Programming in Java

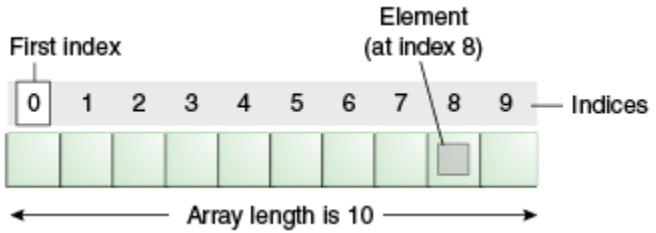
4. Arrays and Strings

Introduction

- Arrays and Strings are special Java classes
  - Special in the sense that the allow for a syntax that is more like regular programming languages
  - For example, we have been treating Strings like primitive variables
  - Java provides that syntax as “syntactic sugar” to make it easier to code in a natural stype

Arrays

- Arrays is Java look just like arrays in other programming languages
  - An array name with an indexed set of entries like a[0] etc.
  - We cannot reference an array entry that is out of range
  - Arrays know how long they are




```java
public class Runner {

    public static void main(String[] args) {
        // defining an array by providing initial values
        int [] arr = {11,11,12,13};
        System.out.println("arr[] is a" + arr + "with length " + arr.length);
        for (int index = 0 ; index < arr.length; index++) {
            System.out.println("Entry " + index + " is " + arr[index]);
        }
    }
}
```

```
arr[] is a[I@251a69d7 with length 4
Entry 0 is 11
Entry 1 is 11
Entry 2 is 12
Entry 3 is 13
```

Arrays

- Arrays are allocated heap objects

```
public static void main(String[] args) {
    // define an array object that will reference a block of memory containing
    // 4 ints
    int [] arr = null;
    // allocate the memory
    arr = new int[4];
    // initialize
    arr[0] = 10;
    arr[1] = 11;
    arr[2] = 12;
    arr[3] = 13;
    System.out.println("arr[] is a" + arr + " with length " + arr.length);

    for (int index = 0 ; index < arr.length; index++) {
        System.out.println("Entry " + index + " is " + arr[index]);
    }
}
```

```
arr[] is a[I@251a69d7 with length 4
Entry 0 is 10
Entry 1 is 11
Entry 2 is 12
Entry 3 is 13
```

What are Arrays

- An array is a contiguous block of memory on the heap
  - The length of an array is fixed – they can't grow or shrink
- All of the elements in an array must be the same type
  - This means they are all the same size
  - This allow for very fast look ups
  - We don't have to loop through an array to find an element
- If we want to find element at index 3 in an array of ints
  - If the array starts at memory location '4577' and int is 4 bytes wide
  - Then the memory location of the element we want is '4577 +(3 + sizeOf(int))'
  - This allows for constant access time to any element no matter how big the array is

Array Initialization

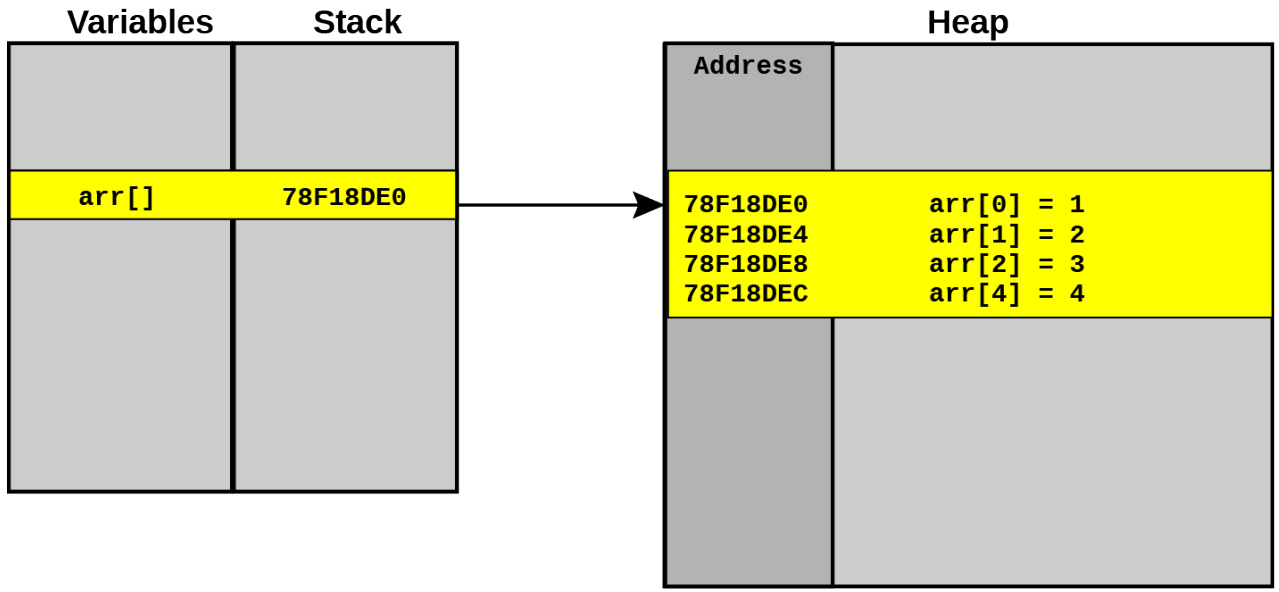
- We create a new array with the new Operator
  - The variable numbers is a reference to the array being created
  - Its type is ‘int[]’ which means it points to an array of ints
- The new operator actually creates the the array in memory
  - Then returns the reference to the heap memory location
- Since we don’t explicitly initialize the elements
  - The are set to the default zero value

```
public class IntArrayUninitialized {
    public static void main(String[] args) {
        int[] numbers = new int[5];  // creates an array of 5 ints

        // Print default values (all will be 0)
        for (int i = 0; i < numbers.length; i++) {
            System.out.println("numbers[" + i + "] = " + numbers[i]);
        }
    }
}
```

