Java Topics Notes (Module 2 and 3)

- Please find the attached links to the references of the topics or notes for a particular topic
- Practicals for these topics will be shared separately.
- Final Dates to cover all these topics will be 8/9/2021 to 9/9/2021 (Approx one week)

Final Keyword

https://www.geeksforgeeks.org/final-keyword-java/

Static

https://www.geeksforgeeks.org/static-keyword-java/

Transient

https://www.geeksforgeeks.org/transient-keyword-java/

Instance

https://www.geeksforgeeks.org/java-instanceof-and-its-applications/

Classes

- ### Fundamentals of classes (Objects, methods, Contructors)

  https://docs.oracle.com/javase/tutorial/java/javaOO/classes.html (Note: just cover the topic (classes) and subtopics underneath it no need to cover other topics in the trail like Nested classes)
  https://docs.oracle.com/javase/tutorial/java/javaOO/objects.html (Note: just cover the topic (objects) and subtopics underneath it no need to cover other topics in the trail like Nested classes)

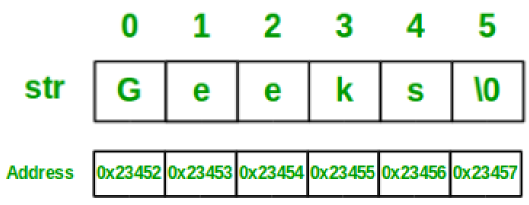
- ### This , Super keywords and access control modifiers

  https://docs.oracle.com/javase/tutorial/java/javaOO/more.html (This and access control modifiers are covered in this link) (Note: just cover the topic and subtopics underneath it no need to cover other topics in the trail like Nested classes)
  https://www.geeksforgeeks.org/super-keyword/
  https://www.geeksforgeeks.org/super-and-this-keywords-in-java/

- Fundamental classes (String, Stringbuffer)
    - ❖ Strings are defined as an array of characters. The difference between a character array and a string is that the string is terminated with a special character '\0'.Below is the basic syntax for declaring a string in **Java programming** language.
    - ❖ **Syntax:**
      String *stringVariableName* = "sequence_of_characters";

Example:

```
String str = "Geeks";
```



➔ Classes and interfaces in Strings in Java

- StringBuffer: **StringBuffer** is a peer class of **String** that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.
- StringBuilder: The **StringBuilder** in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternate to String Class as it creates a mutable sequence of characters.
- StringTokenizer: StringTokenizer class in Java is used to break a string into tokens.

➔ String Comparison can be done in 5 different ways:

Using user-defined function : Define a function to compare values with following conditions :

a) if (string1 > string2) it returns a **positive value**.

b) if both the strings are equal lexicographically
   i.e.(string1 == string2) it returns **0**.
c) if (string1 < string2) it returns a **negative value**.

Using String.equals(): In Java, string equals() method compares the two given strings based on the data/content of the string. If all the contents of both the strings are the same then it returns true. If all characters do not match, then it returns false.

Using String.equalsIgnoreCase() : The String.equalsIgnoreCase() method compares two strings irrespective of the case (lower or upper) of the string. This method returns true if the argument is not null and the contents of both the Strings are same, ignoring case, else it returns false.

Using Objects.equals() : Object.equals(Object a, Object b) method returns true if the arguments are equal to each other and false otherwise. Consequently, if both arguments are null, true is returned and if exactly one argument is null, false is returned. Otherwise, equality is determined by using the equals() method of the first argument.

Using String.compareTo() :

Syntax:

```
int str1.compareTo(String str2)
```

Working: It compares and returns the following values as follows:

a) if (string1 > string2), it returns a **positive value**.
b) if both the strings are equal lexicographically
   i.e.(string1 == string2), it returns **0**.
c) if (string1 < string2), it returns a **negative value**.

