Array & Linked Lists

Lecture 1

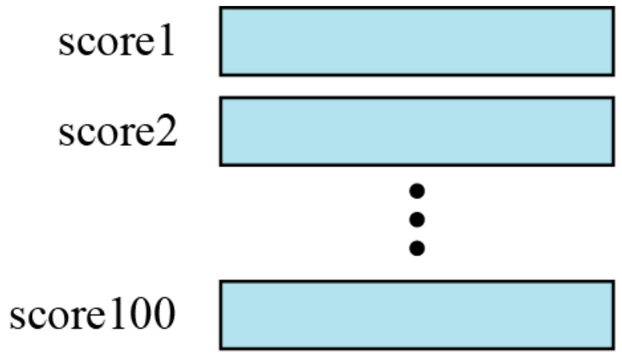
Chapter Scope

- Lists
  - Arrays
  - Linked Lists
    - Singly Linked List
    - Doubly Linked List
  - Linked Lists Analysis (Complexity Analysis)
    - With Singly Linked List
    - With Doubly Linked List

Arrays

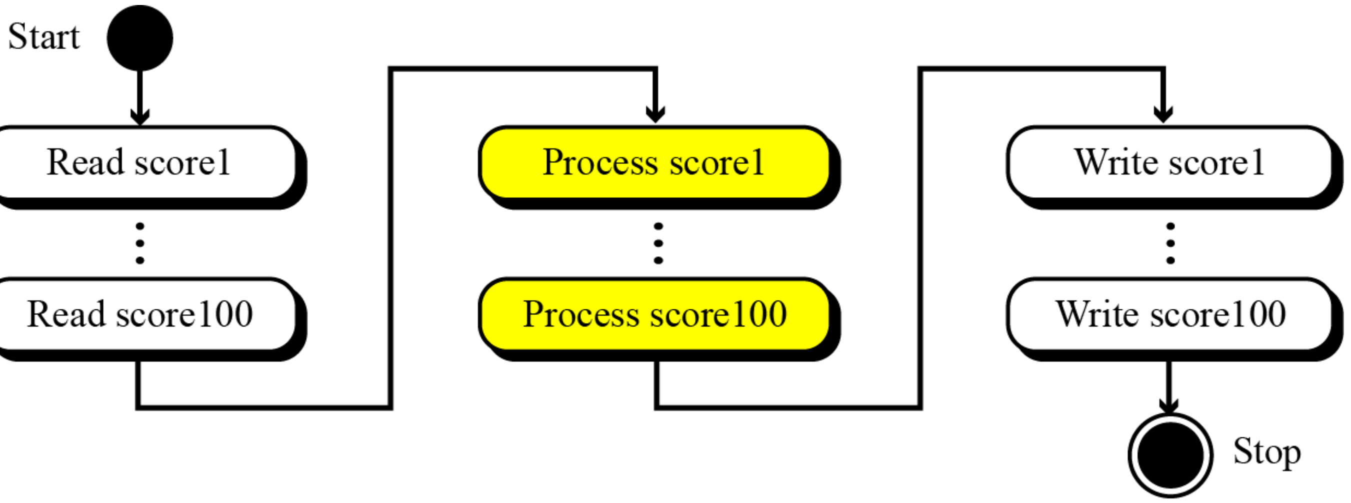
Arrays

- Imagine that we have 100 scores. We need to read them, process them and print them.
- We must also keep these 100 scores in memory for the duration of the program.
- We can define a hundred variables, each with a different name, as shown below



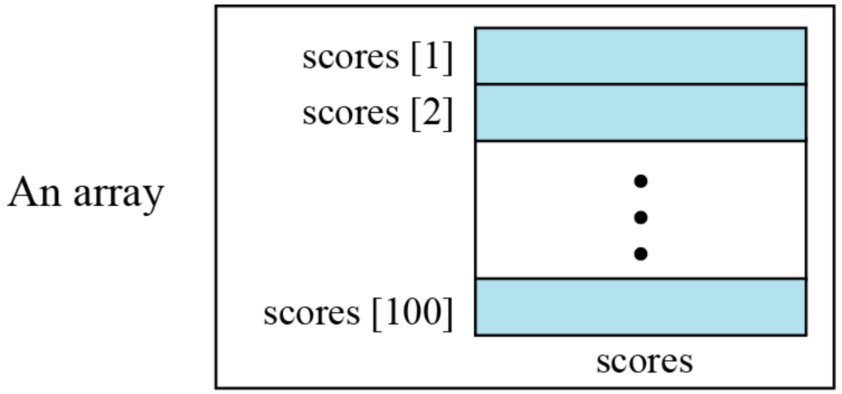
Arrays (cont…)

- But having 100 different names creates other problems. We need 100 references to read them, 100 references to process them and 100 references to write them. The following Figure shows a diagram that illustrates this problem.