Arrays in Memory

```
int[] arr = {1,2,3,4}
```



Array Initialization

- We can also statically initialize an array by providing a list of value as shown.
  - Java can figure out the length needed by counting the number of elements

```
public class IntArrayStaticInit {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50};  // static initialization

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

Array Iteration

- The index of the array provides a built in way to loop over the element
  - This is called an iterator
- We can use this to write a more compact for loop that iterates over an array
- In the example, the for loop can be read as
  - *For each value num in the list number, execute the print statement*
  - *Start at the beginning of the array and continue until you reach the end of the array*

```
public class IntArrayExample {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50};

        for (int num : numbers) {
            System.out.println(num);
        }
    }
}
```

Multidimensional Arrays

- A multidimensional array in is an array of arrays.
- For example
  - A 2d array is an array of rows
  - Each row element is an array of columns
  - The syntax is arr[row][col]
- This extends to any number of dimensions
- Note that there is no requirement that all the rows have the same number of columns
  - When they don't, we call it ragged or jagged array

```java
public class MultiDimArrayExample {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

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

```
1 2 3
4 5 6
7 8 9
```

Copying Arrays

- Recall that an array is an object
- Assigning an array variable to another array variable does not copy the array
  - It copies the address of the array – this is called a shallow copy
  - We have to copy the array element by element – this is called a deep copy
- Two arrays are “equal” the point to the same memory location
- Two arrays are “equivalent” if corresponding elements are equal
- There are utility methods in the java.util.Arrays class that can make the task easier

Java Array Utilities

Method	Description
`Arrays.toString(arr)`	Converts array to readable string
`Arrays.sort(arr)`	Sorts the array in ascending order
`Arrays.equals(a, b)`	Checks if two arrays are equal
`Arrays.copyOf(arr, len)`	Copies array to a new array of given size
`Arrays.fill(arr, val)`	Fills array with a specified value
`Arrays.binarySearch(arr, key)`	Searches for a key in a sorted array

- By convention whenever you see '.equals()' it means equivalent
- Actual testing for equal in terms of being the same object is done with '=='

Arrays of Objects

- We can also have arrays of object
  - In this case, the elements of the arrays are the memory locations of the object
  - This meets the constant size requirement since all address are the same width