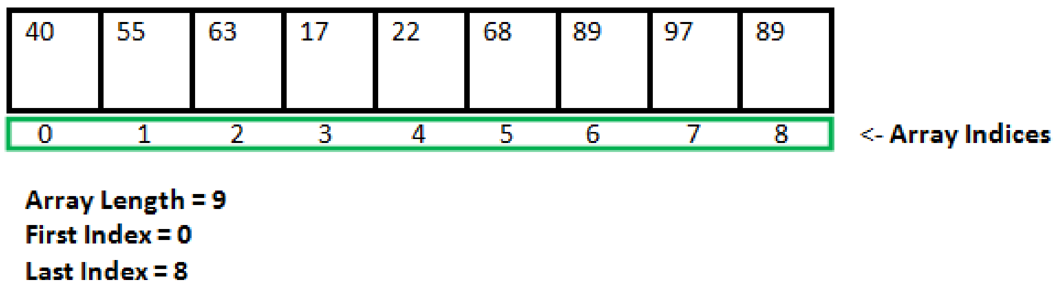
Arrays

An array is a group of like-typed variables that are referred to by a common name. Arrays in Java work differently than they do in C/C++. Following are some important point about Java arrays:

- In Java all arrays are dynamically allocated.(discussed below)

- Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.
- Java array can be also be used as a static field, a local variable, or a method parameter.
- The **size** of an array must be specified by an int value and not long or short.

An array can contain primitive data types as well as objects of a class depending on the definition of the array. In the case of primitive data types, the actual values are stored in contiguous memory locations. In case of objects of a class, the actual objects are stored in heap segment.



Array Declarations:

```
type var-name[];
OR
type[] var-name;
```

Example:

```
// both are valid declarations
int intArray[];
or int[] intArray;

byte byteArray[];
short shortsArray[];
boolean booleanArray[];
long longArray[];
float floatArray[];
double doubleArray[];
char charArray[];

// an array of references to objects of
// the class MyClass (a class created by
// user)
MyClass myClassArray[];

Object[]  ao,        // array of Object
Collection[] ca;  // array of Collection
                     // of unknown type
```

Instantiation of an Array

```
var-name = new type [size];
```

Example:

```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

OR

```
int[] intArray = new int[20]; // combining both statements in one
```

Note :

1. The elements in the array allocated by *new* will automatically be initialized to **zero** (for numeric types), **false** (for boolean), or **null** (for reference types). Refer Default array values in Java

2. Obtaining an array is a two-step process. First, you must declare a variable of the desired array type. Second, you must allocate the memory that will hold the array, using new, and assign it to the array variable. Thus, **in Java all arrays are dynamically allocated.**

Array Literal

In a situation in which the size of the array and variables of the array are already known, array literals can be used.

```
int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };

 // Declaring array literal
```

Note:

- The length of this array determines the length of the created array.
- There is no need to write the new int[] part in the latest versions of Java.

Accessing elements of an Array

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of the array can be accessed using Java for Loop.

```
// accessing the elements of the specified array

for (int i = 0; i < arr.length; i++)

