JAVA

**Deepansh Sabharwal**

**Don’t forget to use ;**

**And {} for LOOPS**

1. JVM which runs the code doesn’t accept it directly.
2. How JAVA works
   1. We write the JAVA code 🡪 Compiler (JAVAC) 🡪 Byte Code 🡪 JVM (public static void main)
3. Even if we give 1000 files to the compiler, for JVM to execute the code we’ll have to specify the file to be executed.
4. Extensions:
   1. JAVA: **a.java**
   2. Byte Code: **a.class**

JVM: JAVA VIRTUAL MACHINE

JRE: JAVA RUNTIME ENVIRONMENT

1. WORA: Write Once, Run Anywhere
2. The default value in JAVA is Double 🡪 8 bytes (not float 🡪 4 bytes)
3. This is to get more accurate answers
4. A diagram of data types

   Description automatically generated
5. For compiling: **javac Hello.java**
6. For running: **java Hello**
7. Let’s say we have byte b = 127 **{127 is the max value of byte}**
   1. We have int a = 234
      1. Can we do b = a?? **NO**
      2. Can we do a = b?? **YES {Conversion}**
   2. How do we deal with b = a??
      1. We can convert the type of the value by **CASTING (explicit conversion)**
      2. b = (byte) a;
      3. If a = 257 (which is outside of the range of byte), we perform the modulus function (%)
      4. It will divide **a/(byte range, i.e 256)** and the answer is the remainder.
8. String [] **args** is an array of String objects passed to the main method when the program starts.
   1. A screenshot of a computer code

      Description automatically generated
   2. Output:
      1. **apple**\n**orange**\n**banana**
9. float f = 5.6f;
   1. Always use f after the number for float
   2. If we convert it to int….
   3. Int t = (int)f;
   4. **Output is the value before the decimal.**
10. For arithmetic operations
    1. Everything is same as +, - etc.
    2. **++num is pre – increment** (increments and then fetch the value)
    3. **num++ is post – increment** (fetches the value and then increments it)
11. Logical Operators
    1. AND: &&
    2. OR: ||
    3. NOT: !
12. For IF – ELSE statements
    1. No indentation like python
    2. Use {} after if () for multiple statements
    3. Use else to print counter answer
    4. Use else if () to compare more conditions
13. Ternary Operator
    1. Instead of using if-else every time, this operator can be used to simplify the code.
    2. **Not always useful**
    3. Ex: result = **n%2==0 ? 10 : 39;**
14. Switch statement:
    1. Syntax: switch(n) {
       1. Case value (1,2,3…):
          1. Condition
          2. Break;
          3. Default Statement, same syntax, don’t use break
          4. With newer versions of java, strings are supported as well
          5. Newer version syntax
             1. **Case: “Saturday”, “Sunday” -> System.out.println()**  
                }

***LOOPS***

1. While Loop
   1. Syntax:
      1. while () {everything to be written here}
      2. Do while: **do {text/commands} while ();**
   2. Use when number of iterations are not known
2. For Loop
   1. Syntax:
      1. For (int i = 0, i<=7, i++) {commands} (for this i++, fetches and then increments)
   2. Use when number of iterations are known

