Java Training

Table of contents

Encapsulation

Programming Languages

- **C** : procedural programming language
- **C++** : procedural and object oriented programming language
- **Java** : Object oriented programming language

  It is a portable and secure programming language.
- **Python** : Object oriented programming language

Classes and objects

Class : A classs is a template or a blueprint from which objects are created

Object : An object is an instance of a class. Its a physical entity that has the state and behaviour defined by its class. It is a thing that follows the structural blueprint i.e a class. A class can have multiple objects.

Examples:

- the class animal has cats and dogs as objects
- class: humans, objects: safwan
- class: chocolate, objects: cadbury and hersheys.

Running Java code

Type the program in notepad or any other text editor

1. **Save** : with an extension .java
2. **Compile** : type javac filename.java in the command prompt
3. **Run** : java classname

```
class Titanic{
  public static void main(string rose[]){
    System.out.println("welcome to the titanic");
  }
}
```

save the file as titanic.java

open command prompt

go to the directory of the file

javac titanic.java

java Titanic

System is a predifined class, out is an object and println is a method. Methods which are present inside a class will be accessed using objects.

Any line that starts with // in java is called a single line **comment** and it will be ignored by the compiler while running the code. Likewise, /* */ is the syntax for multi-line comments.

practice program

```
class HelloWorld{
  public static void main(string[] args){
    //TYTYTYTYTYTY
    System.out.println("**********");
    System.out.println("*        *");
    System.out.println("*        *");
    System.out.println("*        *");
    System.out.println("*        *");
    System.out.println("**********");
  }
}
```

prints out a hollow box on the command prompt.

datatype program

```
class Datatypes{
  public static void main(String[] args){
    byte a = 100;
    short b = 120;
    long l = 454545454545L;
    float c = 12.1123145f;
    double d = 14.787878787878d;
    char name = 't';

    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
    System.out.println(l);
    System.out.println(d);
    System.out.println(name);
  }
}
```

writing a program to add to integer numbers

```
class AddingNumbers{
   public static void main(String[] args){
    int Number1 = 20;
    int Number2 = 30;

    int SUM = Number1 + Number2;

    System.out.println(SUM);
  }
}
```

Q. Write a program to multiply two numbers

```
System.out.println(25 * 360);
```

Q. Write a program assigning two numbers in variables x and y, divide y with x and print quotient and remainder. Calculating remainder using % is called mod or modulus operator.

```
class Calcuting{
  public static void main(string[] args){
    int x = 10;
    int y = 2;

    System.out.println(x/y);
    System.out.println(x%y);
  }
}
```

Q. Do a complete program for all the arithmetic operators.

```
public class Calculating {
    public static void main(String[] args){
        int x = 10;
        int y = 2;

        System.out.println(x+y);
        System.out.println(x-y);
        System.out.println(x*y);
        System.out.println(x/y);
        System.out.println(x%y);
    }
}
```

Datatypes

Byte

1 byte size. Range: -128 to 127

byte variableName = 789;

whole numbers.

Short

2 bytes size. Range: -32768 to 32767

short variableName = 20784;

whole numbers.

Int

4 bytes size. Range: -2147483648 to 2147483647

int variableName = 347989;

whole numbers.

Long

8 bytes size. Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

long variableName = 745637875783784L;

whole numbers.

Float

Double

4 bytes size. 6 to 7 decimal digits

float variableName = 46357.783f;

8 bytes size. 15 decimal digits.

double varName = 57.76763d;

Boolean

1 bit size. stores true or false.

Char

2 bytes size. ASCII characters

String

String variableName = new String “Hello World”;

String variableName2 = “Hello World”;

String has some in- built functions:

```
String s = new String("hello");
String s2 = new String("world");

s.charAT(2); // prints the character at index
s.length();  // prints the length of the string
s.indexOf("o");  // prints the index of given character in the string
s.toUpperCase();  // prints the string in all caps
s.toLowerCase();  // prints the string in all lowercase letters
s.compareTo("hell");  //
s.concat(s2);  // inbuilt function to concatenate two strings
```

Operators

Arithmetic Operators

- addition +
- subtraction -
- multiplication *
- division quotient /
- division remainder %

Assignment Operators

- addition assignment +=
- subtraction assignment -=
- multipliction assignment *=
- division assignment /=
- modulus assignment %=

Relational Operators

- equal to ==
- less than <
- less than or equal to <=
- greater than >
- greater than or equal to >=
- not equal to !=

Logical Operators

- and &&

  both sides should be true
- or ||

  atleast one should be true
- not !

