



Data Structures and algorithms

INTRODUCTION

WEEK 1

LECTURER: DR ANWARUL PATWARY

Outline

- ▶ What are data structures?
- ▶ What are algorithms?

Data structures

- ▶ Data Structures-
 - ▶ Is the way of organizing data.
 - ▶ It makes program more efficient.

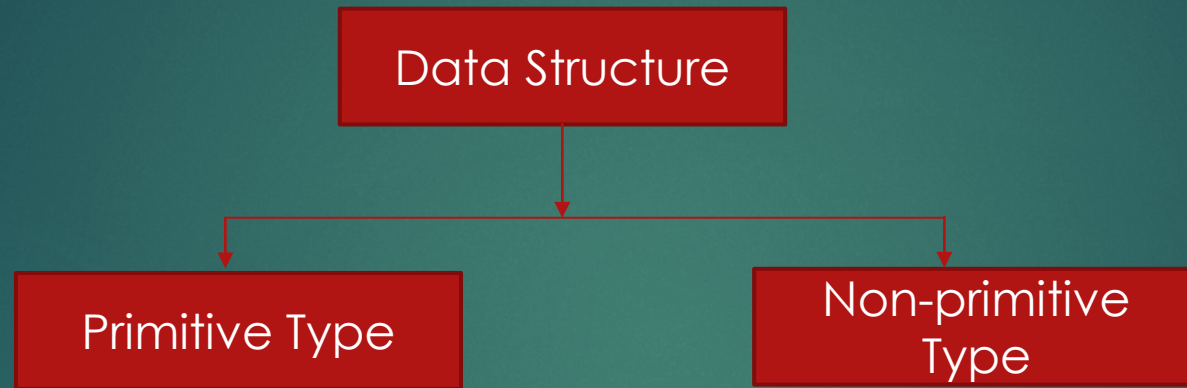
87, 90, 65, 60, 56

We discuss some examples

Why do you need Data structures?

- ▶ Proper representation of data to achieve efficiency.
- ▶ It also allows us to achieve an important object-oriented programming concept: component reuse.

Type of Data structure



- Primitive: int, double, float, and char.
- Non-primitive: linked-list, stack, queue, graph

Basic operations

45, 76, 87, 89, 90

- ▶ 1. Create
- ▶ 2. Selection
- ▶ 3. Updating
- ▶ 3. Searching
- ▶ 5. Sorting
- ▶ 6. Merging
- ▶ 7. Delete

Selecting a data structures

Select a data structure as follows:

1. **Analyze the problem** to determine the **basic operations** that must be supported.
2. Quantify the **resource constraints** for each operation.
3. Select the data structure that **best meets** these requirements.

Algorithm

- ▶ Algorithm is a set of instructions the computer will follow to solve a problem.
- ▶ An algorithm solves some computational problem.
- ▶ Algorithms are the fundamental building blocks from which programs are constructed. Only by fully understanding them it is possible to write very effective programs.

pseudocode

Step 1: Get the student grade. (Input)

Step 2: Sum of students grade.

Step 3: Summation/number of students.

Step 4: output

Basic Java

- ▶ Java is an object-oriented programming language.
- ▶ Code is organized into classes, which have instance variables and methods.

```
class HelloWorld {  
    public static void main(String args []){  
        System.out.println(" Hello World");  
    }  
}
```

Array

- ▶ An **array** is defined as a set of finite number of homogeneous elements or same data items.
- ▶ It means an array can contain one type of data only, either all integer, all float-point number or all character.