**Classes in JAVA**

**EVERY CLASS IN JAVA EXTENDS OBJECT**

* So, let’s say we make a class Calculator.
* Now we make a method in that that adds 2 numbers
  + **Let’s call this add(int n1, int n2){}**
* Now we specify what should this method return for us??
  + For that we use the return keyword and return the variable we made.
* **Now how can we access this??**
  + We write the usual code.
  + We then create a primitive function: Calculator calc = new Calculator();
    - * + Here **calc** is the **primitive function** and **new** is the **instance** of the class as we did in python
  + Then we create the usual result variable that adds both and print it
* #Facts: JDK compiles and JVM runs, but JVM is a part of JRE. What is JRE?
  + If you use any external files, classes into your code, you need JRE to do that so basically you need JRE to **run** the JAVA code.
* Methods
  + So even if we don’t write any code, there’s a default method in JAVA that is **main** that is used for execution of the program we write
  + Now let’s say I create another class Computer.
    - I create 2 methods, Playmusic and GetmeaPen, pass any parameters if required and give any necessary if else commands.
    - If I create a method using void, we want the method to return nothing
    - We can use different method options such as double, String, int etc.
    - Class Computer
      * public void Playmusic() {write the function behaviour here}
      * public String GetmeaPen(int cost) {write the function behaviour here}
    - **Operator overloading**: we create an add function that adds 2 integer and then we create an add1 function that accepts 2 floats, so this can be done and is known as Operator overloading. **Be very sure to check which function have you called and are the correct parameters given.**
* Stack vs Heap
  + Stack
    - **Used for:** Storing method calls and local variables.
    - **Fast access:** Very quick to allocate and deallocate.
    - **Automatic cleanup:** When a method finishes, everything in its stack frame is removed.
    - **Size:** Usually small.
    - **Example:** int x = 10; inside a method → stored on the stack.
  + Heap
    - **Used for:** Storing objects and instance variables.
    - **Slower access:** Takes longer to manage.
    - **Garbage collected:** Java automatically removes unused objects.
    - **Size:** Larger.
    - **Example:** Student s = new Student(); → s is on the stack, but the actual object is in the heap.
  + Simple analogy
    - **Stack =** To-do list → temporary and quick.
    - **Heap =** Bookshelf → long-term storage, shared and organized.

**ARRAYS IN JAVA**

* Why do we need an **ARRAY**?
* Instead of storing **different values in different variables**, we can use an **array to store multiples values in a single variable**.
  + **int nums [] = {12, 4, 5}**
  + We can access the elements **System.out.println(nums[1])**
  + Also, this is mutable, i.e., we can assign a new value to the given index.
  + nums[1] = 6; this will make the first index value 6 instead of 4 as shown in the example above.
  + int nums [] = new int[4], this is not assigning a value, this says that we make an empty array of length 4.
  + If we print now, we get a value of 0 if we print any index.
* **Multi- Dimensional ARRAY**
  + What we basically do here is create 2D arrays.
  + Let’s say we create an array nums
  + Now to create it we use nums[][].
  + int nums[][[] = new int[length][length];
  + Now we can use nested for loops to add the values in this array.
  + #New concepts
    - **Math module**: We used Math.random() in one of our codes. This module needs to be **TYPECASTED** if you want to use it as an integer as its **default value is float (double) < 1**. We do this by nums[i][j] = (int)(Math.random() \*100); (See code for MD array)
    - Print: if we want the output to be printed on the same line, we can simply use System.out.print(nums[][]);. No need of **println.**
    - Same method can be used for 3D arrays.
* Drawbacks of ARRAY: while it is fast and an efficient way to do work, it cannot be used with different data types such as string and int together.
* **By default, the value of integer array will be 0, if no value specified.**
* Instead of specifying the number of iterations and making a mistake, we can use variable.length to get the exact number.
  + **for (int i = 0; i<nums.length; i++){}**
* Array of objects:
  + So, we basically create a class and them make instance variables, name/marks/rollno.
  + We then create the object s1
    - Student s1 = new Student();
    - Then we create different values like s1.name, s1.marks etc
    - Now we can create an array
      * Student students[] = new Student[3];
      * Assign each instance like students[0] = s1
      * Then use for loop to print

**Enhanced For Loop**

* + We can use this syntax instead of the huge syntax
    - for (int/data type **n** : **nums**) {}
    - Like in the case of the object array we made, we use the data type of the class which in that case was Student