Arrays (cont…)

- An array is a sequenced collection of elements, normally of the same data type, although some programming languages accept arrays in which elements are of different types. We can refer to the elements in the array as the first element, the second element and so forth, until we get to the last element.



Arrays (Declaration)

- **Declared** using [ ] operator

```
1- datatype[] arrayRefVar;
```

Example:

```
double[] myList;
```

OR

```
2- datatype arrayRefVar[];
```
// This style is allowed

Example:

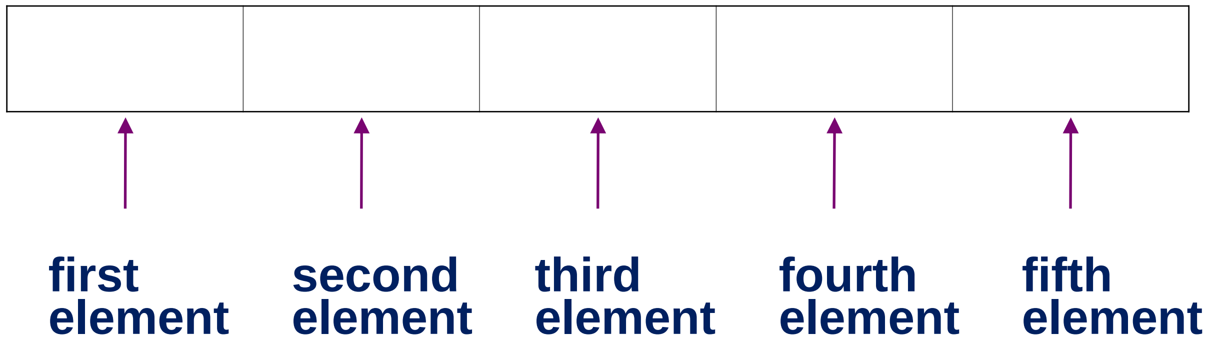
```
double myList[];
```

Arrays (Storage in Memory)

- In the definition:

```
int [ ] tests;

tests = new int[SIZE];    // SIZE is 5
```

allocates the following memory



Arrays (Terminology)

- In the definition:

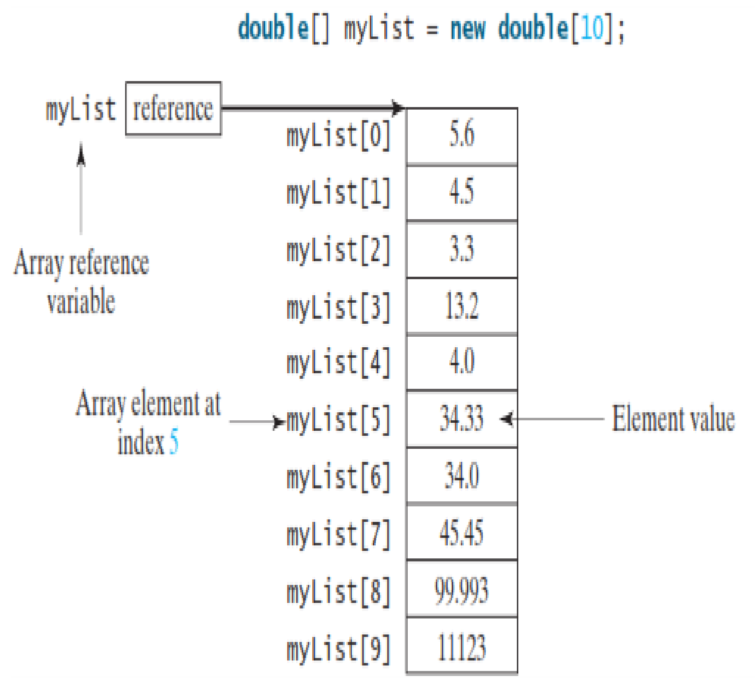
```
int [ ] tests;
tests = new int[ISIZE];
```

- `int` is the data type of the array elements
- `tests` is the name of the array
- `ISIZE`, in `[ISIZE]`, is the size declarator. It shows the number of elements in the array.
- The size of an array is the number of bytes allocated for it

(number of elements) * (bytes needed for each element)

Arrays (Terminology) (cont…)

Array Terminology Examples



The array myList has ten elements of double type and int indices from 0 to 9.

