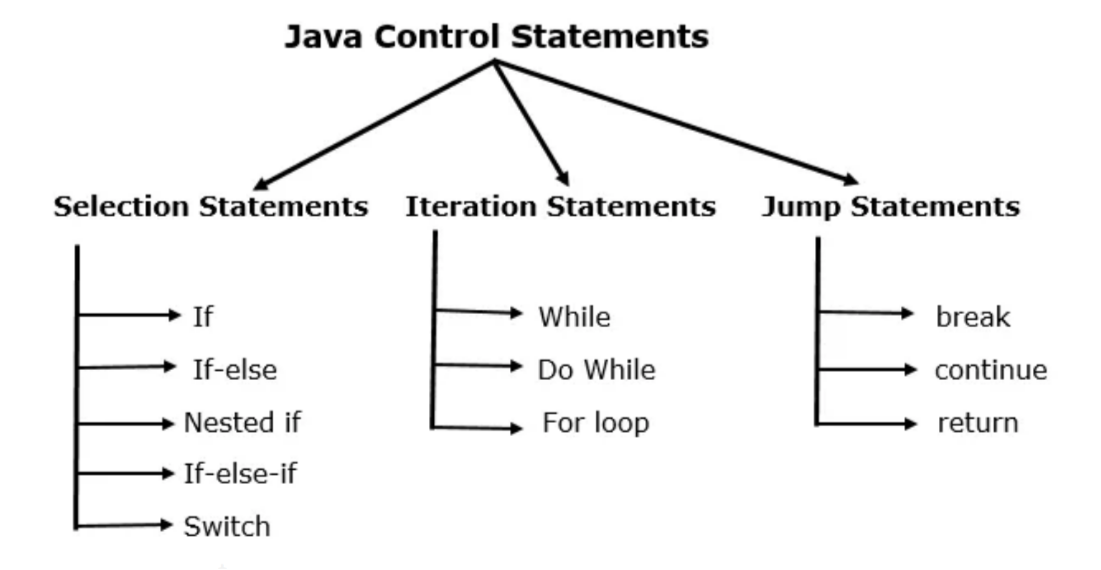
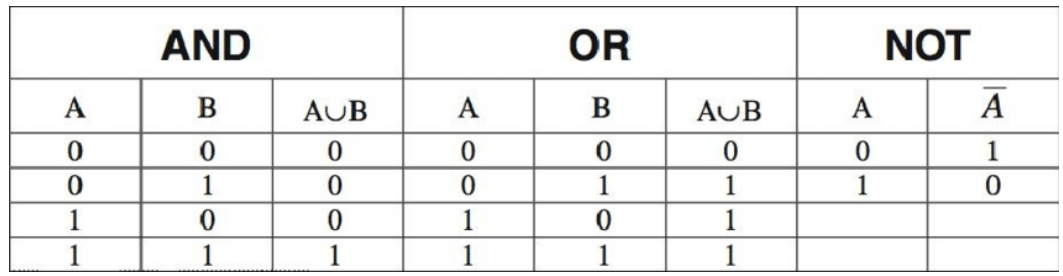
<https://medium.com/@SameerBramhecha/java-jumpstart-mastering-control-statements-1fafb5c4766b>

<https://www.geeksforgeeks.org/decision-making-javaif-else-switch-break-continue-jump/>





**AND --> && (double AND operator)** 🡪 If all conditions are true then only result will be true else result will be false.

**OR --> || (double pipes**)🡪 If any one conditions is true then only result will be true else result will be false.

We’ll explore three main types of control statements in Java:

**selection statements, iteration statements, and jump statements**. Selection statements allow your program to make decisions based on conditions. Iteration statements enable repetitive execution of a block of code until a condition is met. Jump statements provide more control over the flow of execution within loops and other control structures.

Java supports two selection statements: **if** and **switch.**These statements allow you to control the flow of your program’s execution based upon conditions known only during runtime.