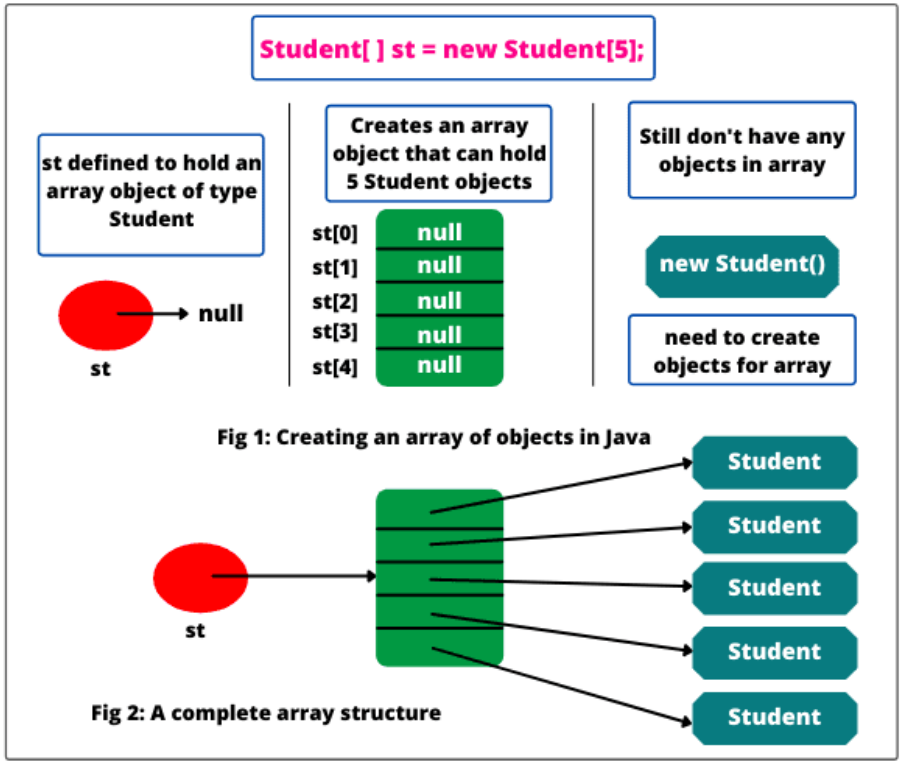


Fig 1: Creating an array of objects in Java

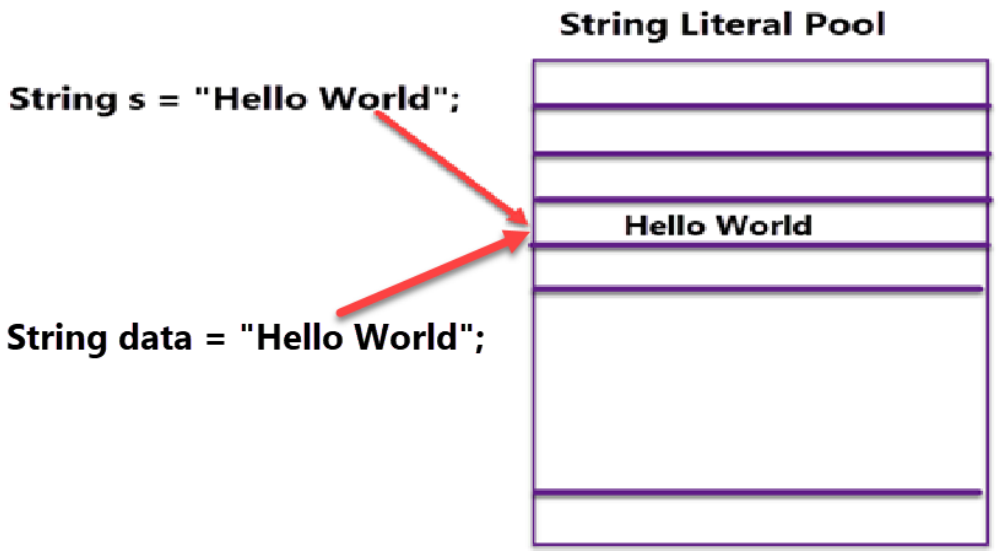
Fig 2: A complete array structure

Lab 4-1

Arrays

Strings

- Strings are objects just like other objects
- Java allows syntactic sugar so that we can write them as if they were primitive types
- Strings are immutable
  - Once created, the value of a string can not be changed
  - This is because there in only one copy of String literal, like “Hello World” that is shared by all String objects that have that literal as a value
  - These unique literals are said to be interned in a special constant area called the string pool



String Functions

- Java provides many built-in methods in the String class
  - Any method that changes the string does not alter the original string but creates a new transformed version

Method	Description	Example
`length()`	Returns number of characters	`"Hi".length()` → `2`
`charAt(index)`	Returns character at position	`"Java".charAt(1)` → `'a'`
`toUpperCase()`	Converts to uppercase	`"java".toUpperCase()` → `"JAVA"`
`toLowerCase()`	Converts to lowercase	`"JAVA".toLowerCase()` → `"java"`
`substring(start, end)`	Extracts part of string	`"Hello".substring(1, 4)` → `"ell"`
`contains(str)`	Checks if string contains substring	`"Test".contains("es")` → `true`
`indexOf(str)`	Finds index of substring	`"Java".indexOf("v")` → `2`
`trim()`	Removes leading/trailing spaces	`" hi ".trim()` → `"hi"`

StringBuilder

- Strings are immutable
  - Changing the content of a string requires creating new string
  - This is a very high overhead way to manipulate string
- StringBuilder
  - Uses a buffer of char to change the content of a string
  - Faster and uses less overhead
  - After we are finished, we convert the StringBuilder to a normal String object

```
public class StringBuilderExample {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder(); // Create a new empty StringBuilder

        sb.append("Hello");   // Add "Hello"
        sb.append(" ");       // Add a space
        sb.append("World!");  // Add "World!"

        System.out.println(sb.toString());  // Output: Hello World!
    }
}
```


Lab 4-2
Strings


Java™