The Length of an Array

Once an array is created, its size is fixed. It cannot be changed. You can find its size using

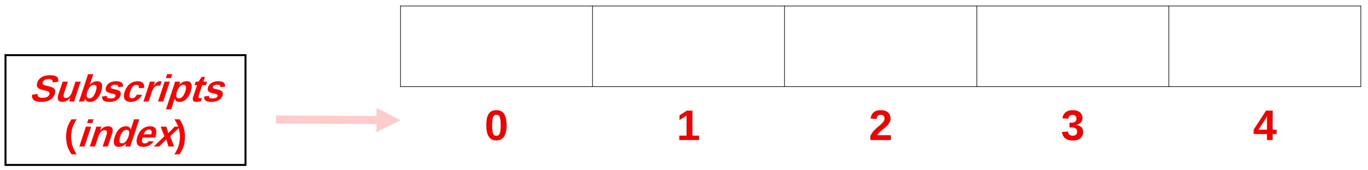
arrayRefVar.length

For example,

myList.length //returns 10

Arrays (Accessing Array Elements)

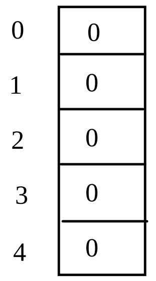
- Each array element has a *subscript (index)*, used to access the element.
- Subscripts *(index)* start at 0