**The ‘*if’*statement**

|  |
| --- |
| if(condition){  statement1; } else{  statement2; } |

**Nested ifs:**

An if statement inside another if statement.

|  |
| --- |
| /\*  \* This source file was generated by the Gradle 'init' task  \*/  package org.example;  public class App {  public static void main(String[] args) {  String userName = "kamal shaik";  String password = "abc123";  **if (!userName.isEmpty() && !password.isEmpty()) {** // ! ex cla mation --> NOT  System.out.println("Given value is not empty...");  **if (userName.length() > 8 && userName.length() < 15) {**  System.out.println("Strong password!");  **} else {**  System.out.println("Weak password!!!!");  System.out.println("Password length should be between 8 to 15 characters!");  }  **} else {**  System.out.println("The email address or mobile number you entered isn't connected to an account. Find your account and log in.");  }  }  } |

**If-else-if** ladder**:**

|  |
| --- |
| String name = "my name is Kamal"; **if (name.isEmpty()) {**  System.*out*.println("Name is empty!, Please enter you Name!!"); **} else if (name.length() < 5) {**  System.*out*.println("Name is lessthen 5 characters!!!"); **} else if ((name.length() > 5 && name.length() < 10) || (name.length() > 10 && name.length() < 20)) {**  System.*out*.println("Name is greaterthen 10 characters and lessthen 20 characters!!!");  System.*out*.println("Name is greaterthen 5 characters and lessthen 10 characters!!!"); **} else {**  System.*out*.println("I am DONE!"); } |

**Switch Statement:**

Instead of using many *if-else-if*statements, the *switch* statement will be used.

In case of if-else-if ladder all conditions will be checked line by line, if condition is true then only statements will be executed. If you have 100 if else if statements all 100 conditions are checked one by one. It is time consuming process, here unnecessarily we are checking all conditions.

To over come above issue they have introduced “SWITCH” statement. In case of switch statement, it will check for all the conditions, instead it will directly **jump to matching CASE**.

|  |
| --- |
| package com.skh; import java.io.FileReader;import java.util.Scanner;import java.util.concurrent.\*;import java.util.stream.IntStream; public class App {  public static void main(String[] args) {  int inputValue = 2;  switch(inputValue){  case 1:  System.*out*.println("You choose Menu Option #1");  System.*out*.println("You choose Menu Option #1");  System.*out*.println("You choose Menu Option #1");  System.*out*.println("You choose Menu Option #1");  System.*out*.println("You choose Menu Option #1");  System.*out*.println("You choose Menu Option #1");  break;  case 2:  System.*out*.println("You choose Menu Option #2");  break;  case 3:  System.*out*.println("You choose Menu Option #3");  break;  default:  System.*out*.println("You choose an unavailable option");  }  System.*out*.println("End of the program..!!!");  } } |

|  |
| --- |
| String name = "Azad"; switch (name){  case "Kamal" :  System.*out*.println("Hi this is " + name);  break;  case "Arafath":  System.*out*.println("Hi this is " + name);  break;  case "Azad":  System.*out*.println("Hi this is " + name);  break;  default:  System.*out*.println("Invalid name entered!"); } |

**Java Iteration Statement:**

Iteration Statements, or what we generally call as loops, allow us to execute a block of code repeatedly as long as a condition is met/satisfy. There are three main types of loops in Java: ***‘while’, ‘do-while’ and ‘for’***loop.

***for* Loop:**

|  |
| --- |
| *for (int index = 10; index <= 40 ; index++){*  */\*  1.* ***initialization*** *it is starting position.  2.* ***condition*** *--> till what level loop should iterate.  3.* ***increment or decrement*** *\*/* System.*out*.println(index);  } |

***For loop Execution flow:*** *step 1. intialization + condition checking = true --> go inside loop execute statements.*

*This part runs only once at the beginning of the loop.  
 step 2.It will go to increment / decrement section. + it will check for condition.  
 if condition is true it will execute inside statements.  
 step 3.It will go to increment / decrement section. + it will check for condition.  
 if condition is true it will execute inside statements.  
 step 4. when ever condition fails it will exit from loop.*

