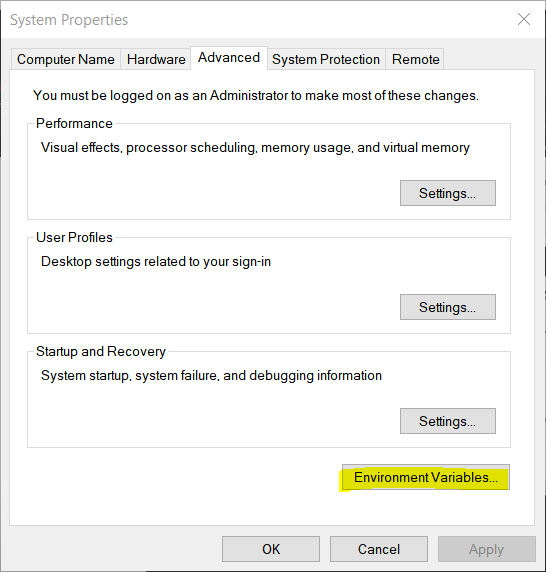
**Core Java**

* **Java:**
  + Java is made by Sun Microsystems in 1995 and later on bought by Oracle and currently Java trademark is held by Oracle. Java is updated every 6 months.
  + Java is mainly used for Mobile development, Web development as well as Enterprise applications. Kotlin, Scala, Groovy come under java technology. Java is the most readable language, we can read code line by line and it is easy to understand.
  + Java has multiple features such as **WORA (Write Once Run Anywhere), multiple thread, Collection API concept, Exception handling and many more**.
  + Java is an object oriented programming language, so it means everything should be in an object and to create an object we need to have a class.
* **Setup Java Environment:**
  + **IDE:** To write Java, we can use Notepad and WordPad for simple programming but to work with java professionally, we can use different **IDEs(Integrated Development Environment)** such as **VS Code(Light weight editor), Eclipse and IntelliJ,** where we can type the code, compile the code, run the code and debug the code.
    - **VS Code URL:** [**VS Code**](https://code.visualstudio.com/)
  + **Compiler:** To compile the java code, we need to install **JDK (Java Development Kit)** 
    - **JDK URL:** [**Oracle JDK v24**](https://www.oracle.com/java/technologies/downloads/%23java24)
  + Once we complete the installation of IDE and JDK, to verify it, we can run “**JAVA --VERSION**” command to “**CMD**” and it will give you version of Java and JDK installed to the system also by running “**JAVAC --VERSION**” to get the current version and confirmed that compiler installed correctly.

A screenshot of a computer program

AI-generated content may be incorrect.

* + While we check the versions and if we get an error then it means that path for java environment has not set up very well.
  + To set up environment variables, follow the following steps:
    - Start > Search “**Edit System Environment Variables**”
    - It will open below window, click on “Environment Variables”.



* + - * Select “path” and click on “**Edit…**” button under “**System variables**”

A screenshot of a computer program

AI-generated content may be incorrect.

* + - * Click on “**New**” and add “**C:\Program Files\Java\jdk1.8.0\_161\bin**” path or copy the path of **“…\Java\jdk1.8.0\_161\bin**” then click “**OK**”.

A screenshot of a computer program

AI-generated content may be incorrect.

* + - * Click on “**New…**” under “**User variables for…**” then add below values, and click on “**OK**” and “OK” for main window:
        + **Variable Name:** JAVA
        + **Variable value:** “**C:\Program Files\Java\jdk1.8.0\_161\bin**” path or copy the path of **“…\Java\jdk1.8.0\_161\bin**”

A screenshot of a computer

AI-generated content may be incorrect.

* **How Java Code Works on Backend:**
  + Java have JVM (Java Virtual Machine), using that we can run java application.
  + Under JVM there is OS (Operating System) comes and under that there is HW (Hardware).
  + Java is a platform independent, means when we write a Java code, it will run on any machine. Only this is required for it is that respective machine need to have JVM.
  + **JVM does not understand the Human readable code or instruction, but it accept only BYTE code. So, to convert this java code into Byte code it uses “Java Compiler”**
  + So, the basic flow is,
    - **Users create Java code > javac compiler compiled code to BYTE code > BYTE code goes on JVM > JMV look for “public static void main(String args[]” syntax to execute the code**.
  + In the real time scenario, there are 100s of files in the java project, so JVM starts with the first file/Main method file or file which contains main method file.
  + While running java code, it also requires some libraries and in java JRE (Java Runtime Environment) fulfill this request. Also, JVM is the part of JRE. JDK is only use by developers.
  + Main method contains one specific signature **“public static void main(String args[])”**

**(NOTE: In java there is only one Main method and only one file which have main method from where java code starts the execution.)**

