  
            d++;  
        }  
    }  
    return d;  
}
```

Example2: Count element less than x

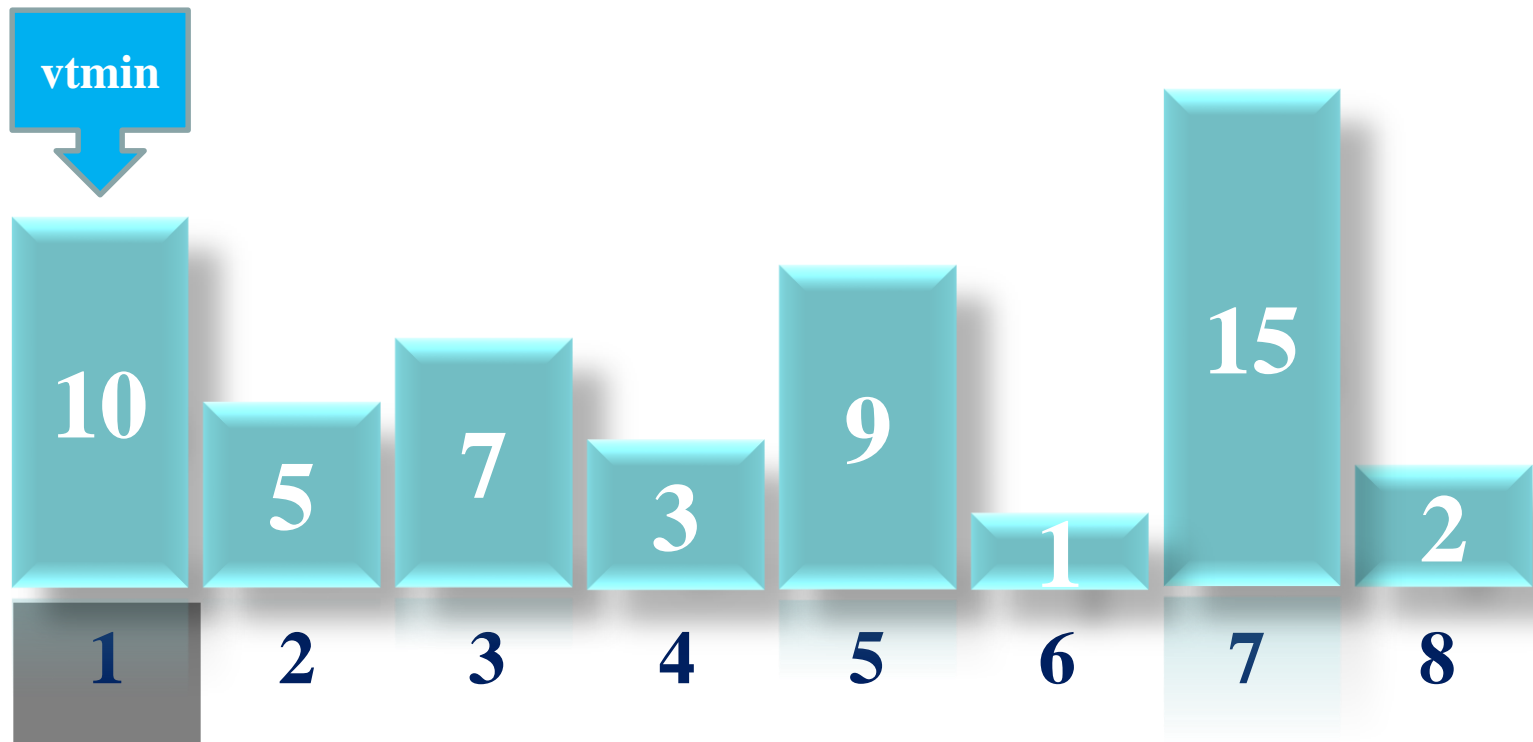
```