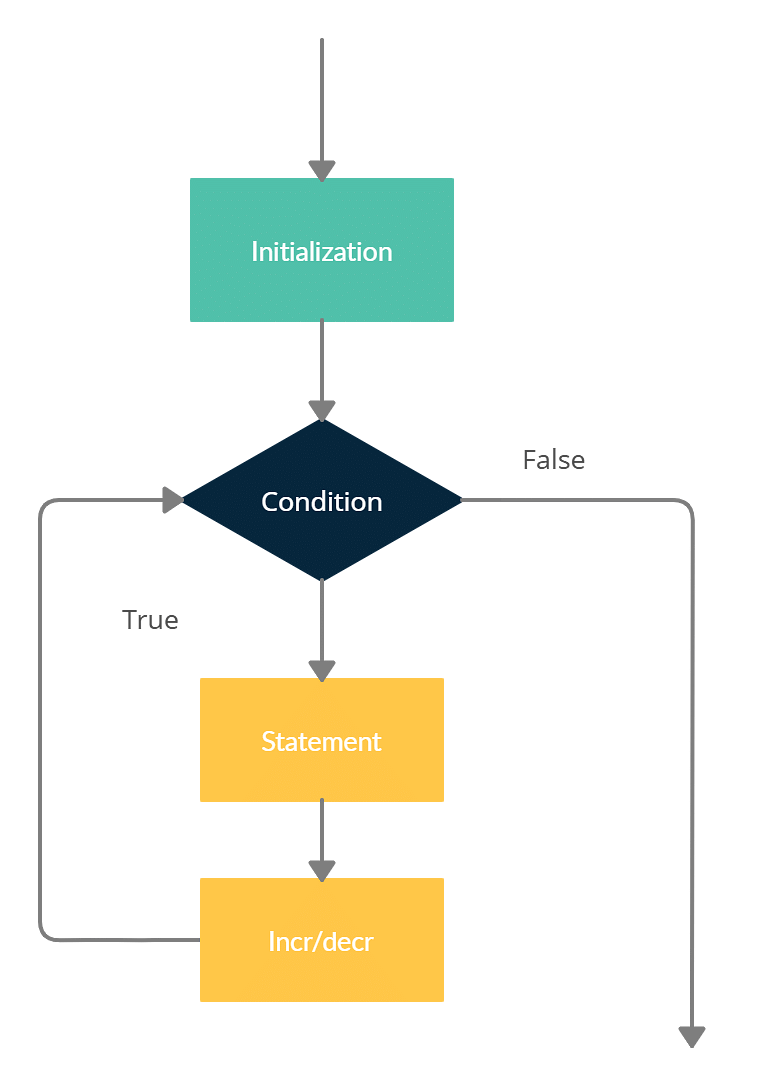
**Infinite loop:**

If we don’t keep initialization section and condition section and increment or decrement section properly, it may fall under infinite loop.

In infinite loop it will always execute inside loop only, it will not come out of loop. In this case as loop running continuously it will occupy your system RAM and system will get hang.

|  |
| --- |
| **for (; ; ) { *// infinite loop***System.*out*.println("Hi Azad...!"); }  **for (int i = 0; ; i++) { *// infinite loop***System.*out*.println("Hi Azad...!"); } |