Fixed-size collections

11

- ▶ An `ArrayList` is used when the size of a collection is not known in advance, or might vary
- ▶ But in some situations, the collection size can be pre-determined from the data
- ▶ For this situation, a special fixed-size collection type is available: the *array*
- ▶ Arrays can store object references or primitive values
- ▶ Arrays use a special more-concise syntax
- ▶ Arrays are very common in a wide range of programming languages

Uses of arrays

12

- ▶ Arrays are used when we have large numbers of same-typed values or objects that we want to operate on as a collection
 - ▶ A collection of temperatures that we want to average
 - ▶ A collection of student marks that we want to analyse
 - ▶ A collection of names that we want to sort
- ▶ e.g. Bureau of Meteorology monthly data
 - ▶ We know there are twelve months in a year

ALBANY

Max	25.1	25.1	24.1	21.5	18.7	16.6	15.7	15.9	17.4	18.8	20.8	23.4
Min	13.5	14.3	13.3	11.6	9.8	8.1	7.4	7.4	7.9	9.0	10.6	12.3
Rain	28	25	29	66	102	104	126	104	81	80	46	24

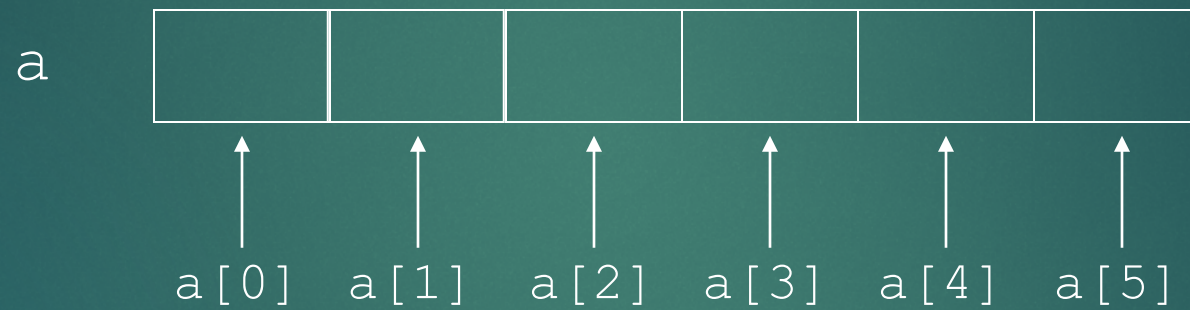
PERTH AIRPORT

Max	31.4	31.7	29.5	25.2	21.4	18.7	17.6	18.3	20.0	22.3	25.4	28.5
Min	16.7	17.4	15.7	12.7	10.2	9.0	8.0	7.9	8.8	10.1	12.4	14.6
Rain	8	14	15	46	108	175	164	117	68	48	25	12

Arrays

13

- ▶ An array is an indexed sequence of variables of the same type



- ▶ The array is called `a`
- ▶ Its elements are called `a[0]`, `a[1]`, `a[2]`, *etc.*
 - ▶ Each element is a separate variable
- ▶ Notice that indexing starts from 0
 - ▶ Same as with `ArrayList` and `String`

Declaring arrays

14

- ▶ An array variable is declared using the usual syntax
 - ▶ The `[]` denotes an array variable

```
int[] a;
```

- ▶ Declares `a` to be a variable representing an array of `ints`

```
double[] temps;
```

- ▶ Declares `temps` to be a variable representing an array of `doubles`

```
String[] names;
```

- ▶ Declares `names` to be a variable representing an array of `Strings`

```
Student[] marks;
```

- ▶ Declares `marks` to be a variable representing an array of `Students`

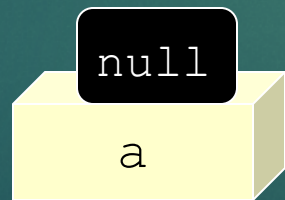
Creating Arrays I

15

- ▶ An array is an *object* in a Java program
- ▶ Therefore the declaration simply creates a variable to “point to” the array, but does not create the array itself
- ▶ Hence the declaration

```
int[] a;
```

allocates a space called `a`, big enough to hold an *object reference*, and initialised to `null`