**A diagram of a computer program

AI-generated content may be incorrect.**

* + As java follow WORA (Write Once-Run Anywhere), once developer develop application using JDK, and to use that application to other machine, we just have to install JRE and JVM (JDK does not required), and we can easily execute the code at any platform/OS.
* **Java Code Basic Code Structure:**
  + Java code file extension is **“.java”.**
  + Java code execution point is Main method of java where **“public static void main(String args[])”** main method comes.
  + To compile the code use **“javac FileName.java”** and it will create **“.class”** file.
  + Run code use **“java FileName”** command.
  + Java Basic Code Syntax:

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Java Variables:**
  + Variables are used to store values, which can be either provided by the user, predefined or assigned during the runtime.
  + **If we define variables in class called instance variable.**
  + **If we define variables in the methods call local variable.**
  + **To define values in java, follow below syntax:**
    - DataType VariableName = Value;
    - **The value of variables also called LITERALS.**

**(NOTE: equal to ‘=’ is the assignment operator. It will take the value form right side and assigned to left side in variable.)**

**A screen shot of a computer code

AI-generated content may be incorrect.**

* **Data types in Java:**
  + **There are 2 types of data types in java:**
    - Primitive data type
    - Non- primitive data type
  + **List of Primitive data type:**
    - **Integer**: Stores values without decimal point (0-9)
      * **Subtype of Integer:**
        + **byte**: supports 1 byte.

**Range**: -27 to 27-1 (-128 to 127)

* + - * + **short**: supports 2 bytes.

**Range**: -32,768 to 32,767

* + - * + **int**: supports 4 bytes.

**Range**: -2,147,483,648 to 2,147,483,647

* + - * + **long**: supports 8 bytes.

**Range**:-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

**(Note: While define, need to add “l” at the end of value.** Ex: long l = 12356l;**)**

* + - **Float**: Stores values with decimal point (0.0-9.0)
      * **Subtype of Float:**
        + **float**: supports 4 bytes.

**As Java define double as a default datatype, and we want to define float data type variable then we need to add “*f*” at the end of the value.**

**If we do not add “*f*”, then it will consider as double data type.**

**Ex:** *float fl= 5.6f;*

* + - * + **double**: supports 8 bytes.

**(Note: While defining decimal value variable, by default java supports “double” data type because while doing calculations, double data type store longer digits after decimal point then float.)**

* + - **Character**: Stores only single character (A’, ‘b’)
      * **Subtype of Character:**
        + **char**: supports 2 bytes.
        + **To define a character data type, we need to define value between single quotes (‘H’) because double quotes (“”) are only for String data type.**
        + **Also character data type stores only one single character.**
    - **Boolean**: Stores only either “**True**” or “**False**”
      * **Subtype of Boolean:**
        + **boolean**: supports 1 bit.
        + **In other programming languages, the values of boolean data type is 0 for false and 1 for true but in java the values are “True” or “False”**

**(NOTE: IF WE TRY TO STORE OUT OF RANGE VALES TO THE SMALL DATATYPE THEN IT WILL THROOW AN ERROR. FOR EXAMPLE, AS BYTE STORE (-128 to 127) RANGE AND IF WE STORE “129” IN IT, THEN IT WILL THROW AN ERROR.)**

* + **In integer data type, if we have long numbers, then we can add underscore (\_) between the numbers while defining the values.**
    - **Ex: int num = 10\_00\_00\_000;**
      * **Output = 100000000;**
  + **If we define a alphabet into the character data type then we perform the addition or subtraction operation, it will simply change the alphabet value**
    - **Ex: char c = ‘a’;**

**c++;**

* + - * **Output: b**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* **Data Type Conversion in Java:** *Small -> Large*
  + In java there are two techniques to convert a variable from one data type to another.
    - Type Conversion
    - Type Casting
  + NOTE: We can not perform conversion/casting from char to Boolean datatype, it only supports in the range of numerical conversion between byte to double data type.
  + **Type Conversion:**
    - Convert small data type into large data type without any data loss.
    - It is also known as **Implicit Casting** or **Widening conversion**.
    - Order of implicit conversion:
      * **byte -> short -> int -> long -> float -> double**
    - Example: Converting integer data type to double data type.
    - This type of conversion happens automatically because there is no risk of data loss.