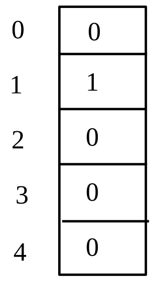
Arrays (Example)

```
public class Test {
  public static void main(String[] args)
{
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
        values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created



After the first iteration

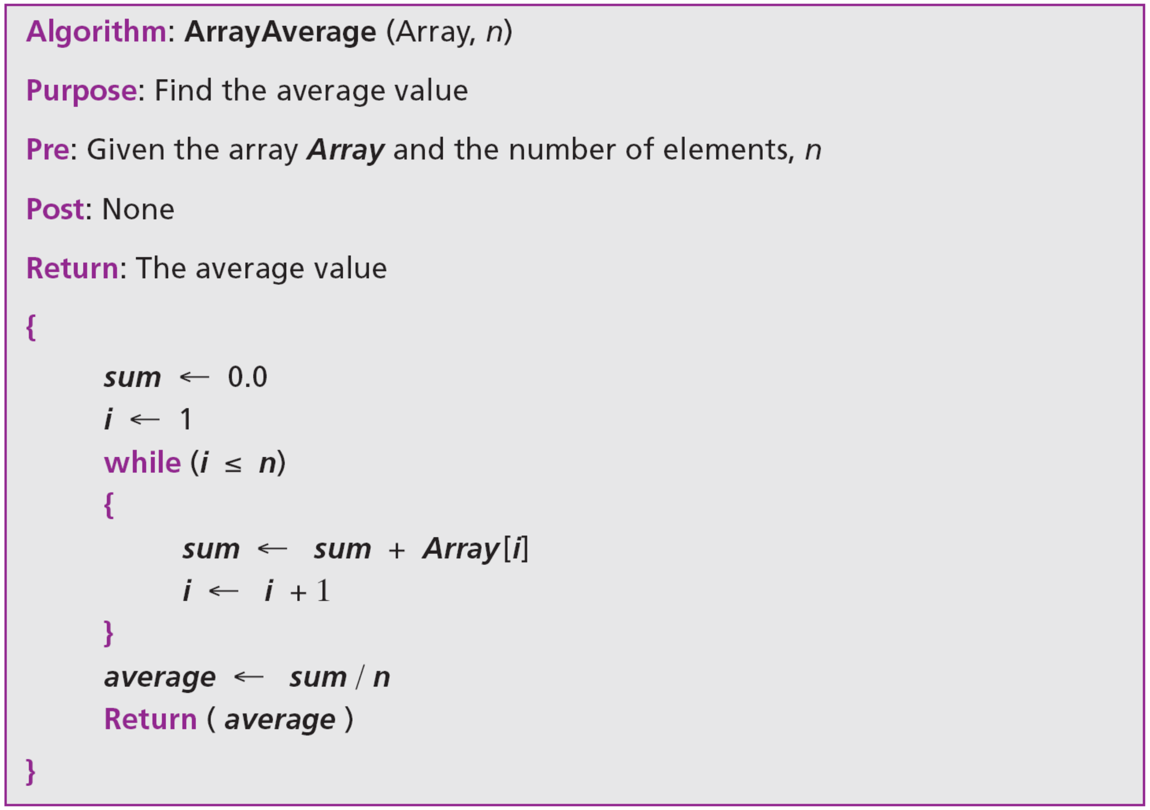


Operations on Array

- Although we can apply conventional operations defined for each element of an array, there are some operations that we can define on an array as a data structure.
- The common operations on arrays as structures are **searching**, **insertion**, **deletion**, **retrieval** and **traversal**.
- Although searching, retrieval and traversal of an array is an easy job, insertion and deletion is time consuming.