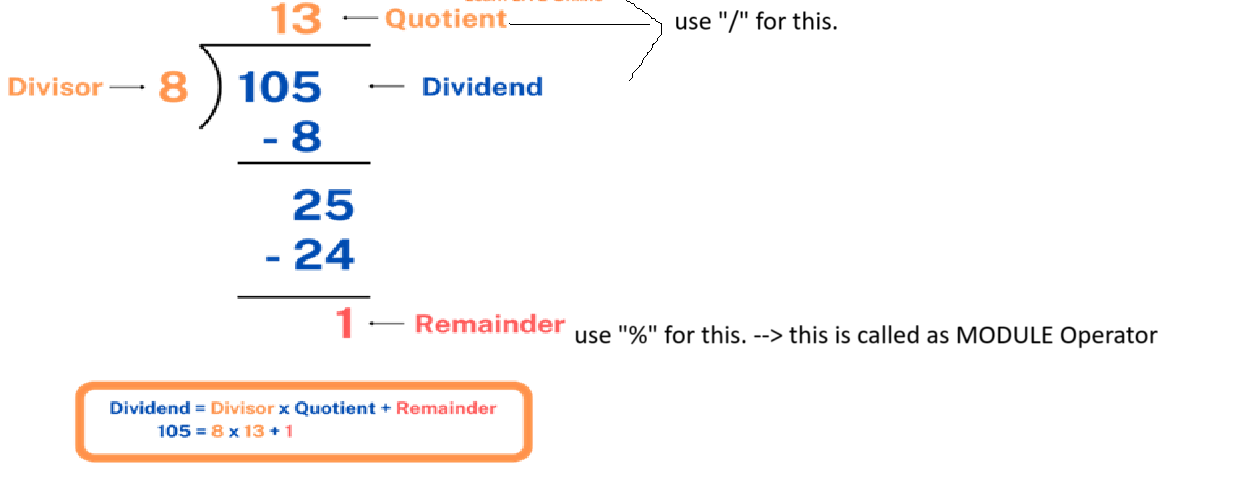
We need to create while creating loops.



<https://www.programiz.com/java-programming/for-loop>

**Stars Program:**

|  |
| --- |
| for (int x = 1; x <= 7; x++) {  for (int y = 1; y <= x; y++) {  System.out.print("\* ");  }  System.out.println();  } |



**Program:**

**PRIME numbers program.**

[Java Program to Check Whether a Number is Prime or Not](https://www.programiz.com/java-programming/examples/prime-number)

**While loop:**

Syntax:

|  |
| --- |
| while (condition){  // body of the loop } |

Example:

|  |  |
| --- | --- |
| int i = 0; *// variable declaration / initialization.* while (i < 10) { *// condition checking* System.*out*.print(i + " ");  i++; *// increment / decrement* } | For loop:  for(int i = 0; i < 10; i++){  System.*out*.print(i + " ");  } |

While loop also we write like for loop only, but syntax representation changes.

**“While” means అయితే.**

If condition is true it will go inside the loop, then increment happens, then again it checks the condition again it will go inside loop, until condition fails this loop repeats.

String[] strArr = {"Azad", "Arafath", "Kamal**"};**

***Array 🡪 Collection of Similar data type values.***

**Why should we go for arrays?**

Different types of Arrays:

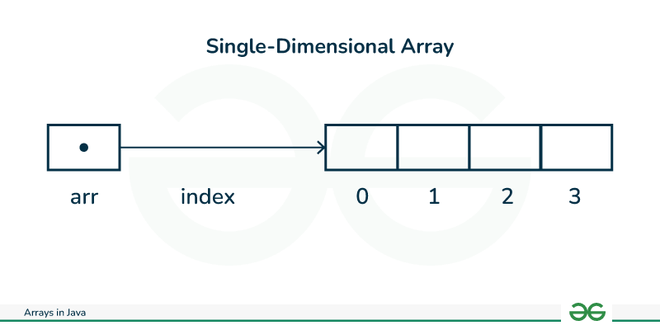
|  |
| --- |
| String[] namesArr = {"AAA", "BBBB", "cccc","Arafath", "Azad", "Kamal"};  **int**[] intArr = {32,34,4534,23,1};  **double**[] doubleArr = {32,34,4534,23,1};  **float**[] fArr = {343.223f,545.45f,23233f,5656f};  **char**[] charArr = {'c','A','V','K','P'};  // custom classes.  Employee[] empArr = {**new** Employee(), **new** Employee(), **new** Employee()}; |