  !(false) = true

Unary Operators

- Increment operator
  - **pre-increment ++x**

    first the value is incresed by one and then it is assigned.
  - **post-increment x++**

    first the value gets assigned and then increased by one.
- Decrement operator
  - **pre-decrement - -x**

    first the value is decreased by one and then it is assigned.
  - **post-decrement x- -**

    first the value is assigned and then decreased by one.

Bitwise Operators

conversion from decimal to binary and vice versa:

to convert from decimal to binary, fractionise and take the remainders as binary bits.

To convert back to decimal, multiply respective bits to 32 16 8 4 2 1.

Convert 7, 4, 9 : 7 - 0111, 4 - 0100, 9 - 1001

> ### Number conversions

⇒ Binary : base 2 : 0 and 1

⇒ Decimal : base 10 : 0 to 9

⇒ Octal : base 8 : 0 to 7

⇒ Hexadecimal : base 16 : 0 to 15

Types of bitwise operators

- & and operator

  0 0 = 0

  1 0 = 0

  0 1 = 0

  1 1 = 1

```
a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise AND Operation of 5 and 7
  0101
& 0111
 ________
  0101  = 5 (In decimal)
```

- | or operator

  0 0 = 0

  1 0 = 1

  0 1 = 1

  1 1 = 1

```
a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise OR Operation of 5 and 7
  0101
| 0111
 ________
  0111  = 7 (In decimal)
```

- ^ xor operator

  0 0 = 0

  1 0 = 1

  0 1 = 1

  1 1 = 0

```
a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise XOR Operation of 5 and 7
  0101
^ 0111
 ________
  0010  = 2 (In decimal)
```

- ~ complimentary operator

  x ~x

  0 1

  1 0

- Negation operator

  ~n ⇒ -(n+1)

  eg: ~4 ⇒ -5

  4 binary is 0000 0100

- Left shift operator

  shifting the binary bits to the left by a specified number of spaces.

zeroes will become one and ones will become zeroes.

negation: 1111 1011

this is negative five.

- Right shift operator

  shifting binary bits to the right by a specified number of spaces.

  eg: System.out.println(8>>3); ⇒ 1

Operator Precedence rule and associativity law

Precendence means priority of operators and associativity means that when there is more than one operator in the given expression, to evaluate we will be using the associativity law either going left to right or right to left.

BODMAS rule is used for priority in arithmetic operators.

Q. Sam is having 75 candies, he is giving half of it to his best friend Angel. Since Angel Sam a lot, she is giving back half. Now, calculate and display how many candies Sam and Angel are having individually.

```
class Sam{
  public static void main(String[] args){
    float samsCandies = 75f;
    float angelsCandies = 0f;

    angelsCandies += samsCandies/2;
    samsCandies -= samsCandies/2;

    samsCandies += angelsCandies/2;
    angelsCandies -= angelsCandies/2;

    System.out.println("sam's candies: " + samsCandies);
    System.out.println("Angel's candies: " + angelsCandies);
  }
}
```

Q. Assign five subject marks as S1, S2 and so on, which belong to student Rohit. Now, do the calculation to find his average marks and the percentage he has scored in total.

```
class Rohit{
  public static void main(String[] args){
    byte s1 = 80;
    byte s2 = 75;
    byte s3 = 98;
    byte s4 = 67;
    byte s5 = 100;

    float average = (s1 + s2 + s3 + s4 + s5) / 5f;
    float percentage = (s1 + s2 + s3 + s4 + s5) * 100/500f;

    System.out.println("Average: " + average);
    System.out.println("Percentage: " + percentage);
  }
}
```

Q. 5 xor 7, 5 and 7, 5 or 7.

ans: 5 - 0101

7 - 0111

5^7 = 0010 = 2

5&7 = 0101 = 5

5 | 7 = 0111 = 7

Concatenation

1️⃣ same operator used for arithmetic and concatenation

1+ 2 = 3

“1” + “2” = “12”