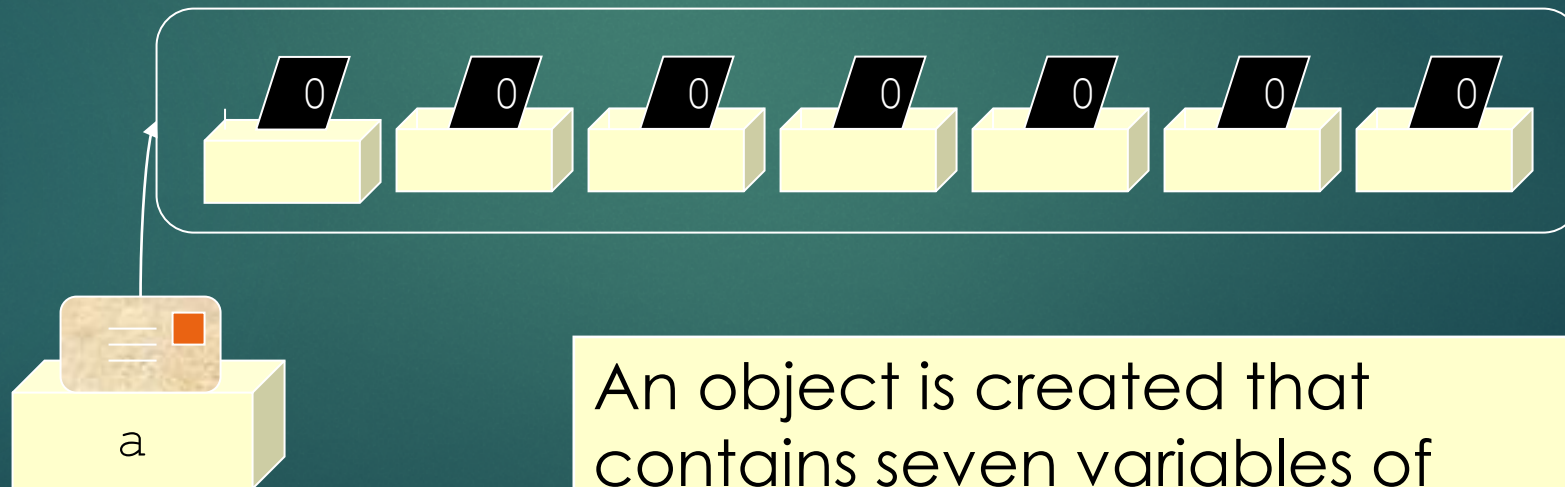
The declaration allocates space for the array reference, but **not** for the array itself

Creating Arrays II

16

- In order to actually create the array, we must use the keyword `new` (just like creating any other object)

```
a = new int[7];
```



An object is created that contains seven variables of type `int`

The size of an array

17

- ▶ The expression `a.length` denotes the size of the array currently pointed to by the variable `a`
 - ▶ We will see examples of its use later
- ▶ The size of an array is a property of the object that is created, not the variable that points to it
- ▶ An array variable can point to different arrays at different times, possibly with different sizes