int DemNhoHonX(int []a, int x){  
    int d = 0;  
    for (int i = 0; i < a.length; i++)  
        if (a[i] < x)  
            d++;  
    return d;  
}
```

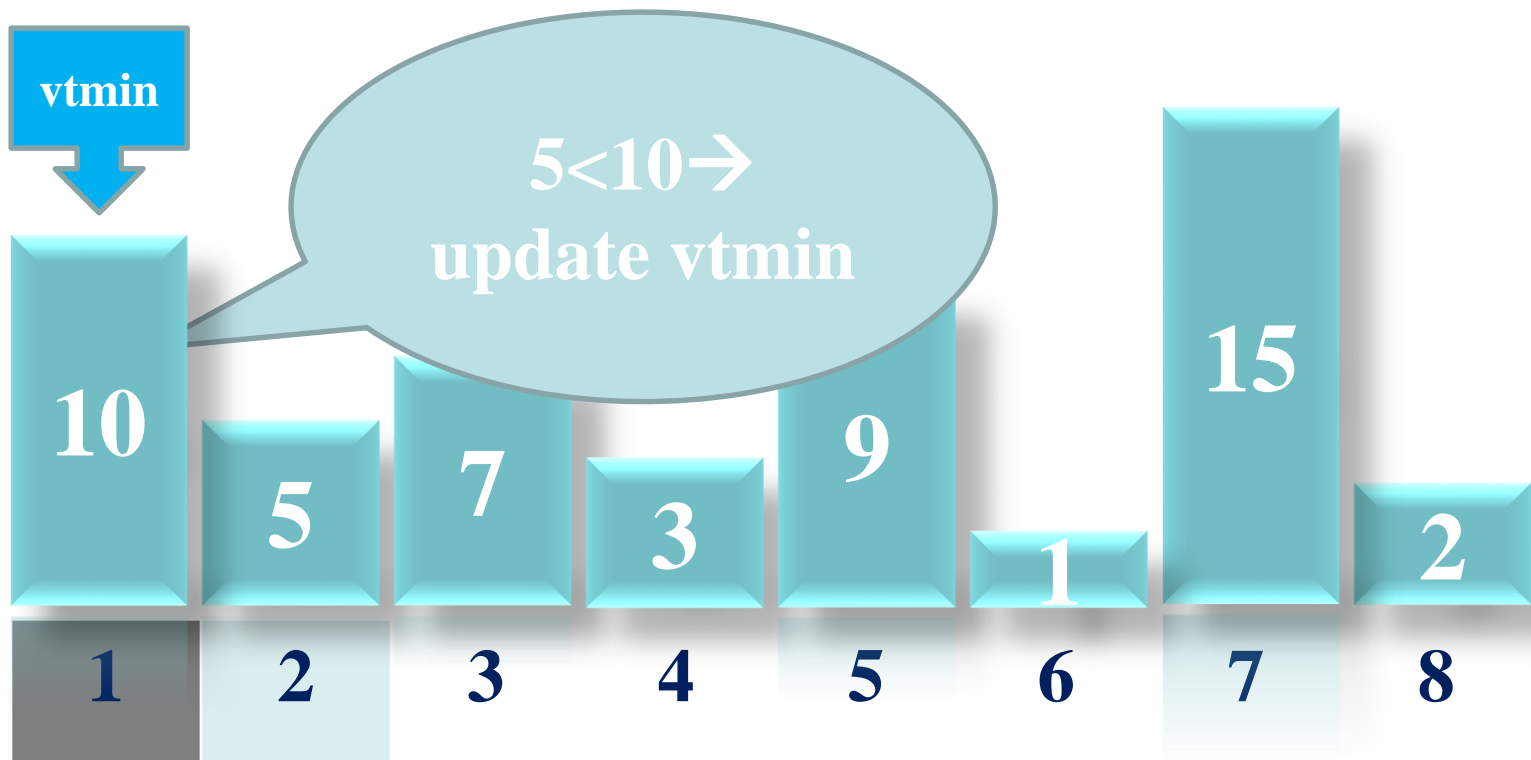
Find the position of the smallest element?

Suppose find the smallest position ?



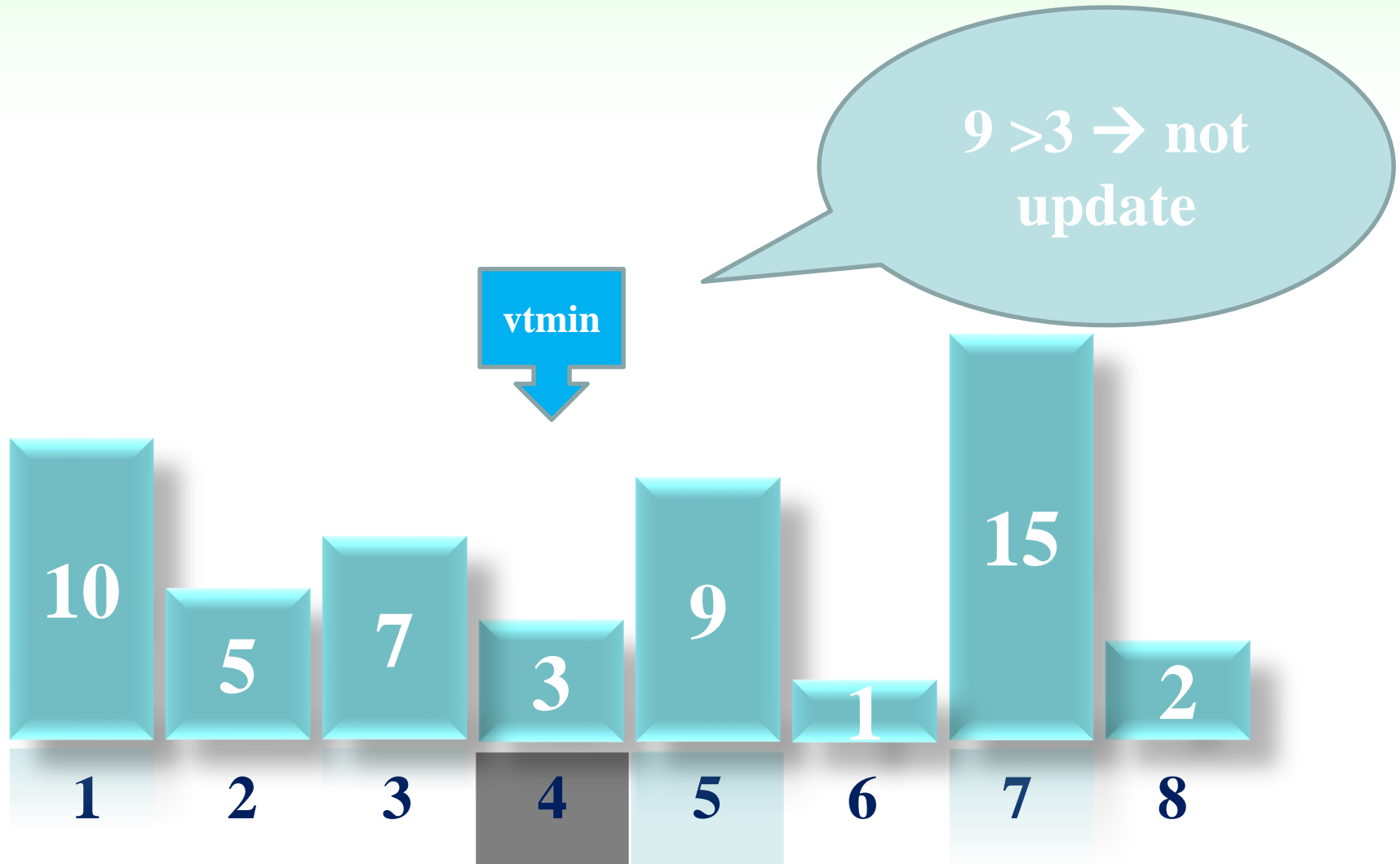