- The elements need to be shifted down before insertion and shifted up after deletion.
- An array is a suitable structure when a small number of insertions and deletions are required, but a lot of searching and retrieval is needed.

Operations on Array (cont…)

- The following algorithm gives an example of finding the average of elements in array whose elements are reals.



Algorithm: **ArrayAverage** (Array, *n*)

Purpose: Find the average value

Pre: Given the array ***Array*** and the number of elements, *n*

Post: None

Return: The average value

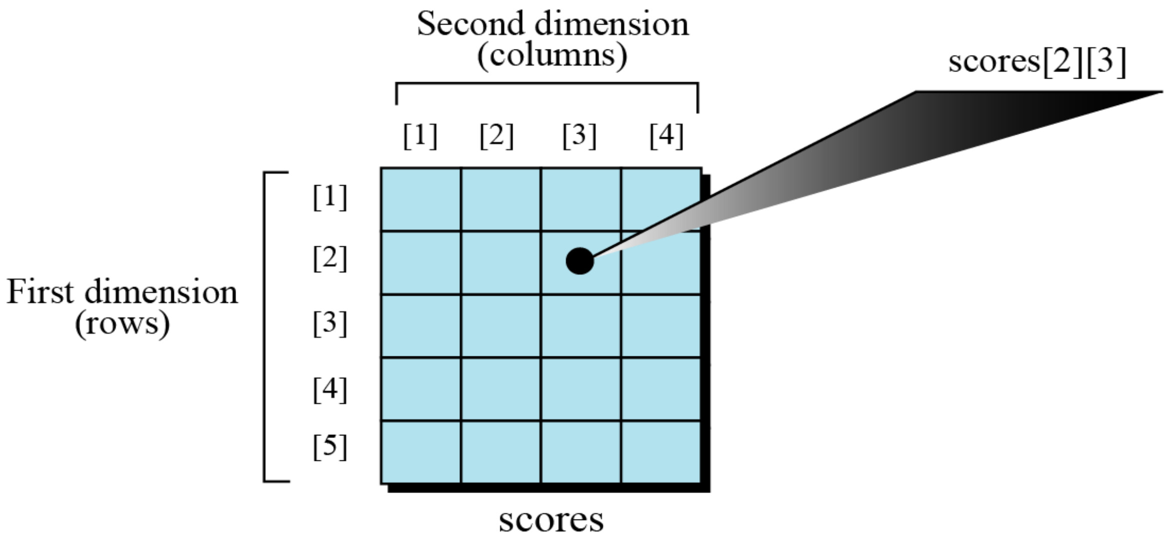
```
{
    sum ← 0.0
    i ← 1
    while (i ≤ n)
    {
        sum ← sum + Array[i]
        i ← i + 1
    }
    average ← sum / n
    Return ( average )
}
```

Arrays: The Pros & Cons