    System.out.println("Element at index " + i + " : "+ arr[i]);
```

You can also access java arrays using foreach loops

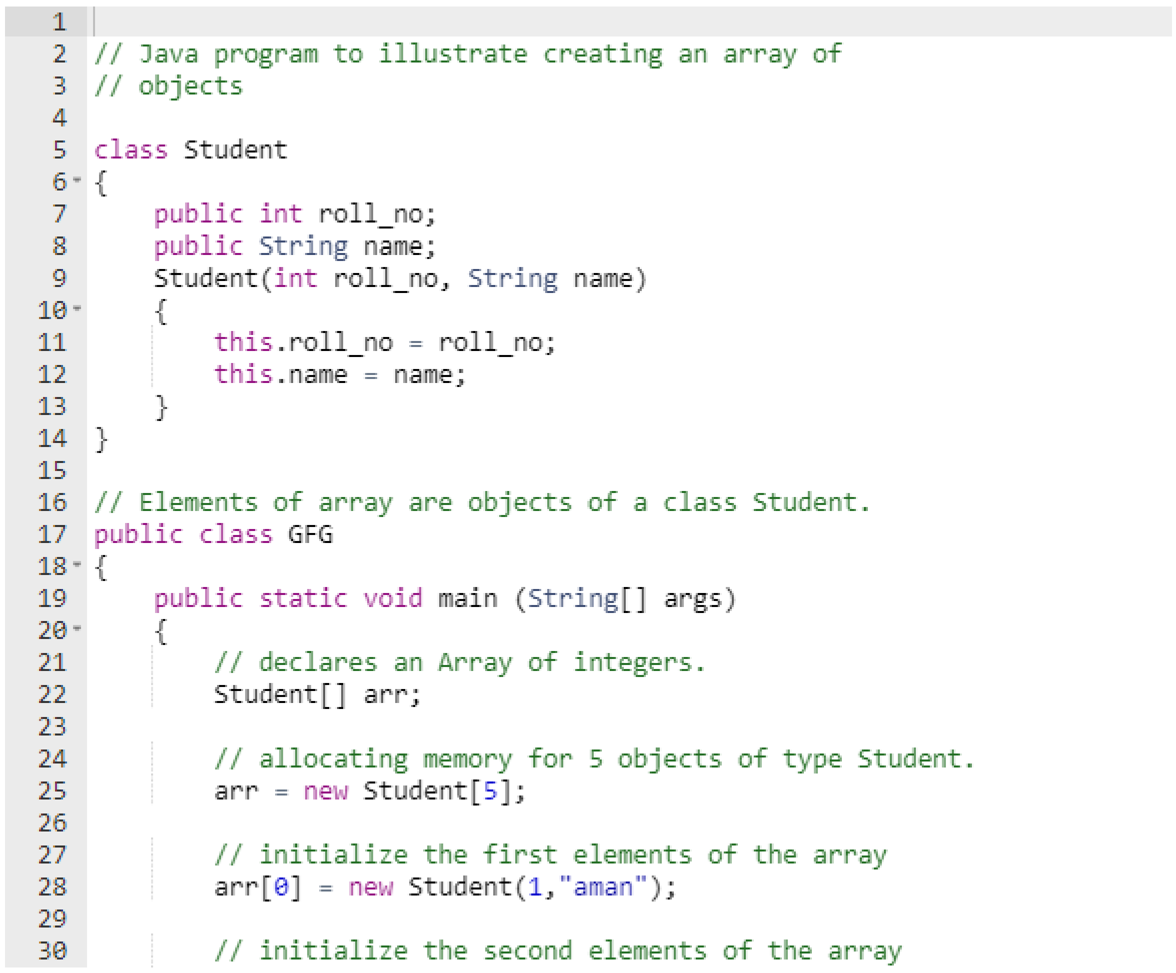
Arrays of Objects

An array of objects is created just like an array of primitive type data items in the following way.

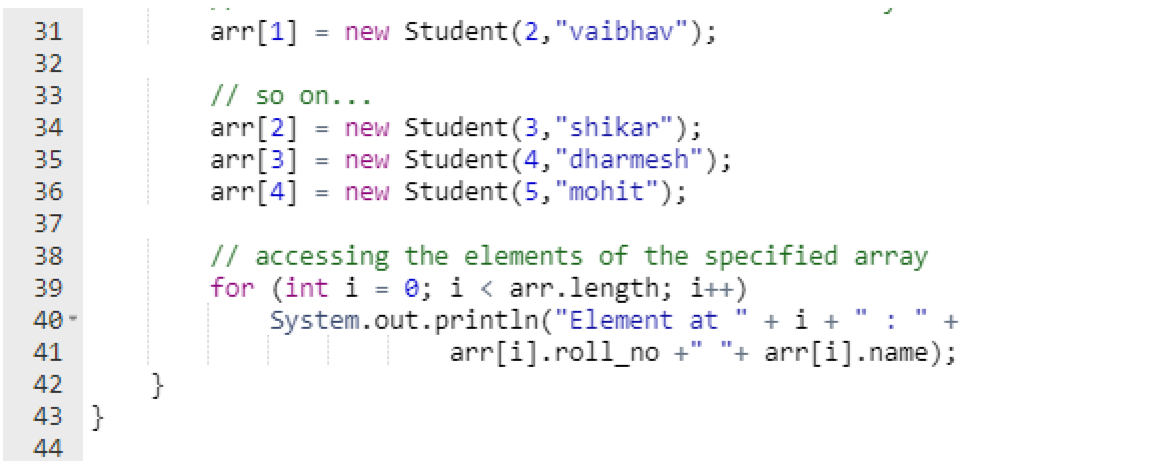
```
Student[] arr = new Student[7]; //student is a user-defined class
```

The studentArray contains seven memory spaces each of the size of the student class in which the address of seven Student objects can be stored.

Example:



```java

// Java program to illustrate creating an array of
// objects

class Student
{
    public int roll_no;
    public String name;
    Student(int roll_no, String name)
    {
        this.roll_no = roll_no;
        this.name = name;
    }
}

// Elements of array are objects of a class Student.
public class GFG
{
    public static void main (String[] args)
    {
        // declares an Array of integers.
        Student[] arr;

        // allocating memory for 5 objects of type Student.
        arr = new Student[5];

        // initialize the first elements of the array
        arr[0] = new Student(1,"aman");

        // initialize the second elements of the array
```



```java
        arr[1] = new Student(2,"vaibhav");

        // so on...
        arr[2] = new Student(3,"shikar");
        arr[3] = new Student(4,"dharmesh");
        arr[4] = new Student(5,"mohit");

        // accessing the elements of the specified array
        for (int i = 0; i < arr.length; i++)
            System.out.println("Element at " + i + " : " +
                        arr[i].roll_no +" "+ arr[i].name);
    }
}

```

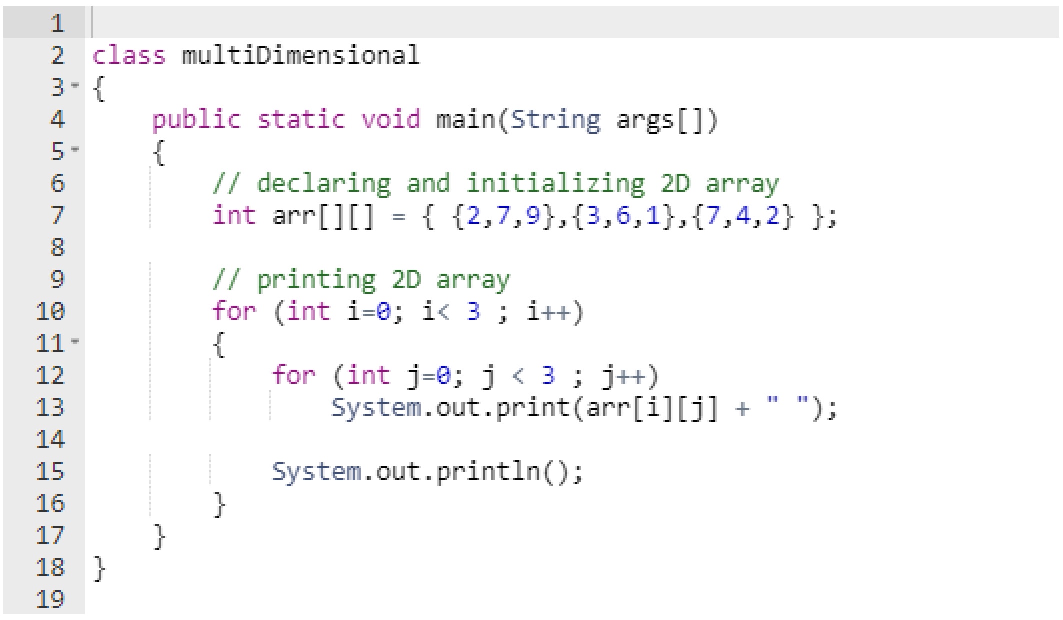
What happens if we try to access element outside the array size?

Compiler throws **ArrayIndexOutOfBoundsException** to indicate that array has been accessed with an illegal index. The index is negative, greater than, or equal to size of array.

Multidimensional Arrays

Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension. Examples:

```
int[][] intArray = new int[10][20]; //a 2D array or matrix
int[][][] intArray = new int[10][20][10]; //a 3D array
```