Step 1: Suppose the smallest has position 1 (v_{tmin}), it's value is 10









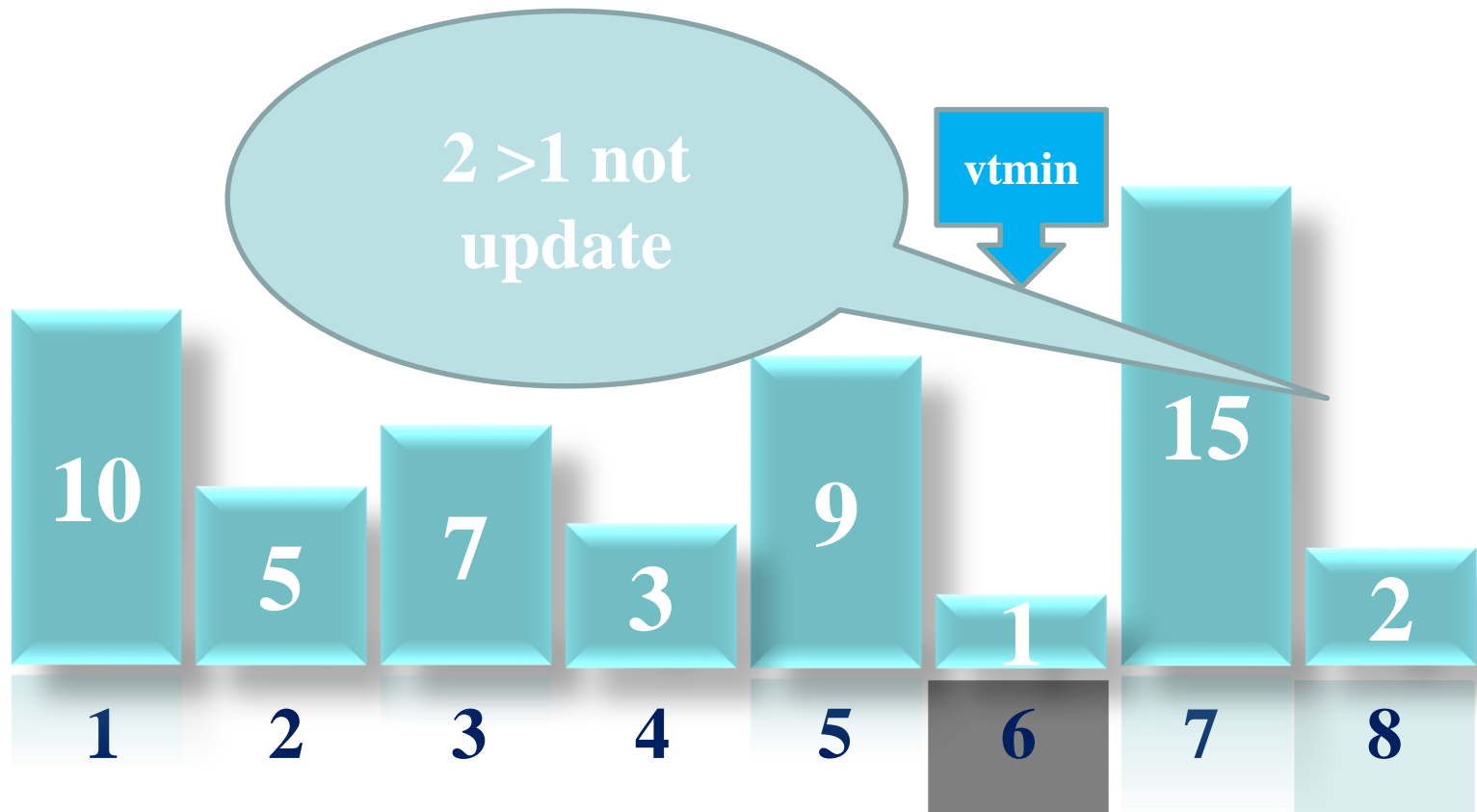


$1 < 3 \rightarrow$
update vmin

vmin







CODE