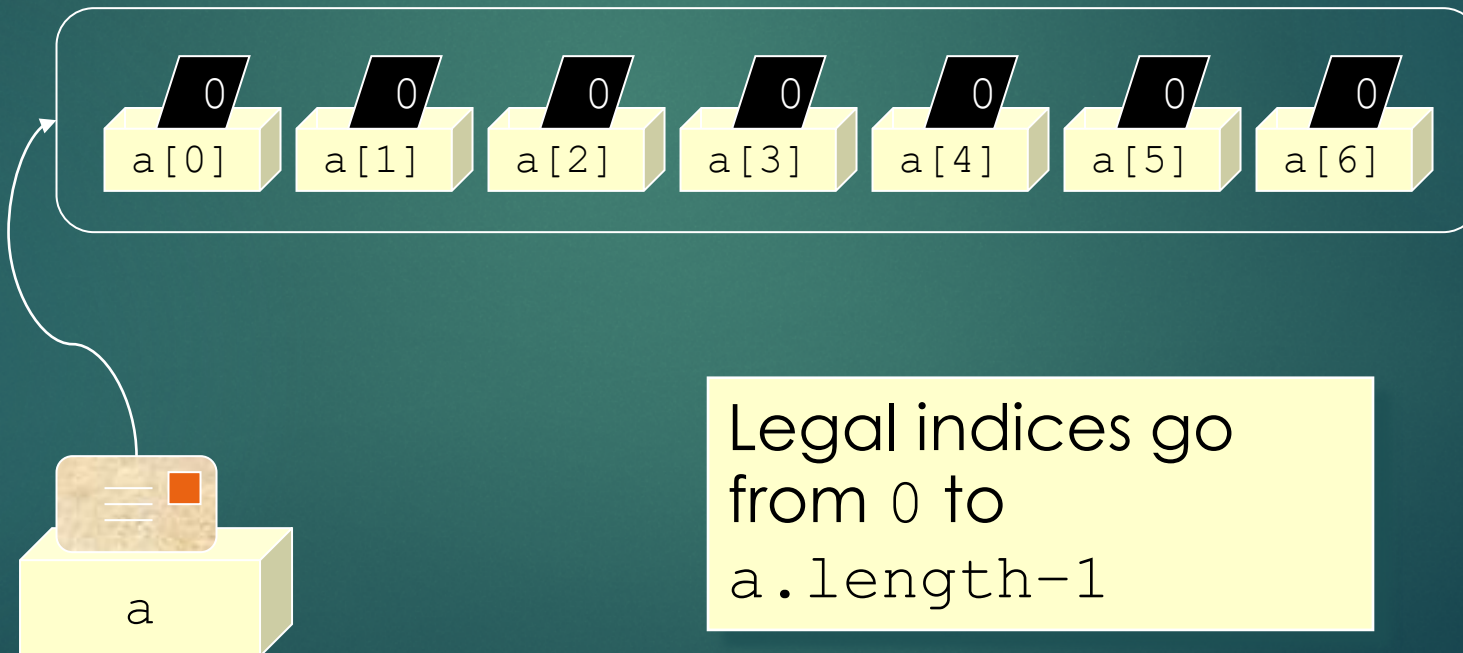
```
int[] a;  
a = new int[7];  
System.out.println(a.length);  
a = new int[666];  
System.out.println(a.length);
```

7
666

Creating Arrays III

18

- The seven variables do not have individual names
- They are referred to by the array name, and their index



Referencing array elements

19

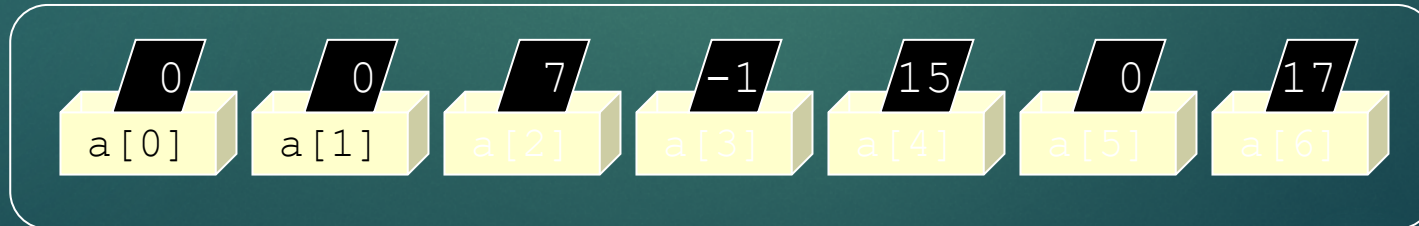
- Array elements can be used in the same ways and in the same contexts as any other variable of that type

```
a[4] = 15;
```

```
a[2] = 7;
```

```
a[3] = 2 * a[2] - a[4];
```

```
a[6] = a[0] + 17;
```