**A computer screen shot of a program

AI-generated content may be incorrect.**

* + **Type Casting:** *Large -> Small*
    - Manually converting a large data type into smaller one.
    - It is also known as **Explicit Casting** or **Narrowing Conversion.**
    - It does not convert automatically because it leads to data loss.
    - Order of Explicit Casting:
      * **double -> float -> long -> int -> short -> byte**
    - Example: converting double data type to integer data type.
    - Syntax:
      * Big DataType Variable = Value;

SmallDataType VariableName = (SmallDataType) Value/BigDataTypeVariable;

**A screen shot of a computer program

AI-generated content may be incorrect.**

**NOTE: Here in this example, we lose 0.99 after converting double to integer value.**

* **Assignment Operators:**
  + List of Operators:
    - Arithmetic Operators
    - Relational Operators
    - Logical Operators
  + **Arithmetic Operators:**
    - **Addition: +**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* + - **Subtraction: -**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + - **Multiplication: \***

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + - **Division: /**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* + - **Modulus/Remainder: %**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + **Relational Operators:**
    - Relational operators mainly compare two values or expressions. And it always returns Boolean result (true or false).
    - We can use assignment operator in IF…ELSE conditions, ITERATION LOOPS(FOR, WHILE, DO…WHILE).
    - **Equal to: ==**

**(NOTE: SINGLE EQUAL TO(=) USE FOR ASSIGNMENT OF VALUE WHILE DOUBLE EQUAL TO(==) USE FOR COMPARITION OF VALUES.)**

* + - **Not Equal to: !=**
    - **Greater than and equal to: > >=**
    - **Less than and equal to: < <=**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + **Logical Operators:**
    - It is mainly used to combine multiple conditions and return a Boolean result (true or false).
    - **Logical AND: &&**
      * When we want all conditions to be true, we need to use the logical AND operator.
    - **Logical OR: ||**
      * When at least one condition among all conditions needs to be true, we use the logical OR operator.
    - **Logical NOT: !**
      * While using NOT operator whatever result we get, it will give opposite of it. For example, if we get the answer TRUE and use the ! OR operator, then it will give FALSE answer.

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Conditional Statements:**
  + Conditional Statements are used to make decisions or execute a block of code based on certain conditions. It executes a specific block of code on whether a condition is true or false.
  + We can add multiple conditions in the single IF statement using logical operators.
    - EX.: if (x > 10 && x < 20)
  + **Types of conditional statements in Java:**
    - **if statement:**
      * There will be only one block of IF condition. If condition will true for that block the it will execute it otherwise it will move forward to other block.
      * Syntax:
        + if (Condition) { …. }

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + - **if…else statement:**
      * There will be only one and IF and one ELSE block of code. If condition true for IF block then it will skip ELSE block and complete the execution.
      * While condition false for IF block, then it will execute the ELSE block and complete the execution.
      * Syntax:
        + if (Condition) {

------------------

} else{

------------------

}

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + - **Ternary Operator:**
      * The Ternary Operator is a short-hand version of if-else statement.
      * It is used to assign a value based on a condition in a single line.
      * Syntax:
        + Variable = (Condition) ? Value\_IF\_True : Value\_IF\_False;
        + Condition = A Boolean expression like a > b, x == y
        + ? = Separates condition from true value
        + Value\_IF\_True = Value assigned if condition is true
        + : = separates true value and false value
        + Value\_IF\_False = Value assigned if condition is false
      * It is only use for simple IF-Else condition and mostly use to assign the values.

A screen shot of a computer program

AI-generated content may be incorrect.

* + - **if…else if…else statement:**
      * If there will be multiple conditions need to check one by one then we can use IF…ELSEIF…ELSE condition block.
      * Syntax:
        + if (Condition) {

------------------

} else if (Condition) {

------------------

} else {

------------------

}

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + - **Nested if statement:**
      * A Nested if Statement means an if statement inside another if statement.
      * It is used when there are **multiple conditions** to check, and one condition depends on another condition.
      * Syntax:
        + if (Condition1) {

if (Condition2){

--------------------

}else{

-------------------

}

} else{

------------------

}

A screen shot of a computer program

AI-generated content may be incorrect.

* + - **Switch statement:**
      * A switch statement is used to execute one block of code among many options based on the value of a variable.
      * It is an **alternative to multiple if-else-if statements** when you are checking the same variable for **multiple constant values**.
      * Syntax:
        + switch (expression/condition){

case value1:

//Code for execute for case value 1

break;

case value2:

//Code for execute for case value 2

break;

case value3:

//Code for execute for case value 3

break;

…. …. …. …. …. …. ….

default:

//Code for execute if none of the case match

break;

}

A screen shot of a computer program

AI-generated content may be incorrect.

* **Looping Statements:**
  + Looping Statements are mainly used to repeat a block of code multiple times. Instead of writing the same code again and again, we can use loops.
  + For example, if we want to print numbers from 1 to 100, using a loop we can done this task withing few statements instead of writing same 100 print statements.
  + **Advantages of Loops:**
    - Reduce code duplication
    - Perform repetitive tasks like printing numbers, processing arrays, or reading input
    - Make programs more efficient and readable
  + **List of Loops in Java:**
    - For Loop
    - While Loop
    - Do…While Loop
    - Enhanced For Loop or For-Each Loop
  + ***for* Loop:**
    - For loop is mainly used when there is a number of iterations are known in advance.
    - For loop syntax first includes initialization, condition and increment/decrement in the single line then loop body will come.
    - **Syntax:**
      * for (initialization; condition; addition/subtraction)

{ //Code to execute }

A screen shot of a computer program

AI-generated content may be incorrect.