```
static int TimVTMin(int []a, int n)  
{  
    int vtmin = 0;  
    for (int i = 1; i < n; i++)  
    {  
        if (a[i] < a[vtmin])  
            vtmin = i;  
    }  
    return vtmin;  
}
```


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];
}
```

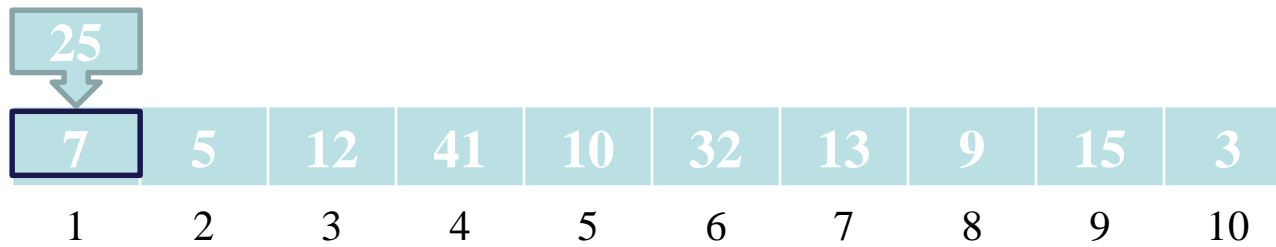
Find element X

- **Idea**

Compare x inturn with with 1st element, 2nd,... of array a until the desired element or to the end of the array without find.

- Find $x = 10$

- Find $x = 25$



CODE

(If x is not in the array, return -1)

```
static int TimVTX(int []a, int x){  
    for (int i = 0; i < a.length; i++){  
        if (a[i] == x)  
            return i;  
    }  
    return -1;  
}
```

Check if the array is satisfied a certain condition

- **Case1**: check the existence of an element that satisfies a certain condition → find the element to conclude.
- **Case2**: check all if all elements that satisfy a certain condition → find the element not satisfies to conclude.

Sample Case 1:

```
static bool KiemTraTonTaiXXX(int []a, int n){  
    for (int i = 0; i<n; i++)  
        if (a[i] satisfy the conditions)  
            return true;  
    return false;  
}
```

Sample Case 2:

```
static bool KiemTraXXX(int []a, int n){  
    for (int i = 0; i<n; i++)  
        if (a[i] not satisfy the conditions)  
            return false;  
    return true;  
}
```

Example1: Check the existence of odd number?

```
static bool KiemTraTonTaiLe(int []a){  
    for (int i = 0; i < a.length; i++)  
    {  
        if (a[i] % 2 != 0)  
            return true;  
    }  
    return false;  
}
```

Example2: Check are all element negative? (*true/false*)

```
static bool KiemTraToanAm(int []a){  
    for (int i = 0; i<a.length; i++)  
    {  
        if (a[i] >= 0)  
            return false;  
    }  
    return true;  
}
```

Calculate conditional Sum, Average

Sample SUM:

```
static int TongXXX(int []a){  
    int s = 0;  
    for (int i = 0; i<a.length; i++){  
        if (a[i] satisfy the conditions)  
            s += a[i];  
    }  
    return s;  
}
```


Sample Average:

```
static float TrungBinhXXX(int []a){  
    int s = 0;  
    int d = 0;  
    for (int i = 0; i < a.length; i++){  
        if (a[i] satisfy the conditions){  
            s += a[i];  
            d++;  
        }  
    }  
    if (d == 0)  
        return 0;  
    return (float) s / d;  
}
```

Example1: Calculate sum of all odd value elements

```
static int TongLe(int []a, int n){  
    int s = 0;  
    for (int i = 0; i<n; i++)  
    {  
        if (a[i] %2!=0)  
            s += a[i];  
    }  
    return s;  
}
```