```java

class multiDimensional
{
    public static void main(String args[])
    {
        // declaring and initializing 2D array
        int arr[][] = { {2,7,9},{3,6,1},{7,4,2} };

        // printing 2D array
        for (int i=0; i< 3 ; i++)
        {
            for (int j=0; j < 3 ; j++)
                System.out.print(arr[i][j] + " ");

            System.out.println();
        }
    }
}

```

Output:

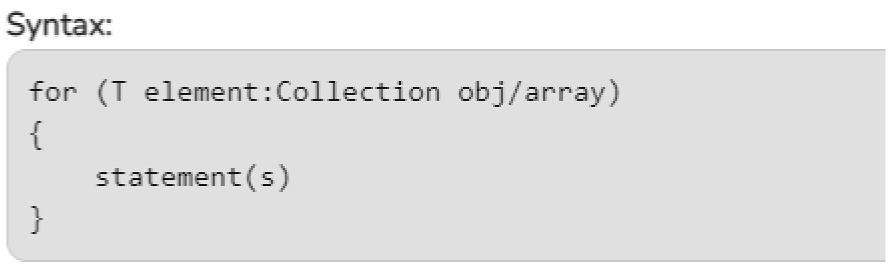
```
2 7 9
3 6 1
7 4 2
```

Collections

https://www.javatpoint.com/collections-in-java

Enhanced For-loop

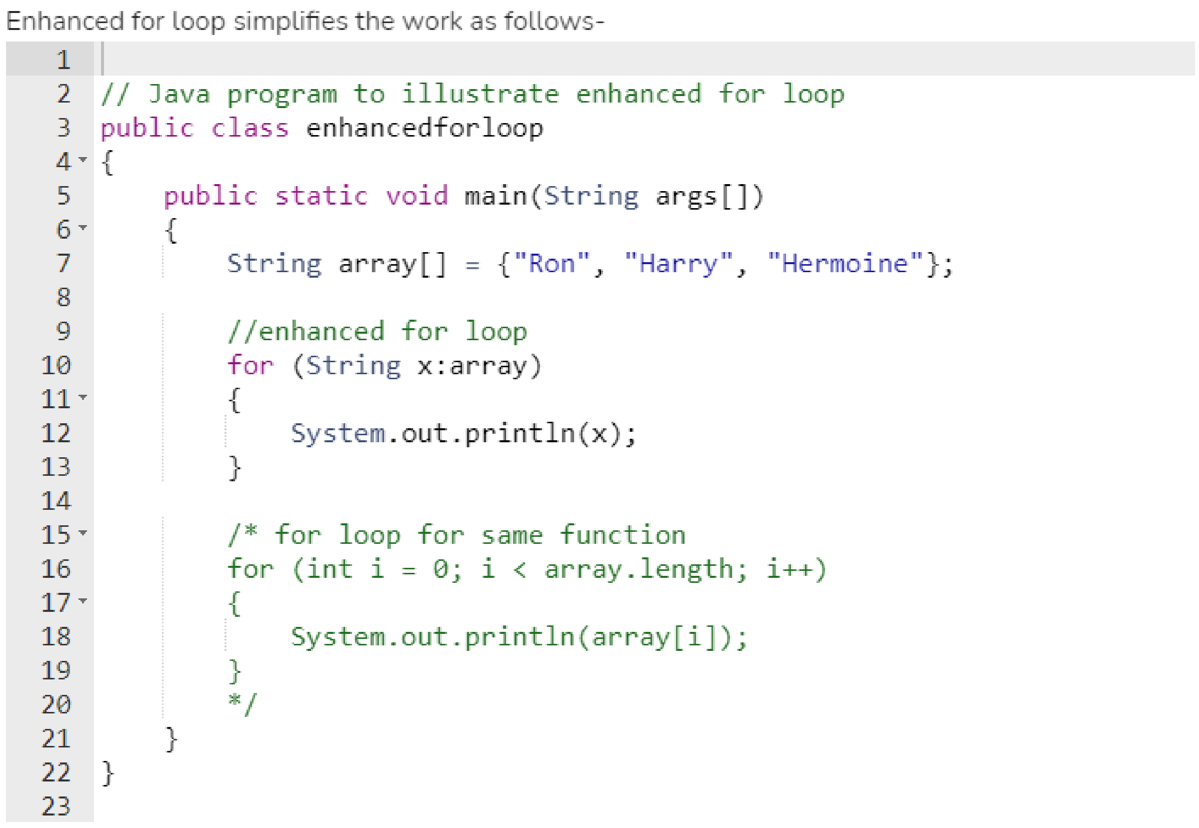
- Enhanced for loop provides a simpler way to iterate through the elements of a collection or an array. It is inflexible and should be used only when there is a need to iterate through the elements in a sequential manner without knowing the index of the currently processed element.
- The object/variable is immutable when an enhanced for loop is used i.e., it ensures that the values in the array can not be modified, so it can be said as a read-only loop where you can't update the values as opposed to other loops where values can be modified.



Syntax:

```
for (T element:Collection obj/array)
{
    statement(s)
}
```

Example Code:



Enhanced for loop simplifies the work as follows-

```

// Java program to illustrate enhanced for loop
public class enhancedforloop
{
    public static void main(String args[])
    {
        String array[] = {"Ron", "Harry", "Hermoine"};

        //enhanced for loop
        for (String x:array)
        {
            System.out.println(x);
        }

        /* for loop for same function
        for (int i = 0; i < array.length; i++)
        {
            System.out.println(array[i]);
        }
        */
    }
}

```

Date and Calendar

https://www.javatpoint.com/java-util-date
https://www.javatpoint.com/java-util-calendar

Practicals

Arrays

1. Find the duplicate in a limited range array

   Given a limited range array of size n and containing elements between 1 and n-1 with one element repeating, find the duplicate number in it without using extra space.
   For example:

```
Input:  { 1, 2, 3, 4, 4 }
Output: The duplicate element is 4


Input:  { 1, 2, 3, 4, 2 }
Output: The duplicate element is 2
```

2. Sort an array of 0's, 1's, and 2's

   Given an array containing only 0's, 1's, and 2's, sort it in linear time and using constant space.

   For example,

```
Input:  { 0, 1, 2, 2, 1, 0, 0, 2, 0, 1, 1, 0 }

Output: { 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2 }
```

String

1. Given a string s consisting of some words separated by some number of spaces, return *the length of the last word in the string.*A word is a maximal substring consisting of non-space characters only.

   Example 1:

   ```
   Input: s = "Hello World"
   Output: 5
   Explanation: The last word is "World" with length 5.
   ```

   Example 2:

   ```
   Input: s = "   fly me   to   the moon  "
   Output: 4
   Explanation: The last word is "moon" with length 4.
   ```

   Example 3:

   ```
   Input: s = "luffy is still joyboy"
   Output: 6
   Explanation: The last word is "joyboy" with length 6.
   ```

2. Valid anagram

   Given two strings s and t, return true *if* t *is an anagram of* s*, and* false *otherwise*.

   Example 1:

   ```
   Input: s = "anagram", t = "nagaram"
   Output: true
   ```

   Example 2:

   ```
   Input: s = "rat", t = "car"
   Output: false
   ```

   Constraints:

   - `1 <= s.length, t.length <= 5 *` 10^4
   - `s` and `t` consist of lowercase English letters.

Collections (basic practicals)

1. How to convert an array into a collection ?
2. How to change a collection to an array ?

   Hint: Convert a collection to an array by using list.add() and list.toArray() method of Java Util class.

3. How to iterate through elements of HashMap ?

   Hint: use iterator Method of Collection class to iterate through the HashMap.

4. How to find a sublist in a List ?

   Hint: use indexOfSubList() & lastIndexOfSubList() to check whether the sublist is there in the list or not & to find the last occurance of the sublist in the list.
5. How to rotate elements of the List ?

Date (basic practicals)

1. How to display current date and time ?
2. How to add time to date?
3. How to display time in different country's format?
4. How to display date in different formats?