**Strings in JAVA**

* How do we create a string??
* The default value is **null.**
* **String is itself a class in java** so we can write the usual method of calling a method in a class that is using the new keyword: **String name =. new String(“Name”);**
* But we **don’t have to do that**. We can simply say **String name = “Deepansh”** and that is what we usually use (Stored in the heap memory).
* Since it is a class, it’ll have different methods to use from.
* Strings are immutable.
* String buffer or String builder can help us work our way through the immutable strings.
  + String Buffer
    - Has a default buffer/extension for 16 characters. If I input Deepansh, which is 8 characters, then the buffer would be 16+8 = 24.
    - Has different methods to append or delete or insert characters

**Static keyword JAVA**

* If we take the case of the code we have written, we can see that the type of mobile will remain same for both instances, instead of assigning a new value to each object, we can make the instance variable type **static: static String type;**
* Now when we assign the value as we did in array of objects, **no need to write s1.type = “Phone”**
* We can simply write **ClassName.type = “Phone”**, and this value will be **changed at all instances.**
* Importance of static keyword in public static void main
  + Without static, main would be an instance method, that means we would have to create an object to call the class, which is usually the file name. But the point to be considered here is that main is the pillar of execution, we can’t create an object of the class. Which makes static keyword very essential.

**Naming convention in JAVA**

* It is recommended to follow CAMEL CASING.
* For Class, Constructors, Interfaces: The first letter should be capital.
* Variable, methods: if small, then all lowercase, if big use camel casing like showMyData().
* Constants: All uppercase, PIE, DATA etc.

**Packages**

* It is a good practice to create packages/folders in JAVA of a class or similar files.
* You do this by writing this on top of the file **packages foldername;**
* Let’s say we have a big outer folder with one inner folder and a file. **The methods of the inner folder files cannot be accessed directly by the single file in the outer folder.**
  + We use import **foldername.filename/classname**

**Final keyword in JAVA**

* Let’s say I create an integer num that has an assigned value of 8. Now if I use num again to assign 9 to it, it’ll work.
* If we want to fix the value, we write final int num = 8;
* Now reassigning gives an error
* Now we have a class Calc as we used for all the examples. I don’t want any other class to inherit from it. I can just write **final class ClassName {}**, and no inheritance takes place.

**Pillars Of OOP**

**EVERY CLASS IN JAVA EXTENDS OBJECT CLASS.**

* **Encapsulation**
  + If we want to hide certain information, let’s say we make a class Human, we want to hide the age and name from getting accessed directly. We can use encapsulation for this.
  + If we have made the instance variables private, they cannot be accessed inside the class we created. There are different ways to access them throughout and assign them values.
  + Ways to do it
    - If we create a class Human, which has 2 instance variables, age and name, we can access them easily throughout (See the OOP techniques from above).
    - If we make both these attributes private, just by adding a private keyword in front of them, they cannot be accessed outside the class (By standard OOP methods).
  + Ways to access them outside the class
    - Using GETTERS AND SETTERS.
    - Create a method, public int getage(){return age;}. Do the same for name. Use this if you assigned values directly to the instance variables
    - Create a method, public void setage(int a){age = a;}. Do the same for name. We have used for SETTERS as we don’t want to them to return anything.
    - Now we can access both the set and get methods outside the primary class by using the standard OOP technique.
      * Human obj = new Human();
      * obj.setage(12);
      * Then we can print the getname() and getage() as set won’t return anything.
  + THIS variable in JAVA.
    - When we use the setter, if we want to call the local variable and the instance variable together, we use THIS.
    - public void setage(int age){this.age = age;}. This syntax is to be followed from now onwards.
* **Constructors** 
  + Every time you create an object; the constructor will be called.
  + If in encapsulation I create a constructor, public Human() {age = 12, name = “john”}
  + The getters will return this if I haven’t used the setters outside the function like above statements.
  + This will be the default value John : 12 and not null : 0
  + Similarly, we can create a parameterized constructor, class Human(int a , String b){}
  + If we remove both the constructors we created, java creates a default constructor on its own, that doesn’t return anything.
  + Two times constructor running means, 2 new objects are created.
* **Inheritance**
  + What is Inheritance?
    - Using features of the super class
    - Let’s say you don’t have a phone, but your parent does, so which means you can say that you have one too.
  + So, I created a file ClassinJAVA, where a function Calc was defined. Now I can directly access the class Calc in any new file I create. I can simply create a new reference variable like Calc obj = new Calc();
  + Then can access all the methods in that.
  + Now if I want to inherit the Calc class into a new class, let’say AdvCalc, that has 2 new features, I can simply create a new file and on the first line which says public class Filename extends Calc.
  + Add new features and use them
  + Now again if I create a new class VeryAdvCalc, that has 2 more features, I’ll simply extend AdvCalc I created. This is MULTI-LEVEL Inheritance.
  + **Multiple Inheritance in JAVA DOES NOT WORK!**
    - **Let’s say I have 2 parent classes A and B, and I want multiple inheritance in C. Now let’s say there’s a show() that is present in both, JAVA didn’t give a solution, rather removed the problem.**
  + In JAVA, the first line in a constructor is always the super() method. [John Cena reference]
  + Every class in JAVA, **extends** the Object class, a default class with various methods inbuilt.
    - If A is the super class for B, Object is the super class for A.
* **Method Overriding**
  + If I have 2 classes as we used above, Calc (return a+b;) and AdvCalc. Now AdvCalc extends Calc which means it has add() method as well. Now if I create an object for AdvCalc which returns a+b+1, then the output printed will be that of a+b+1 and not Calc.
* **Polymorphism**
  + It basically refers to multiple behaviour.
  + There are 2 types, COMPILE TIME and RUNTIME
  + Compile time refers to method overloading and Runtime refers to method overriding.
  + So, if I create 3 classes (A, B, C) and create 3 obj respectively, all having a method show(), we are not sure which method will be called when we run it. This is Runtime Polymorphism.