Example2: Calculate average of all negative value elements

```
static float TrungBinhAm(int []a, int n){  
    long s = 0;  
    int d = 0;  
    for (int i = 0; i<n; i++){  
        if (a[i] < 0){  
            s += a[i];  
            d++;  
        }  
    }  
    if (d == 0)  
        return 0;  
    return (float)s / d;  
}
```

Sort

Sample sorting algorithm:

```
static void Sort(int [ ]a, int n){  
    for (int i = 0; i < n-1; i ++ ) {  
        for(int j = i+1; j < n; j ++ )  
            if (a[i] > a[j])  
                swap(a[i],a[j]);  
    }  
}
```

Insert element

- Given an array:

12	5	7	9	21	38
0	1	2	3	4	5

- Insert **111** into position **3** of the above array

111

12	5	7	9	21	38	
0	1	2	3	4	5	

Insert element

- Write a function to insert an element have value x into position k in an array has size n :

```
public static int[] themvt(int[] a, int x, int k) {  
    int[] e = new int[a.length+1];  
    int i;  
    for(i=0; i<k; i++)  
        e[i]=a[i];  
    e[k]=x;  
    for(i=k+1; i<e.length; i++)  
        e[i]=a[i-1];  
    return e;  
}
```

Applying Assignment

Write a function to insert an element has value x after the smallest element in an array (suppose there is no duplicated element in the array)

Delete element

- Given an array :

12	5	7	9	21	38
0	1	2	3	4	5

- Delete element at position 3 in the above array.

12	5	7	9	21	38
0	1	2	3	4	5

Delete element

- Write a function to delete an element at position k in an array:

static int []XoaTaiVTk(int []a, int k);

Applying Assignment

Write a function to delete an element x (*if exist*) in an array size n (suppose there is no duplicated element in the array).

60' test

- 1. Write a function to calculate the sum of even element in an array.

static int TinhTong(int []a)

- 2. Write a function to count how many element equal to the sum of all other elements

static int demTong(int []a)

Example: array a[]={ 1,5,4,12,2}. Return value is 1

- 3. Write a function to find the most frequently appear element in an array.
- static int dem_lan_xuat_hien(int []a)*

Example : array a[]={ 1,2,1,3,4,3,2,3}. Return value is 3

Assignment

Write a program input an integer array and functions to do following tasks:

1. Delete duplicated elements, keep only one.
2. Sort in incremental order then insert an element X such that the order is kept intact.
3. Find and delete an element X in an array.

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];
```



- The two sets of brackets in the data type indicate that the scores variable will reference a two-dimensional array.
- Notice that each size declarator is enclosed in its own set of brackets.

Accessing Two-Dimensional Array Elements

- When processing the data in a two-dimensional array, each element has two subscripts:
 - one for its row and
 - another for its column.

Accessing Two-Dimensional Array Elements

The `scores` variable holds the address of a 2D array of `doubles`.

		column 0	column 1	column 2	column 3
Address		scores[0][0]	scores[0][1]	scores[0][2]	scores[0][3]
	row 0	scores[0][0]	scores[0][1]	scores[0][2]	scores[0][3]
	row 1	scores[1][0]	scores[1][1]	scores[1][2]	scores[1][3]
	row 2	scores[2][0]	scores[2][1]	scores[2][2]	scores[2][3]

Accessing Two-Dimensional Array Elements

Accessing one of the elements in a two-dimensional array requires the use of both subscripts.

The `scores` variable holds the address of a 2D array of `doubles`.