+ operator acts as arithmetic additional operator when its used with numbers. On the other hand when we use the same operator with strings, the operation is called as *concatenation.*

formatted output program

```
System.out.println("Arithmetic Operations - my project 1");
int n1 = 100;
int n2 = 5;
System.out.println(n1);
System.out.println(n2);
int add = n1 + n2

// below we use concatenation to format string and variable together.
System.out.println("Addition: " + add);

/*
Output:
Arithmetic Operations - my project 1
Addition: 105
*/
```

Decision making statements

if statement

```
int cost_mobile = 20000;
int budget = 15000;
```

```
if (budget >= cost_mobile) {
  System.out.println("purchased");
}

// if (condtion) { statements }
```

Only when the condition inside () is true, the statements inside curly braces {} are executed.

else statement

```
int cost_mobile = 20000;
int budget = 15000;

if (budget >= cost_mobile) {
  System.out.println("purchased");
}
else {
  System.out.println("did not purchase");
}

/*
if (condition) { statements }
else { different statements }
*/
```

if the statements inside the if condition are false, the system ignores its statements and executes the statements inside else { }

else if statement

```
int cost_mobile = 20000;
int budget = 15000;

if (budget >= cost_mobile) {
  System.out.println("purchased");
}
else if (budget <= cost_mobile){
  System.out.println("did not purchase");
}

/*
if (condition){ statement }
else if (different condition){ different statements }
*/
```

Nested if statements

```
int num = 100;
if (num> 50){
  if (num == 100){
    System.out.println("num is equal to 100");
  }
}
```

Q. Get an integer input from user and find out wether it is an odd or an even number.

```
Scanner scan = new Scanner(System.in);
System.out.print("enter your number: ");
int num = scan.nextInt();

if (num%2 == 0){
  System.out.println(num + " is an even number");
}
else{
  System.out.println(num + " is an odd number");
}
```

Q. Get an integer input from user and find out wether it is positive or negative.

```
Scanner scan = new Scanner(System.in);
System.out.print("enter your number: ");
int num = scan.nextInt();

if (num > 0){
  System.out.println(num + " is a positive number");
}
else if (num < 0){
  System.out.println(num + " is a negative number");
}
else{
  System.out.println("given number is zero");
}
```

Q. Get a number from the user and check if the number is greater than 50. If it is, check if its in between 51 to 100.

```
Scanner scan = new Scanner(System.in);
System.out.print("enter your number: ");
int num = scan.nextInt();

if (num > 50){
  System.out.println("the number is greater than 50");
  if (num >= 51 && num <= 100){
    System.out.println("and the number is in between 51 and 100");
  }
}
else{
  System.out.println("the number is out of range");
}
```

Switch Case

```
int a = 100;

switch(a){
  case 1:
    System.out.println("its 1");
    break;
  case 10:
    System.out.println("its 10");
    break;
  default:
```

```
    System.out.println("default");
}
```

Switch case checks the value of a variable against multiple values in **case:** statements and then executes given code for each case. There is also a **default** case that is executed if no other case satisfies the variable value.

Switch case is a concise way to run different code based on what the value of a variable is. Its shorter to use a switch case in this case rather than if else if statements.

The **break;** line breaks out of the loop, it can be used in other loops and conditional statements like *while*, *for* and *if* as well.

Q. Design a calculater using switch case

Q. Display rainbow colors using switch case

User input

java.util is a package that contains the Scanner class that is used for input taking.

```
import java.util.*;

class UserInput{
  public static void main(String args[]){
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter first number: ");
    int a = sc.nextInt();

    System.out.print("Enter second number: ");
    int b = sc.nextInt();

    int c = a + b;
    System.out.print("Total: " + c);
  }
}
```

token of the input can be taken in any datatype using nextInt(), nextFloat(), nextLong() etc.

To take a sentence as a string input, we use nextLine(); and to get only one word we use next();

```
import java.util.*;

class StringInput{
  public static void main(String args[]){
    Scanner scan = new Scanner(System.in)

    System.out.print("Enter your string: ");
    String mystring = scan.nextLine();

    System.out.print("Enter your character: ");
    String mychar = scan.next();
  }
}
```

Q. Get four float numbers from the user and find the product and display the result.

Q. Write a program to find out area of a circle and display it by taking the radius as a user input.

Q. Take three sides of a triangle as an input and find out wether they can frame a triangle or not.

[**Hint:** sum of any two sides should be greater than or equal to the third side]

Looping statements

loops or control statements are used for code repitition. If we want to run a block of code *n* number of times, instead of writing it repeatedly, we write the block of code once and place it inside a loop.