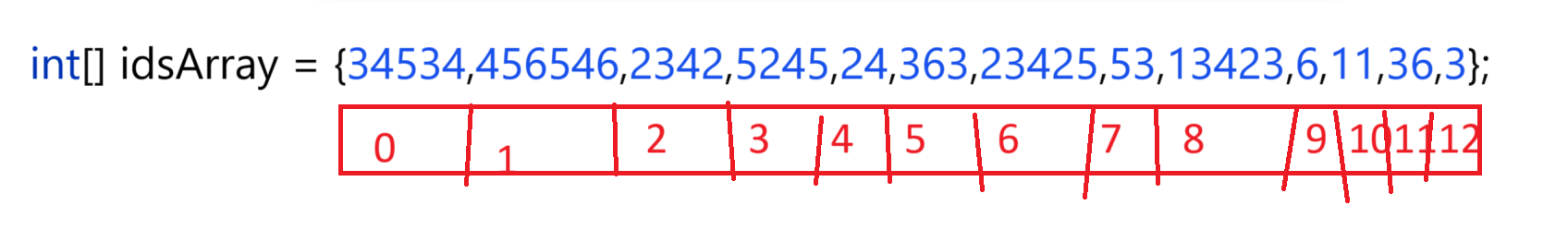
In Array every value has one index value. INDEX values always starts from “0”,1,2,3,4,5…etc incremental manner.

**Why INDEX?**

If you want to iterate or fetch values from an array we need to make use of INDEX’s only.

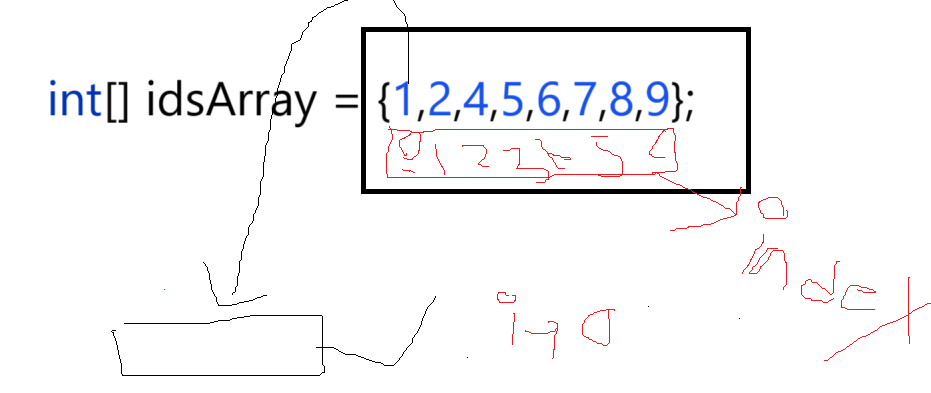
IndexValue = arrayValue -1; 🡪 N-1. 🡪 array.length-1 🡪 you will get LAST value in the array.





|  |
| --- |
| int**[]** idsArray = {1111,456546,2342,3433,4444,5555,66666,777774,2232,45664,53,13423,6,11,36,3,23,34,12,4};  System.*out*.println("Array Length : "+idsArray.length);  **for (int i = 0; i < idsArray.length; i++) {  System.*out*.println(idsArray[i]); }** |

|  |
| --- |
| int**[]** idsArray = {1111, 456546, 2342, 3433, 4444, 5555, 66666, 777774, 2232, 45664,  5245, 24, 363, 23425, 53, 13423, 6, 11, 36, 3, 23, 34, 12, 4};  System.*out*.println("Array Length : " + idsArray.length);  **int i = 0; while (i < idsArray.length) {  System.*out*.println(idsArray[i]);  i++; }** |



**Jump Statements:**

1. **break**
2. **continue**
3. **return**

**‘*break*’ statement:**

The break statement is used to exit from a loop or switch statement prematurely. When the ‘*break*’ statement is encountered, the control flow jumps to the statement immediately following the loop or switch.

|  |
| --- |
| **int**[] namesArr = { 12, 23, 2, 34, 232, 566, 34, 23, 121, 434, 23233434 };  System.***out***.println("array length : " + namesArr.length);  **for** (**int** i = 0; i <= namesArr.length - 1; i++) {  **if** (namesArr[i] > 500) {  **break**;  }  System.***out***.println(namesArr[i]);  } |

**‘*continue*’ statement:**

The ‘*continue*’ statement is used to skip the current iteration of a loop and proceed to the next iteration. It can be particularly useful when you need to skip certain values in a loop.