- **Pros:**
  - Fast element access
- **Cons:**
  - While many applications require resizing, static arrays
  - are impossible to resize
  - Required size is not always immediately available

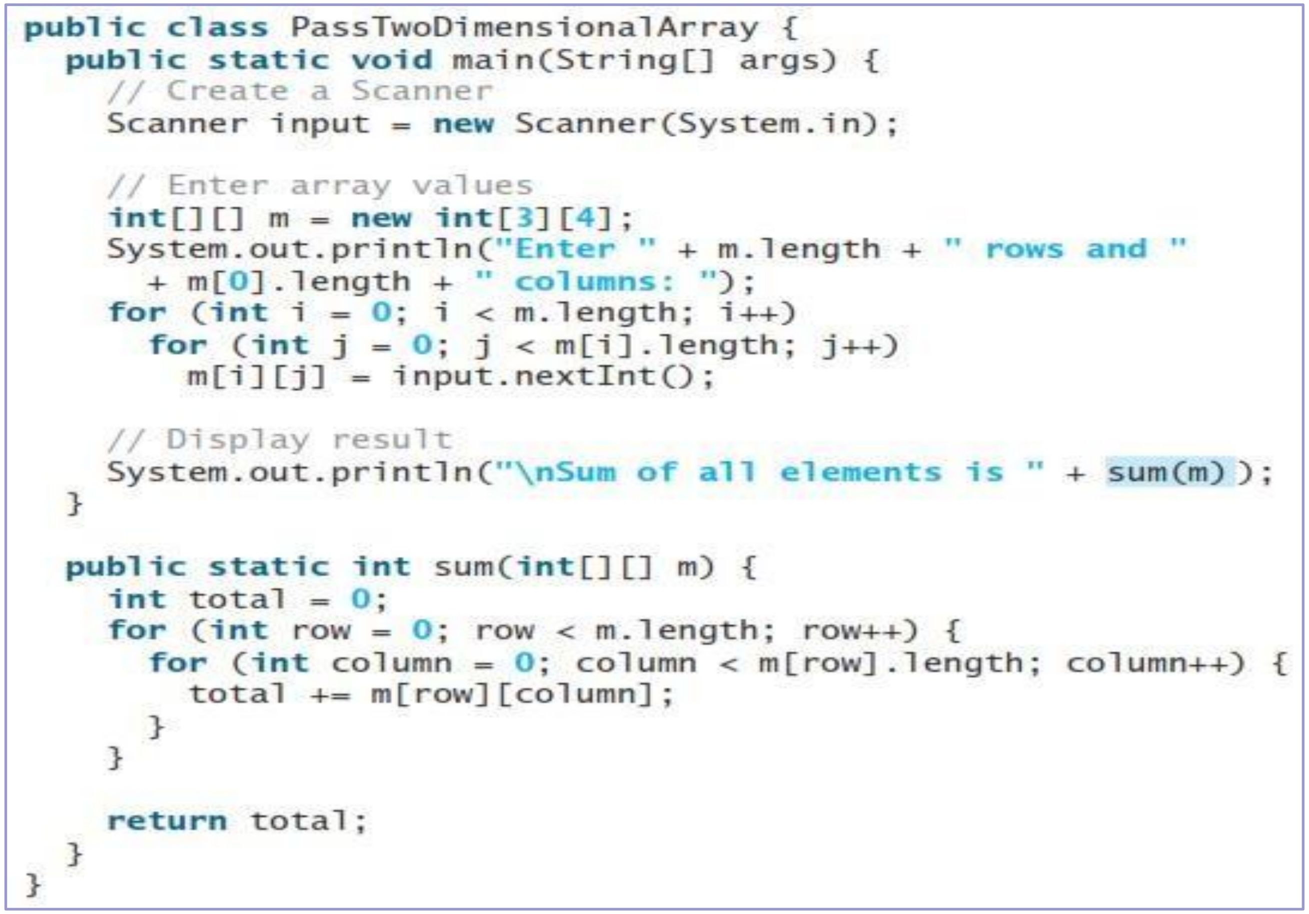
Multi-Dimensional Arrays

- The arrays discussed so far are known as **one-dimensional** arrays because the data is organized linearly in only one direction.
- Many applications require that data be stored in more than one dimension. The Figure below shows a table, which is commonly called a **two-dimensional** array.



Multi-Dimensional Arrays (cont…)

-



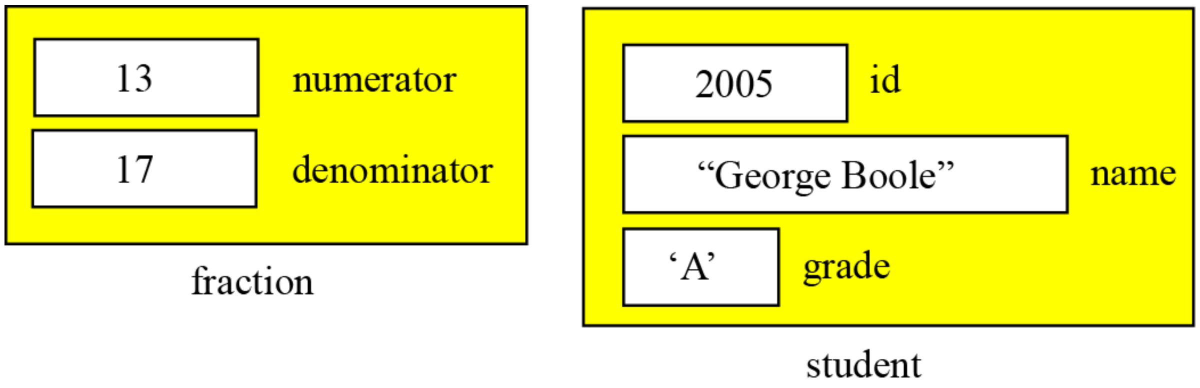
```
public class PassTwoDimensionalArray {
  public static void main(String[] args) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);