While loop

```
int num = 1;

while (num < 6){
  System.out.println(num);
  num = num + 1;
}

/*
while (condition) { code to be looped }
the code will be repeated as long as the condition is true
*/
```

Q. Generate all odd numbers between 20 to 40 using a while loop.

```
int num = 21;

while(num > 21 && num <=40){
  System.out.println(num);
  num = num + 2;
}
```

Q. Display all the numbers which are divisible by 2 and 4 in between the range 10 to 50.

```
int num = 10

while (num < 50){
  if (num%2 == 0 && num%4 == 0){
    System.out.println(num);
  }
  num += 1;
}
```

Q. Display all the numbers which are divisible by 3 and 7 between the range 1 to 100.

```
int num = 1;

while (num <= 100){
  if (num%3 == 0 && num%7 == 0){
    System.out.println(num);
  }
```

```
  num += 1;
}
```

Q. Reverse a given number.

```
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
int rev = 0;
int rem = 0;

while (n>0){
  rem = n % 10;
  rev = rev * 10 + rem;
  n = n/10;
}

System.out.println(rev);
```

Q. Write a program to find the number of digits of a given input integer

```
Scanner scan = new Scanner(System.in);

System.out.print("enter number: ");
int num = scan.nextInt();
int i = 0;

while (num > 0){
  num = num%10;
  i += 1;
}

System.out.println("the number of digits is "+i);
```

Do - while loop

The difference between while and do-while loop is wether the condition is true or false, the block of code will get executed at least once in do-while loop unlike while loop because it checks for the condition after executing the code.

```
int num = 1;

do {
  System.out.println(num);
  num += 1;
} while (num < 10);

/*
do {code to be repeated}
while (condition);
*/
```

For loop

For loop contains three sections inside the parenthesis,

- initialisation - declaring an iterator variable
- test condition - condition until which the loop runs
- increment or decrement

difference between for loop and while loop is that number of iterations are unknown in while loop but known in for loop, the syntax is different, initialisation of a variable is done first inside a for loop but in a while loop we can initialise anywhere in the body.

```
for (int i =1; i <= 10; i++){
  System.out.println(i);
}

// for (initialisation; test condition; increment or decrement) { statements }
```

Q. Write a program to print the multiplication table of an inputted number.

```
Scanner scan = new Scanner(System.in);

System.out.print("enter your number: ");
int tableNumber = scan.nextInt();

for (int i = 1; i<=10; i++){
  System.out.println(tableNumber + " x " + i + " = " + (i*tableNumber));
}
```

Q. Write a program to print the factorial of inputted number using a for loop.

```
Scanner scan = new Scanner(System.in);

System.out.print("enter your number: ");
int i = 1;

for (int factorialNumber = scan.nextInt(); factorialNumber > 0; factorialNumber--){
  i = i * factorialNumber;
}

System.out.println(i);
```

Q. Sum of natural numbers from 1 to n using for loop

```
Scanner scan = new Scanner(System.in);
System.out.print("enter your number: ");
int sum = 0;

for (int number = scan.nextInt(); number>0; number--){
  sum = sum + number;
}

System.out.println("sum of the natural numbers is "+sum);
```

Enhanced for loop

Enhanced for loops are nothing but nested for loops.

```
int i,j;

for (i=0; i<3; i++){
  for (j=0, j<3; j++){
    System.out.print(i+""+j+"\t");
  }
  System.out.println("");
}

/*

output :
  00  01  02
  10  11  12
  20  21  22

*/
```

for every outer loop iteration, the inner loop gets executed completely.

Q. Use nested for loops to print a pattern of numbers increasing in every new row.

```
import java.util.*;

class Pattern{
  public static void main(String args[]){
    Scanner scan = new Scanner(System.in);

    int inputNum = scan.nextInt();

    for(int i = 1; i<=inputNum; i++){
      for (int j = 1; j<=i; j++){
        System.out.print(j+" ");
      }
    System.out.println();
    }

  }
}
```

Arrays

Normally, we can use one value with one variable, in order to make this more efficient by storing multiple values in one variable name, we are using this concept of Arrays.

> 💡 **Definition:** Collection of homogenous datatype values stored together inside one variable name.