* + - For “*for*” Loop, if we want to skip any of the components (initialization, condition or increment/decrement) we can write outside of loop and skip into the for-loop bracket.
    - Syntax:
      * Initialization;

for (; //Condition; ){

//Code to execute

Increment/Decrement;

}

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + ***while* Loop:**
    - While loop is mainly used when the number of iterations are unknown before execution, and the loop should be continued as long as condition is true.
    - In the while loop, condition is checked before executing the loop body.
    - So, it means that to execute the body of the loop, the condition must be fulfilled. Otherwise, it will not execute the loop body and move to the next statement.
    - While loop mainly use to read data from a file or read a data from the database because we don’t know what the exact number of characters file will be contains or don’t know how many lines of data will have in database. So, for this scenarios while loop is the best option.
    - Syntax:
      * Initialization; //If required

while (Condition){

//Code to execute

Increment/Decrement; //if required

}

**A screen shot of a computer

AI-generated content may be incorrect.**

* + ***do-while* Loop:**
    - Do while loop is similar to while loop but one major difference is in do-while Loop, first loop body will execute then check the condition.
    - This means that loop body will execute at least once, even if the condition is false.
    - **Syntax:**
      * Initialization;

do{

//Code to Execute.

} while (Condition);

A screen shot of a computer program

AI-generated content may be incorrect.

* + ***for-each* Loop or Enhanced for loop: (Introduced in Java 8)**
    - The enhanced *for-loop* is a simplified way to iterate through elements in arrays or collections like List, Set, etc., without using an index.
    - It only works with an Array and Array type of data.
    - In the normal loop, we use counter variable (i=1;) but in for-each loop we don’t need counter variable. It will iterate automatically till the last elements of it.
    - **Syntax:**
      * for (datatype variable : arrayName or collectionName)
      * Ex: int[] **numbers** = {10, 20, 30, 40};

// Enhanced for loop

for (int **num** : **numbers**) { SOUT(**num**); }

* + - **Example 1:**

A screen shot of a computer program

AI-generated content may be incorrect.

* + - **Example 2:**

A screenshot of a computer program

AI-generated content may be incorrect.

* + - **Example 3:**
      * We can also use “*for-each”* loop with array class.
      * Follow below syntax:

class ClassName{

datatype variableName;

…

}

class main{

psvm(String args[]){

ClassName classObject = new ClassName();

classObject.classVariable = value;

ClassName classArrayObject[] = new ClassName[ArraySize];

classArrayObject = classObject;

for(ClassName variable : classArrayObject){

SOUT(variable.classVariable);

}

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

* + - **Advantages of for-each loop or Enhanced for-loop:**
      * Clean and simple syntax
      * No risk of ***ArrayIndexOutOfBoundsException***
      * Best for read-only iteration
    - **Limitations:**
      * You **can’t modify elements** (e.g., remove items from a list during iteration)
      * You **don’t have access to the index** of elements
* **Classes and Objects:**
  + Java’s main feature is that Java is the Object-Oriented Programing Language.
  + **Class:**
    - In java developer will create a blueprint or class and JVM(Java Virtual Machine) creates an object of classes.
    - Developer creates class/Blueprint > .class file get compiled and creates a byte code > Byte code goes to JVM where will get the object
    - Class is a collection of elements, methods and variables.
    - Definition:
      * A class is a user-defined data type that acts as a blueprint for creating objects.
    - The class contains the attributes below:
      * Fields/Variables: that holds data
      * Methods: that perform action
    - Syntax:
      * class ClassName {

//Define Fields/Variables

dataType variableName;

//Methods

returnType(void, int, double, etc) methodName() {

//code block

}

}

* + **Object:**
    - Object is an instance of a class, and it is created using the “*new*” keyword.
    - Syntax:
      * ClassName objectName = new ClassName();
    - We can also only define a reference variable of class and later create an object. Follow the syntax below for it:
      * ClassName referenceVariable;

referanceVariable = new ClassName();

A screen shot of a computer program

AI-generated content may be incorrect.