    // Enter array values
    int[][] m = new int[3][4];
    System.out.println("Enter " + m.length + " rows and "
      + m[0].length + " columns: ");
    for (int i = 0; i < m.length; i++)
      for (int j = 0; j < m[i].length; j++)
        m[i][j] = input.nextInt();

    // Display result
    System.out.println("\nSum of all elements is " + sum(m));
  }

  public static int sum(int[][] m) {
    int total = 0;
    for (int row = 0; row < m.length; row++) {
      for (int column = 0; column < m[row].length; column++) {
        total += m[row][column];
      }
    }

    return total;
  }
}
```

Records

- A record is a collection of related elements, possibly of different types, having a single name.
- Each element in a record is called a field.
- A field is the smallest element of named data that has meaning.
- A field has a type and exists in memory.
- Fields can be assigned values, which in turn can be accessed for selection or manipulation.
- A field differs from a variable primarily in that it is part of a record.

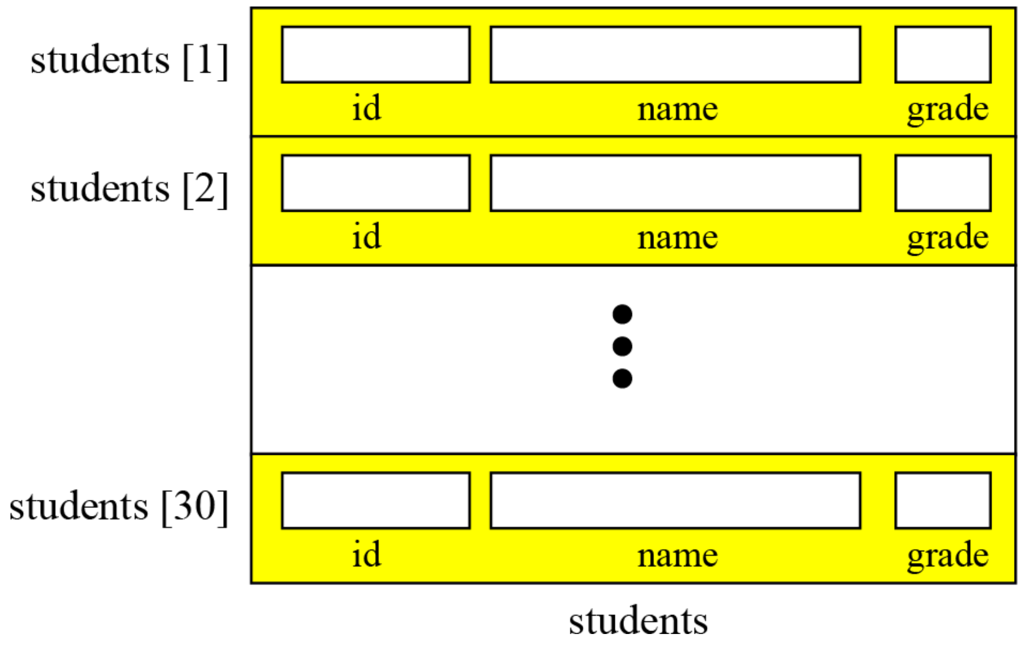
Records (cont…)

- Two examples of records: The first example, fraction, has two fields, both of which are integers. The second example, student, has three fields made up of three different types.



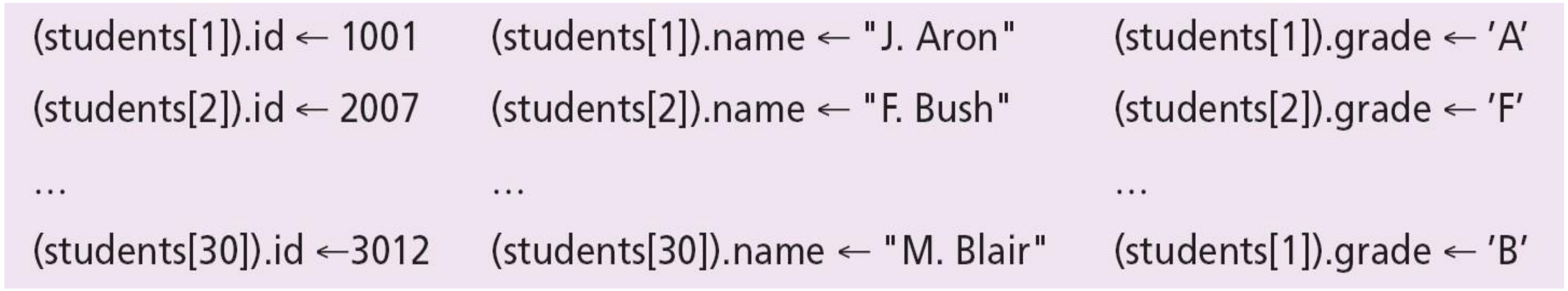
Array of Records

- If we need to define a combination of elements and at the same time some attributes of each element, we can use an array of records.
- For example, in a class of 30 students, we can have an array of 30 records, each record representing a student.



Array of Records (cont…)

- <u>Example:</u>
- The following shows how we access the fields of each record in the students array to store values in them.



```
(students[1]).id ← 1001      (students[1]).name ← "J. Aron"      (students[1]).grade ← 'A'
(students[2]).id ← 2007      (students[2]).name ← "F. Bush"      (students[2]).grade ← 'F'
…                            …                                   …
(students[30]).id ←3012      (students[30]).name ← "M. Blair"    (students[1]).grade ← 'B'
```

Array of Records (cont…)

- However, we normally use a loop to read data into an array of records. Algorithm 11.2 shows part of the pseudocode for this process.

Algorithm 11.2 Part of the pseudocode to read student records

```
i ← 1
while (i < 31)
{
        read (students [i]).id
        read (students [i]).name
        read (students [i]).grade
        i ← i + 1
}
```

Arrays versus Array of Records

- Both an array and an array of records represent a list of items.
- An array can be thought of as a special case of an array of records in which each element is a record with only a single field.

References

- Java Software Structures - Designing and Using Data Structures, 4th edition, Lewis and Chase.
- Data Structures and Problem Solving Using Java, 4th edition, Weiss.
- Data Structures & Algorithms in Java, 3rd edition, Drozdek.
- Data Structures and Algorithm Analysis in Java, 3rd edition, Weiss.
- Algorithms, 4th edition, Sedgewick and Wayne.
- A Practical Introduction to Data Structures and Algorithm Analysis, 3rd edition, Shaffer.
- Data Structures and Algorithms in Java, 6th edition, Goodrich, Tamassia and Goldwasser.
- Foundations of Computer Science, 2nd Edition, Forouzan and Ferous.
- www.asyrani.com
- *http://faculty.kfupm.edu.sa/ICS/jauhar/ics202/*

Any Question

???