|  |
| --- |
| **int**[] namesArr = { 12, 23, 2, 34, 232, 566, 34, 23, 121, 434, 676 };  System.***out***.println("array length : " + **namesArr.length**);  **for** (**int** i = 0; i <= namesArr.length - 1; i++) {  **if** (namesArr[i] > 500) {  **continue**;  }  System.***out***.println(namesArr[i]);  // 100 LINES OF CODE.  } |

|  |
| --- |
| public class App {  public static void main(String[] args) throws ExecutionException, InterruptedException {  String[] namesArr = {"Azad", "karthik", "Arafath", "Prabhas", "emp-kamal", "emp-ravi", "FFFFFFFFF"};  int namesArrLength = namesArr.length;  for(int i = 0; i <= namesArrLength-1; i++){  if(namesArr[i].startsWith("emp")){  continue;  *// break* }  System.*out*.println(namesArr[i]);  }  } } |

***Arrays are fixed in size. Length is FIXED. Once array created we can’t increase size of the array.***

|  |
| --- |
| String[] namesArr = {"Arafath", "Karthik", "Azad", "Kamal"}; namesArr[3] = "SKH"; namesArr[4] = "RAM"; *// "=" is Assignment operator. // "==" is an equality comparator.* for (int i = 0; i <= namesArr.length -1 ; i++) {  System.*out*.println(namesArr[i]); } *// Arrays are fixed in size. Length is FIXED.* |

If we try to add more elements in array greater then existing size, then it will throw Exception.

That execption is “**ArrayIndexOutOfBoundException**”



**How many ways we can create Arrays?**

**Approach 1:** Int[] arr = {12,34,454,5656};

**Approach 2:** int[] arr = new int[**50]; / creating the array with size and latter adding data.**

In approach1 directly we are assigning the values, but approach 1 is having some drawbacks.

We can not add more elements then existing elements, why because **Arrays size is FIXED**. If we add more elements, it will throw Exception.

To over come this problem, we are going to work with second approach.

In the second approach, we declare array with required size/ length. So we can store values in arrays up to added length.

In this approach you should know how many values are you are going to add in the array in future.

When we create an array with predefined size, default values for empty array will be based on Datatype you have taken.

For eg: **int** a = new int[50]; 🡪 here we will be having **default values 0’s** for all 50 indexes.

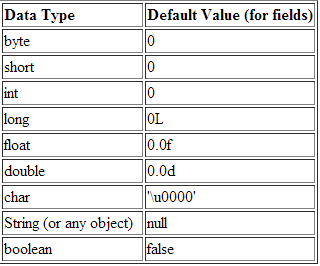
For eg: **boolean** a = new boolean[50]; 🡪 here we will be having **default values false’s** for all 50 indexes.

For eg: **float** a = new float[50]; 🡪 here we will be having default **values 0.0** for all 50 indexes.

For eg: **String** a = new String[50]; 🡪 here we will be having **default values “null”** for all 50 indexes.

**In Java all classes default value is null only.**

**Default values for all PRIMITIVE DATA TYPES.:**



Every primitive datatype has equal **Wrapper class**.

For example:

int 🡪 Integer

byte🡪 Byte

short 🡪 Short

float 🡪 Float

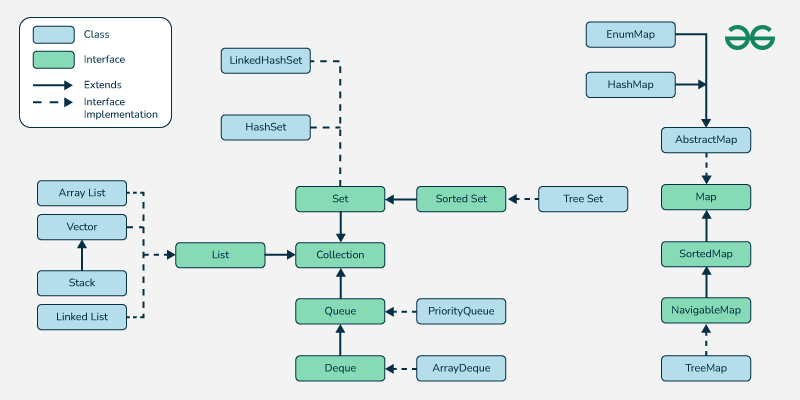
double 🡪 Double

boolean 🡪 Boolean

long 🡪 Long

Left side are primitive data types, right side are wrapper classes. We know that class always star with Upper case letter.