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Polymorphism – Method Overloading and Method Overriding:**
  + Polymorphism is a feature of Object-Oriented Programming that allows one method, class or interface to behave differently based on the object that is calling it.
  + Polymorphism in Java is the ability of an object to take many forms, and it allows the same method or object to behave differently based on the context.
  + **There are two types of Polymorphism:**
    - Method Overloading – Compile time Polymorphism
    - Method Overriding – Run time Polymorphism
  + **Method Overloading – Compile time Polymorphism** 
    - In the overloading method, when there are multiple methods in the **same class have the same method name but different parameters**. (Return type can be different)
    - It occurs within the same class only and also called Compile time polymorphism.
    - Class has multiple methods with the same name but different parameters.
      * Ex. class Addition{

public int add(int a, int b) {return a + b; }

public int add(int a, int b, int c) {return a + b + c; }

}

**A screenshot of a computer program

AI-generated content may be incorrect.**

* + **Method Overriding – Run time Polymorphism** 
    - In the inheritance when the **parent’s class and the child class have same method name with same parameters.**
    - Method overriding occurs when a child class/subclass provides a specific implementation of method that is already defined in its parent class.
    - It only happens in the inheritance.
    - **In method overriding, Method name, Return type and Parameters must be exactly same.**
    - It is also known as Runtime polymorphism and allows dynamic method dispatch.
    - The method in the child class **replaces** the one from the parent class during execution.
    - We can add **“@Override”** annotation above the overridden method in the child class.
    - Whenever we create an object of a child class and call an overridden method, the child class method overrides the parent class method because it has the same name and signature. As a result, the parent method is replaced by the child method at runtime.

A screen shot of a computer program

AI-generated content may be incorrect.

* **Array:**
  + In the variable we can only store single value while in Array we can store multiple values of same datatype.
  + **Definition:**
    - Array is a collection of elements/values of same data type that are stored in a single variable.
  + In Array, we can decide the size of array and we cannot store more values than the size of the array.
  + Array has a fixed size that cannot be changed.
  + It stores the elements of the same data type.
  + The index of array starts form 0 and end with (length-1).
  + To get the size of array then we can use (.length) method.
    - **Syntax:**
      * int nums[] = new int[3];

**int temp = nums.length; //store the size of array in the temp variable**

* + When we define an array with the specific size, but we use only certain index then rest of the empty index have default value “0”. It means that even though we don’t assign the values to empty index, it will still occupy the memory.
    - **Ex:** int num[] = new int[4];

num[0] = 1; num[1] = 2;

In this example, we use only 2 index and assign the value so other 2 empty index will have value 0.

* + **Types of Arrays:**
    - Single Dimension Array
    - Multi Dimension Array
    - Jagged Array
  + **Single Dimension Array:**
    - **Syntax: 1 X 1 array**
      * **When there are values that need to be defined during defining an Array:**
        + DataType variableName[] = { value1, value2, … , valueN};
        + **Ex:** int arr1[] = {1, 2, 3};
      * **When there is only define the size of an Array and defining values during runtime:**
        + DataType variableName[] = new DataType[SizeOfArray];
        + **Ex:** int arr1[] = new int[10];
        + **While we define array with size only then by default all the values will be zero.**
      * **Display individual values using index:**
        + SOUT (“Index Value: ” + arrayVariableName[indexNumber]);
        + Ex.: SOUT (“Index Value: ” + arr1[1]);
      * **Update the value manually with index:**
        + **Syntax:** arrayVariable[indexNumber] = newValue;
        + **Ex:** arr1[3] = 15;

(Note: Accessing the values of array manually is not the correct way to play with it. To work on array, we must use iteration statements or loops. That will reduce the repetitive code as well as faster the execution dynamically.)

A screenshot of a computer program

AI-generated content may be incorrect.

* + Access and display the value of array using ***for*** loop:

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + **Multi Dimension Array:** 
    - **Syntax: 2 X 2 array**
      * **When there are values that need to be defined during defining an Array:**
        + DataType variableName[][] = { { value1, value2, … , valueN}, { value1, value 2, …, valueN} };
        + **Ex:** int arr1[][] = { {1, 2, 3}, {4, 5, 6} };

**A computer screen shot of a program

AI-generated content may be incorrect.**

**(NOTE: EVERYTIME WE WANT TO INCREASE THE DIMENSION, WE HAVE TO ADD EXTRA SQURE BRACKETS WHILE DEFINING THE ARRAY and EXTRA LOOP FOR EVERY BRACKETS.)**

**For 3-dimensional array:**

**int nums[][][] = new int[][][]; //THIS WILL HAVE 3 LOOP TO ITERATE THE VALUES**

* + **Jagged Array:**
    - Jagged array is a type of array where each row has different number of element or different number of columns.
    - For example, 1st row has 3 elements, 2nd row have 5 elements, 3rd row have 1 element.
      * 1 2 3

4 5 6 7 8

9

* + - **Syntax for how to define number columns for individual rows:**
      * **// Define multi-dim array with number of rows only and leave columns size empty**

DataType arrayVariable[][] = new DataType[sizeOfRows][];

**//Define size of columns for each row**

arrayVariable[columnNumber] = new DataType[sizeOfColumn];

* + - * **Example:**

int num[][] = new int [3][];

num[0] = new int[3]; num[1] = new int[4]; num[2] = new int[2];

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + **We can also create an Array of the class, for that follow below steps:**
    - Create a class and define variables in it.
    - Create object of that class.
    - Create array type of object of that class and provide size.
    - Assign object of that class to the index of the array object.
    - **Syntax:**
      * **//Create class**

class ClassName{

//Define Variables

datatype variable1;

datatype variable2;

datatype variable3;