Referencing array elements

20

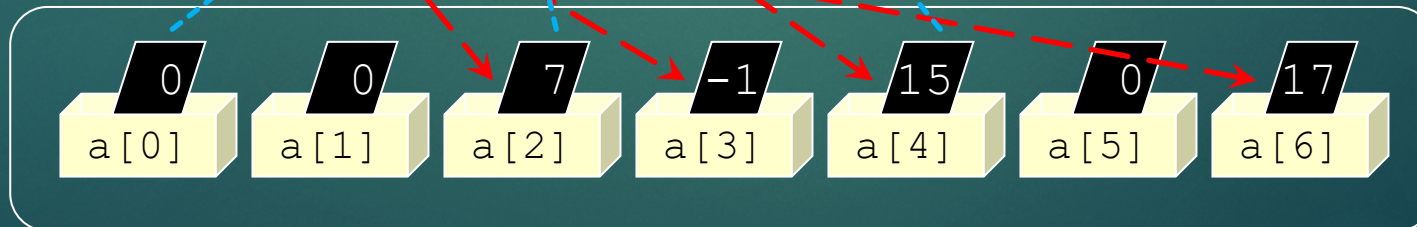
- Array elements can be used in the same ways and in the same contexts as any other variable of that type

```
a[4] = 15;
```

```
a[2] = 7;
```

```
a[3] = 2 * a[2] - a[4];
```

```
a[6] = a[0] + 17;
```



Indexing arrays

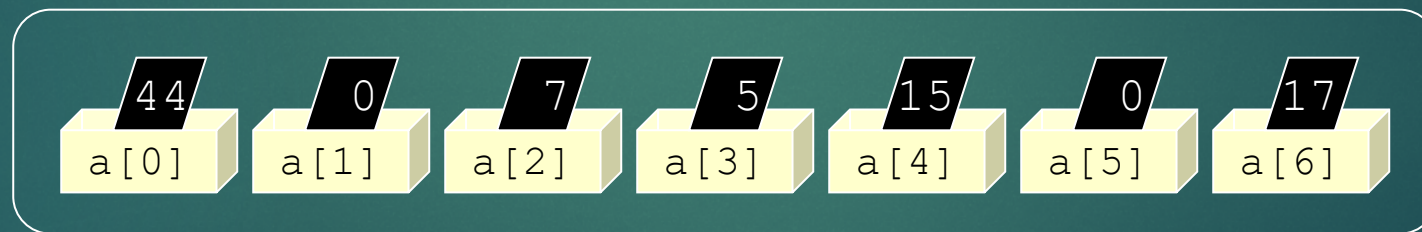
21

- ▶ A lot of the power of arrays comes from the fact that the index can be a *variable* or an *expression*

```
int x = 3;
```

```
a[x] = 5;
```

```
a[3-x] = a[2*x] * 2 + 10;
```



- ▶ This is especially useful when arrays are manipulated inside loops

Indexing arrays

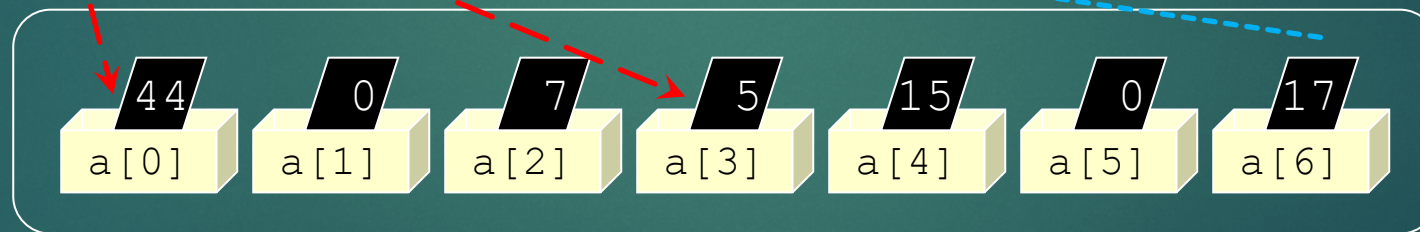
22

- ▶ A lot of the power of arrays comes from the fact that the index can be a *variable* or an *expression*

```
int x = 3;
```

```
a[x] = 5;
```

```
a[3-x] = a[2*x] * 2 + 10;
```



- ▶ This is especially useful when arrays are manipulated inside loops

Summing the integers in an array

23

- `sum({5, 8, 6, 9, 7})` returns 35

```
private int sum(int[] a)
{
    int sum = 0;
    for (int i : a)
    {
        sum += i;
    }
    return sum;
}
```

accumulating variable



- Here `i` is an element of the array `a`
 - For-each loops work the same way with arrays as with `ArrayLists`