To overcome problems with arrays, Java people has introduced “**Collection of Framework**”.



How to assign values to empty array?

We assign values based on index.

|  |
| --- |
| String[] apDistricts = new String[10]; **apDistricts[0] = "Azad"; apDistricts[10] = "Kamal";**  for (int i = 0; i <= apDistricts.length -1 ; i++){  System.*out*.println(apDistricts[i]); }  String fName = apDistricts[0]; System.*out*.println(fName); |

**Program:**

**Find the smallest / minimum value in the array?**

|  |
| --- |
| int[] my\_array = {7, 12, 9, 4, 11};  int minVal = my\_array[0];  for (int i : my\_array) {  if (i < minVal) {  minVal = i;  }  }  System.out.println("Lowest value: " + minVal); |

What is the difference between traditional index for loop and enhanced for loop?

|  |
| --- |
| package org.example.corejava;  public class CoreJavaTest {   public void welcome(){  System.*out*.println("Welcome to Core Java");  }   public void loopArrays(String[] namesArr) {  for (String name : namesArr) {  System.*out*.println(name);  }  }   public void isMale(boolean isMale) {  if (isMale == true) {  System.*out*.println("I am a MAN");  } else {  System.*out*.println("I am a Women");  }  }   public void myDetails(String name, boolean isMale, int age) {  System.*out*.println("My name is " + name);  System.*out*.println("My age is " + age);  if (isMale == true) {  System.*out*.println("I am a MAN");  } else {  System.*out*.println("I am a Women");  }  }   public void findMinValInArray(int[] myArr){  int minValue = myArr[0];  for (int val : myArr){  if(val < minValue){  minValue = val;  }  }  System.*out*.println("Min value is: " + minValue);  }  } |

We should not pass more and more parameters to method. Instead of sending those many parameters,

**create a separate class and** **declare all those parameters in separate class and pass that class as parameter to method**.

What types of Classes we have in java?

1. Classes with business logic.
2. **Class for storing the data and transferring the data**. (**POJO**-**P**lain **O**ld **J**ava **O**bject **class**)

Classes with business logic means that class have method with some logic.

**Pojo class:** a class which doent have business logic and it should contain only

1. **Private instance variables**
2. OPTIONAL: Public default OR parameter constructor.
3. Public setter methods and getters methods.
4. OPTIONAL: toString() method.

How to create pojo class?

What is the use of setter methods and getter methods and constructors in pojo class?

**Constructors** are used to **store data in instance variables**.

**Setter** methods are used to **store data in instance variables**.

**Getter** methods are used to **get the data from instance variable**s.

**toString**() method is used **to display the data** in instance variables in **style/pretty format**.

What will happen if we don’t write toString() method in pojo class? 🡪 by default toString() method prints object hashcode OR address of object.

Is it mandatory to write toString() method? –NO, only for testing purpose we use toString() method.

|  |  |
| --- | --- |
| package org.example.corejava;  public class StudentMarks { **// instance variables.**  private String studentName;  private Integer mathsMarks;  private Integer physicsMarks;  private Integer chemistryMarks;  private Integer teluguMaraks;  private Integer englishMarks;  private Integer hindiMarks;  private Integer envSceince;  **// setter methods and getter methods**  **// toString() method**  } | public class App {  public static void main(String[] args) {  **StudentMarks studentMarks = new StudentMarks();**  studentMarks.setStudentName("Arafath");  studentMarks.setChemistryMarks(45);  studentMarks.setPhysicsMarks(89);  studentMarks.setTeluguMaraks(34);  studentMarks.setMathsMarks(89);  studentMarks.setEnglishMarks(67);  studentMarks.setEnvSceince(45);  studentMarks.setHindiMarks(45);   **Calculator c = new Calculator();**  c.calculatePercentage(studentMarks);  } } |