}

**//Main class**

public class Main{

public static void main(String args[]){

**//Create a object and assign values**

ClassName object1 = new ClassName();

object.variable1 = value1;

object.variable2 = value2;

object.variable3 = value3;

**//Create array type of object and assign the class object**

ClassName arrayObject[] = new className[sizeOfArray];

arrayObject[indexNumber] = classObject;

}

}

* + - To access this data, we can simply run for loop and fetch the data. We can use manual way as well but it will be increase the length of the code as well as repetitive code.
  + **Disadvantages of Arrays:**
    - Array is a fixed size, once we define the size of an array we cannot change it.
    - For the searching operation, array travers between the elements from beginning to end all the time, that will consume too much time for searching operation.
    - Array contains the elements of same datatype so it means in single we can not save multiple datatype values.
      * For example: We have defined the Integer type of array and if we try to save String data in it then it will throw an error, and we cannot perform the operation.
    - **TO OVERCOME THE DISADVANTAGE OF AN ARRAY, WE CAN USE *“COLLECTION” INSTEAD OF ARRAY.***
* **String:**
  + If we want to store a bunch of characters or lines then String is the best option.
  + While assigning value to String, we need to add values between double quotes (“AddString”)
  + String in java is not a primitive data type, but it is a class.
  + **Concepts of String class:**
    - Mutable String
    - Immutable String
  + **Immutable String: String cannot be changed**
    - By default, Strings are immutable. Once we create an object, we cannot change it.
    - **Ways to define String variable:**
      * Direct assign a value but it is not a correct way to define it but we can still use it as java handles create an object on backend side:
        + String variable = “String Value”;

Ex. String s = “Hello World”;

* + - * + With this way, behind the scenes, it will create an object automatically

**(NOTE: WHILE WE USE THIS WAY TO DEFINE A STRING, WE CAN NOT UPDATE OR CHANGE THE DATA OF THE STRING VARIABLE.)**

* + - * Defining by creating String class object, it is a correct way as String is a class and not a primitive data type:
        + String objectReferanceVariable= new String(); //Define empty string variable
        + String objectReferanceVariable= new String(“Hello World”); //Define string variable with value
        + Ex. String s = new String();

String s = new String(“Hello World”);

* + - To concat a string in java or merge two strings in one, we must use plus (+) operator. With String, it does not allow to use other operators.
    - If we create multiple object/reference variables with same data, on the behind the scenes, it will not create separate objects or new memory in the heap, but it will just use the address of object of same value. So, while we compare those both objects, we will get “true”. This is how we can save memories with String class.
      * Ex. //Compare two objects of string which have same values

String s3 = "Name";

String s4 = "Name";

System.out.print("Compare two objects of string which have same values: ");

System.out.print(s3 == s4); **//Output: true**

* + - **Example of simple String operations:**



* + **Mutable String: String can be changed**
    - To make string Mutable, we can use following classes from Java to use:
      * String Buffer
      * String Builder
    - **String Buffer Class:**
      * Syntax: StringBuffer objectVariable = new StringBuffer();

StringBuffer objectVariable = new StringBuffer(“Hello Buffer”);

* + - * While we use StringBuffer, it will give us buffer size of 16 bites.
      * To get the size of string buffer, we can use “.capacity()” method. Even though, we assign any value to string buffer object, we will still keep extra 16 bites of space
      * To convert String type to StringBuffer, we need to use “.toString()” method. We cannot directly assign StringBuffer to String.
      * We can use “.deleteCharAt(indexLocatio)” method to delete the character from particular location.
      * String Buffer is thread-safe meaning that it can be modified without creating new objects and safe to use in multi-thread environments.
    - **String Builder Class:**
      * The functionality as well as methods of String Builder and String Buffer class are same but the only key difference is String Builder class is not thread-safe which make it faster and better suited for single-threaded applications.

A screenshot of a computer program

AI-generated content may be incorrect.

* + - **String Buffer Class Example:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

* **Static: Variable**
  + **Static variable means we are making something class member not an object member. So whatever static variables define in the class that belongs to the class not the object.**
  + We can use static variables when there is a common variable that is going to have same value throughout the code.
  + It is used to indicate that a member (variable, method, class or block) belongs to the class itself rather than to the instance of the class. It means that static member is shared among all instances of the class.
    - Example: If we create one static variable in the class, then we create object of that class and change the value with that object. After that that value will remain the same for all of the objects that we are going to create.

class Mobile {

int id;

static String type;

Public void display(){ SOUT(“[ ” + id + “ : “ + type + “ ]”);

}

class Main{

PSVM(String args[]){

Mobile obj1= new Mobile();

obj1.id = 1;

obj1.type = “Smartphone”;

obj1.display(); **//Output: [ 1 : Smartphone ]**

Mobile obj2= new Mobile();

obj2.id = 2;

obj2.type = “Smartphone”;

obj2.display(); **//Output: [ 2 : Smartphone ]**

SOUT(Mobile.type); **//Output: Smartphone**

}

}

In above example, we have created class and define static variable “type”. When we have created an object of Mobile class and assigned “***type=Smartphone***” in one object, it will remain same until we change it and also shared among all objects/instances of the class.