Type Casting in JAVA

* Let’s say I have a float: double d = 4.5. Now I want to save this in integer, how do I do that?
* We use Type Casting for that
* We make a new variable int i. We can change the data type or loose data by int i = (int) d. Now we lost data, but it was converted to string.
* Up casting
  + I made 2 classes A and B, where B extends A. Now A has a method show() and B has show1().
  + Now I can create an A obj = (A) new B();
    - Here B shows from A which is the parent class so upcasting
  + We can even write A obj = new B(); and that is same.
* Down casting
  + If I create an object obj1 for B using the reference variable obj that belongs to class A, it gives me an error. Instead, I can typecast it and write B obj1 = (B) obj; to remove the error and create the reference for B using A and use obj1.show1().

Inner Class in JAVA

* We take the same example of class A and B, but now class B is the inner class of A.
* How do we access that?
  + - First create the standard object of class A using the syntax used above.
    - Now B is in A, so use A.B obj1 = obj1.new B();
* Anonymous Inner Class [SYNTAX IS VERY CRUCIAL]
  + So, we basically created a class A as we did previously, but now we don’t create a class B as we did above.
  + Instead, we create an obj (reference variable) for A using the common syntax
    - * class A {print in A show}
      * public static void main……….
      * A obj = new A()
      * **{**
        + **Here we write the inner class**
        + **public void show() {print in new show}**
      * **};**
      * **obj.show();**

Interface in JAVA

* It is like a class, but all the methods are by default public abstract.
  + We can write void show(); instead of public abstract void show();
* How do we create an interface??
* This time we’ll create an interface A and not class A.
  + **interface A**
  + **{void show;**
  + **void config;**
  + **} as we discussed in the first point.**
  + **Now we create class B, which doesn’t extend A but implements it.**
  + **class B implements A**
  + **{**
  + **#Here we must write all the methods of interface A, else instead of an interface, it will be an abstract class.**
  + **public void show and void**
  + **}**
  + **How do we print it?**
    - **A obj;**
    - **obj = new B();**
    - **obj.show()**
    - **obj.config();**
* We can implement more than 1 interface using the syntax
  + class B implements C,Y
* Also, an interface can EXTEND another interface.
  + interface Y extends X

**Enum in JAVA (See code)**

* So, the syntax goes like
  + enum Status {Here you write the constants, Running, Pending;}
  + int i = 5 (not important)
  + Status s = Status.ConstantName;
  + We can print the constant number like we do for slicing. Like the reference variable is s, so we can print s.ordinal(ConstantName) and it should say 0 as it’s the first one.