```
scores[2][1] = 95;
```

Address		column 0	column 1	column 2	column 3
row 0		0	0	0	0
row 1		0	0	0	0
row 2		0	95	0	0

Accessing Two-Dimensional Array Elements

- Programs that process two-dimensional arrays can do so with nested loops.
- To fill the scores array:

```
for (int row = 0; row < 3; row++)  
{  
    for (int col = 0; col < 4; col++)  
    {  
        System.out.print("Enter a score: ");  
        scores[row][col] = keyboard.nextDouble();  
    }  
}
```

Number of rows, not the largest subscript

Number of columns, not the largest subscript

keyboard references a Scanner object

Accessing Two-Dimensional Array Elements

- To print out the `scores` array:

```
for (int row = 0; row < 3; row++)  
{  
    for (int col = 0; col < 4; col++)  
    {  
        System.out.println(scores[row][col]);  
    }  
}
```

- See example: [CorpSales.java](#)

Initializing a Two-Dimensional Array

- Initializing a two-dimensional array requires enclosing each row's initialization list in its own set of braces.

```
int[][] numbers = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
```

- Java automatically creates the array and fills its elements with the initialization values.
 - row 0 {1, 2, 3}
 - row 1 {4, 5, 6}
 - row 2 {7, 8, 9}
- Declares an array with three rows and three columns.

Initializing a Two-Dimensional Array

```
int[][] numbers = {{1, 2, 3},  
                  {4, 5, 6},  
                  {7, 8, 9}};
```

The `numbers` variable holds the address of a 2D array of `int` values.

produces:



Address			
	column 0	column 1	column 2
row 0	1	2	3
row 1	4	5	6
row 2	7	8	9

The length Field

- Two-dimensional arrays are arrays of one-dimensional arrays.
- The length field of the array gives the number of rows in the array.
- Each row has a length constant tells how many columns is in that row.
- Each row can have a different number of columns.

The length Field

- To access the length fields of the array:

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

```
for (int row = 0; row < numbers.length; row++)  
{  
    for (int col = 0; col < numbers[row].length; col++)  
        System.out.println(numbers[row][col]);  
}
```

Number of rows

Number of columns in this row.

- See example: [Lengths.java](#)

The array can have variable length rows.

Summing The Elements of a Two-Dimensional Array

```
int[][] numbers = { { 1, 2, 3, 4 },  
                    {5, 6, 7, 8},  
                    {9, 10, 11, 12} };  
  
int total;  
total = 0;  
for (int row = 0; row < numbers.length; row++)  
{  
    for (int col = 0; col < numbers[row].length; col++)  
        total += numbers[row][col];  
}  
  
System.out.println("The total is " + total);
```


Summing The Rows of a Two-Dimensional Array

```
int[][] numbers = {{ 1, 2, 3, 4},  
                   {5, 6, 7, 8},  
                   {9, 10, 11, 12}};  
  
int total;  
  
for (int row = 0; row < numbers.length; row++)  
{  
    total = 0;  
    for (int col = 0; col < numbers[row].length; col++)  
        total += numbers[row][col];  
    System.out.println("Total of row "  
                        + row + " is " + total);  
}
```

Summing The Columns of a Two-Dimensional Array

```
int[][] numbers = {{1, 2, 3, 4},  
                  {5, 6, 7, 8},  
                  {9, 10, 11, 12}};  
  
int total;  
  
for (int col = 0; col < numbers[0].length; col++)  
{  
    total = 0;  
    for (int row = 0; row < numbers.length; row++)  
        total += numbers[row][col];  
    System.out.println("Total of column "  
                       + col + " is " + total);  
}
```

Passing and Returning Two-Dimensional Array References

- There is no difference between passing a single or two-dimensional array as an argument to a method.
- The method must accept a two-dimensional array as a parameter.
- See example: [Pass2Darray.java](#)

Ragged Arrays

- When the rows of a two-dimensional array are of different lengths, the array is known as a *ragged array*.
- You can create a ragged array by creating a two-dimensional array with a specific number of rows, but no columns.

```
int [][] ragged = new int [4][];
```

- Then create the individual rows.

```
ragged[0] = new int [3];  
ragged[1] = new int [4];  
ragged[2] = new int [5];  
ragged[3] = new int [6];
```

Q&A