* + We can access static variables both way by creating objects of the class (m1.type) and without creating object through class name and variable name(Mobile.type). **(The correct way is access through class name and variable name.)**
  + **In the non-static method also we can use static variables.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Static: Method**
  + Same as static variables, static method also belongs to the class and not to any object member.
  + Static method can be called without creating an object, but object/instance methods cannot be accessed directly. We need to create an object for it.
  + **In non-static method we can use static variables but in static method we cannot use static variables. For that we need to pass the object of the class in the parameters. (See Example 2)**
  + To call a static method, we can directly call it directly with class name and method name.
    - Ex. Mobile.display();
  + **Syntax:**
    - class ClassName {

static returnType methodName(parameters) {

// method body

}

}

**Example 1:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Example 2:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Static: Block**
  + We can also initialize a value in a static variable in the default constructor, but the thing is when we change the static variable value during the code execution, and we again create an object, it will automatically re-initialize the values which we have assigned in the default constructor. To overcome this issue, we can create a static block.
  + Static block will call only once, it does not matter how many objects we create.
  + **Syntax:**
    - class ClassName{

datatType variableName1;

dataType variableName2;

static dataType variableName3;

// Static block

static{

dataType variableName;

staticVariableName = value;

}

}

* + Whenever there is static block and constructor define into the same package/file, static block will execute first then default constructor will execute every time object will create.
  + Whenever we execute a Java program, the **class is loaded first**, and then the **objects are created**.
  + In the JVM, there is a special component called the **Class Loader**, which is responsible for **loading classes into memory**. This class loading process happens **only once per class**, during the program’s execution.
  + When a class is loaded:
    - All **static blocks**, **static variables**, and **static methods** are initialized or executed.
    - Since class loading occurs only once, the **static block is also executed only once**, regardless of how many objects are created later.
  + After the class is loaded and static members are initialized, **object creation** begins, and the **constructor** is called for each object.
  + If we don’t create an object then static block also not called. But if we still want to execute the static block then use below syntax in main method. This will simply load a class:
    - Syntax: Class.forName(“className”);
    - Ex.: Class.forName(“Mobile”);

**A computer screen shot of a program code

AI-generated content may be incorrect.**

A computer screen with white text

AI-generated content may be incorrect.

* **Static: Main Method (public static void main (String args[])):**
  + The main() method in Java is defined as static because it serves as the entry point for the program, and the JVM needs to call it without creating an object of the class. Since static methods belong to the class itself rather than an instance, the JVM can directly execute main() when the program starts.
  + If the main() method is defined as non-static, the JVM will not be able to call it directly, resulting in a runtime error, because it would require creating an object first, and the JVM has no way to do that automatically before the program begins. Therefore, main() must be static for the program to run successfully.
  + **Main method : public** 
    - It means the method is accessible from anywhere.
    - Java’s runtime environment (JVM) needs to call this method from outside the class, so it must be public.
  + **Main method: static**
    - It means no need to create an object of the class to call the main method.
    - When the program starts, no objects exist yet — the JVM needs a method it can run without instantiating the class.
    - Making it static lets the JVM directly call main() using the class name.

**(NOTE: If main wasn't static, the JVM would have to create an object of your class to start the program — which could be impossible or unnecessary if your class requires complex setup or has no constructor.)**

* + **Main method: void**
    - It means the method doesn’t return anything.
    - The main goal of main is to run the application, not return a value.
  + **Main method: main**
    - This is the name the JVM looks for as the starting point of execution.
  + **Main method: (String args[])**
    - This allows command-line arguments to be passed to your program.

A screenshot of a computer program

AI-generated content may be incorrect.

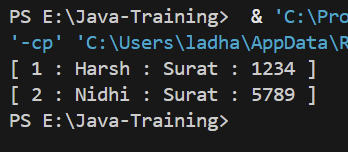
* **Encapsulation: getter() setter() methods**
  + It is a part of OPPS.
  + It is wrapping the data (variables) and the code (methods) that work on the data into single unit, typically a class, and restricting access to some of the object’s component.
  + Encapsulation is to make sure that "sensitive" data is hidden from users.
  + **Encapsulation is like a protective capsule around your data.** You decide who can see it and who can change it.
  + **To achieve this, follow below instruction:**
    - Need to make all the variables *“private” –* so no one can’t access it directly from outside of class.
    - Define *getter()* and *setter()* public methods – that to read and update the value.
      * setter() method – use to assign the value
      * getter() method – use to display – access the value

**(NOTE: WE CAN USE DIFFERENT NAME INSTEAD OF getter and setter, but it will be hard to recognize which method used for encapsulation and which are simple method)**

* + **In Encapsulation, to access the private variables, we need to access through methods using object.**
  + **Benefits of Encapsulation:**
    - Data hiding – prevents direct access to fields.
    - Control – you can add logic inside setters/getters (e.g., validation).
    - Easy maintenance – internal code changes don't affect external code.
    - Improved security – restricts how important data is accessed or changed.