Lambda Expression in JAVA

* So, instead of using same syntax as INNER CLASS as we used above, we can use the LAMBDA EXPRESSION to narrow it to down to a single line code.
* How do we do it?
  + Create an interface All{void show();}
  + In the public static void main instead of creating an object for A, we write
  + A obj = () **->** System.out.println(“In show”);
  + obj.show();
* If we pass an argument of let’s say an integer (which we will first give a parameter above) we can write
  + A obj = (int i) **->** System.out.println(“In show ” + i);
  + obj.show();
* We can also return using the lambda expression.

**Exceptions in JAVA**

* 3 types of exceptions
  + Compile time error
  + Runtime error
  + Logical error
* We will mainly talk about RUNTIME ERROR for using Exception Handling.
* How do we do that?
  + Let’s say we create 2 variables, int i = 0; int j = 0;
  + Now we create a TRY\_CATCH block to resolve the error like division by 0.
  + After creating the variables, we can create these blocks and then we can write the required statements in both.
  + We can print the exception as well in the exception block (along with the print statement).
    - Now we can use multiple catch statements if we more than one exception which must be handled.
    - Like in the code for Exception handling in the file, we can see that we created an array of nums which ranged from 0-4, but we wanted to print nums[5], which is out of bounds for that array.
    - So, the first catch statement would be for ArithmeticError and the second catch statement for ArrayOutOfBounds error.
    - We use Exception e if we are unsure of which error to be handled.
    - NOTE: catch(Exception e) should always be the last exception to be handled as it the parent class for all exceptions which are predefined.
  + Throw keyword in JAVA, lets the user to pass on an exception even when there is no error.
    - The user can throw the exception in the try block itself with the actual exception name (See code).
  + If you want to create your own exception, like DeepanshException, we first must create a class with any name you want which extends Exception which is the parent class as exception is not just a simple class, it has some special features.
  + Then, follow the same syntax/code as we did above.

**Input in JAVA**

* There are several ways of taking input like InputStreamReader and BufferRead, but the code written is too complex or confusing at times. To resolve this we can now use the Scanner method to take input in JAVA.
* How do we do that?
  + First, we can print an input statement that the user can follow and write their input.
  + Then we create an object for scanner
    - Scanner sc = new Scanner(System.in) //taking input
    - This automatically imports the Scanner class on top
  + Now we can define a variable for taking the input.
    - int num = sc.nextInt(); //or any other method of your choice/requirement
  + Next, we can print num.

**Threads in JAVA**

* A thread is the smallest unit you can work with. There can be multiple threads working at the same time.
* How do we do that?
* You can create 2 classes A and B. Now you want both classes to behave like a thread and not a normal class.
  + Both class A and class B extends Thread (which is itself a class).
  + Now both A and B behave as threads, and we can classify this setup as a multiple thread setup.
  + Both A and B will not have the show() method which we’ve used previously. Instead, they’ll have a method run().
  + Now inside the main class, we create 2 objects (reference variables) and then we write
    - obj1.start();
    - obj2.start();
  + We’re simply using run() and start() to make them a thread.

**Collection API in JAVA**

* **ArrayList in JAVA**
  + Now Collection is of 3 types: sets, list and queue.
  + So how do we create an ArrayList? List is a Collection API concept and so is SETS.
    - We do this by first creating a variable nums
    - Collection <Integer>nums = new ArrayList<Integer>();
    - Now there are several methods to add new numbers to this
  + It is vital to use <Integer here> else you’ll see a yellow underline as the data type is not specified
* **Sets in JAVA**