A screen shot of a computer program

AI-generated content may be incorrect.



* **“*this*” keyword: In variable**
  + When an instance variable of a class and a local variable within a method of the same class have the same variable name, the ***“this”*** keyword is used with the instance variable to distinguish it from the local variable.
  + ***“this”***  is the keyword that represents the current object. Current object mean the object which is calling the method.

A screen shot of a computer program

AI-generated content may be incorrect.

* **“this()” keyword: In method / constructor:**
  + When we have child class and parent class with constructors implemented in it and we know whenever we will create an object of a child class it will call parents constructor first and then call child class constructor but if we want to prevent this and only want to call a child class constructor, we can use this() in our child class.
  + It is execute a constructor of same class.

A screenshot of a computer program

AI-generated content may be incorrect.

* **Constructor:**
  + A constructor is a special method with the same name as a class and will call when the object got created.
  + One class has multiple parameterized constructors but it only have one default constructor.
  + In real time, whenever we want to assign some values, always use a method and the constructor is the best option. That is the code standard.
  + **Characteristics of a Constructor:**
    - It has same name as the class name.
    - It does not have any return type.
    - It will call automatically called when the class object got created.
    - **Constructor can be overloaded mean can have multiple constructors created with different parameters.**
    - It is mainly used to assign a initial values. (In default type of constructor, with no parameters.)
  + **Syntax:**
    - class ClassName{

//Constructor Creation

ClassName(){

// Code body

}

}

* + **Types of constructors:**
    - Default Constructor
    - Parameterized Constructor
  + **Default Constructor:**
    - Default Constructor does not have any parameters.
    - If there is no constructor defined, still java defined default constructor automatically on backend.
    - Whenever object is created without any parameters, default constructor automatically called.
    - **Syntax:**
      * class ClassName{

className(){

// code body

}

}

A computer screen shot of a program code

AI-generated content may be incorrect.

* + **Parameterized Constructor:**
    - When we defined constructor by passing some parameters, it is called parameterized constructor.
    - In the parameterized constructor, while creating an object, we need to pass the parameters as we defined it.
    - It is used when we want to provide custom values during object creation.
    - **Syntax:** 
      * class ClassName{

// Define Parameterized constructor

className(dataType variable1, dataType variable2){

// Code body

}

}

class Main{

PSVM(String args[]){

ClassName obj1 = new ClassName(value1, value 2);

}

}

**A screen shot of a computer program

AI-generated content may be incorrect.**

* ***“super()”* Keyword:**
  + ***“super()”*** means call the constructor of a super class.
  + ***With super() if we do not pass any parameter then it will call only default constructor but if we pass any parameter, it will call parameterized constructor.***
  + In java, ***“super()”*** keyword is used to call the methods of parents from child class.
  + While we create a default constructor in both child and parent class and create an object of child class, first it will execute the constructor from parents’ class then will execute constructor from child class both one after another.
  + **When the parent class has a parameterized constructor and the child class also defines a constructor with the same parameters, creating an object of the child class will not automatically call the parent class constructor. In such cases, the *“super()”* keyword must be used in the child class constructor to explicitly call the parent class constructor with parameters.**
  + **Also, by default every constructor’s first statement is “super()” and if we do not mention, java will add it in backend.**
  + **It will execute a constructor of super / parent class.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A computer screen with white text

AI-generated content may be incorrect.**

* **Naming Convention:**
  + In java, basically we follow Camel casing.
  + **Class and Interface:**
    - Class and Interface name will start with a Capital letter and if there is more than one word there will no space between words and all first letters should be capital letter.
      * Ex. class HelloWorld { … }
      * class Calc { … }
  + **Variable and Method:**
    - Variable and method name should be in small letters and if there are more than one words in the name then we need to keep first word should be in small letter and keep next all word’s first letter in capital letters without any spaces between words.
      * Ex. int id;
      * String studentAddress;
      * public void display() { … }
      * public void getAddress() { … }
  + **Constants Variable:**
    - Constants variable which’s values are not going to be changed, will be define in upper case for all the characters of variable name. If there are more than one words in the variable name then we can use snack case and put underscore (\_) between words and keep all characters in the upper case.
      * Ex. double PI;
      * String BRAND;
      * int MY\_DATA;