Array has a fixed size. Once the size has been declared, you cannot store more elements than the fixed size.

Types of Arrays

▼ 1D - One dimensional Array

Declaration of array is done in the following manner

```
int a[] = {100, 200, 300, -9, 4, 9, -2};
```

To access elements inside the array, we use indices inside the [] brackets

```
System.out.print(a[0]);  // output : 10
System.out.print(a[1]);  // output : 20
System.out.print(a[3]);  // output : 30
```

Note : Any particular Array can only have homogenous datatypes i.e it can only store elements that are of the same datatype. This is a limitation of Arrays

Q. Create a float array with five numbers, then extract and print all of its values

```
float array[] = {6.78f, 42.3f, 7.8f, 69.420f, 90.65f};
for (int i = 0; i<5; i++){
  System.out.print(array[i]);
}
```

Q. Find the sum of all the elements of an array.

```
int array[] = {78, 56, 29, 39, 98};
int i, sum =0;

for(i=0; i<5; i++){
  sum += array[i];
}
System.out.println(sum);
```

Q. Count the number of even numbers and odd numbers from a given array.

```
int[] arr = {1,7,45,22,56,90,21,64,5,19};
int even = 0, odd = 0;

for (int i =0; i<10; i++){
  if (arr[i]%2 == 0){
    even += 1;
  } else{
    odd +=1;
  }
}

System.out.println("even count is: " + even + "\nodd count is: " + odd);
```

Q. Find the maximum and minimum values of an array.

```
int arr[] = {1,7,45,22,56,90,21,64,5,19};
int max = arr[0], min =arr[0];

for(int i=0; i<10; i++){
  if (arr[i] > max){
    max = arr[i];
  }
}
System.out.println("max value is: " + max);

for(int i=0; i<10; i++){
  if (arr[i] < min){
    min = arr[i];
  }
}
System.out.println("min value is: " + min);
```

▼ 2D - Two dimensional Array

Declaration of a 2D array

```
int array2D[][] = {{4, 5, 0}, {9, 8, 7}};

// empty 2D array:
int arrayEmpty[][] = new int [3][3];

// the [][] takes in size of the row, here: 3 and 3.
```

To Access the elements of a 2D array, we use double square brackets like this **arrayName[][]**

```
System.out.println(array2D[0][1]);

// output: 5
```

the first square [] bracket takes the row index and the second [] one takes the element index.

so **array2D[1][2]** would be the 3rd element in the second row, which in this case is **7**.

Q. Construct a 2D array with user input and display the array elements

[**Hint :** use nested for loops]

```
int [][] arr= new int [3][3];
for(int i=0; i<3; i++){
  for (int j=0; j<3; j++){
    System.out.print("enter your element ");
    arr[i][j]=scan.nextInt();
  }
}

for (int i=0; i<3; i++){
  for(int j=0; j<3; j++){
    System.out.print(arr[i][j] + "\t");
  }
```

```
    System.out.println();
  }
```

▼ Multi-dimensional Array

Q. Create an array using user input

```
import java.util.*
class ArrayInput{
  public static void main(String args[]){

    Scanner scan = new Scanner(System.in);

    System.out.print("Enter array size: ");
    int size = scan.nextInt();

    int userArray[] = new int[size];

    for (int i=0; i<size; i++){
      System.out.print("Enter array element of index "+ i + " : ");
      int userArray[i] = scan.nextInt();
      System.out.println();
    }

    System.out.println("This is your array: ");
    for (int i=0; i<size; i++){
      System.out.println(userArray[i]);
    }
  }
}
```

Q. Count the number of zeroes present inside an array

```
int array[5] = {34, 0, 89, 0, 6};
int zeroCount = 0;

for (int i=0; i<5; i++){
  if (array[i] == 0){
    zeroCount += 1;
  }
}

System.out.println(zeroCount);
```

Problem Solving

Approaching a problem

Read and understand the question clearly

- input
- logic

- output

Functions

▼ Pre-defined or system defined functions

These functions are already defined by the system in the java language or are present in the packages that we can import to our class. These can be called wherever we want without having to define the code.

Examples are:

- print();
- println();
- nextInt();
- toUpperCase();

▼ User defined functions

Definition: For code reusability, we use functions. Instead of writing the same concept again and again, we write it once and place it inside a function so that we can use that function wherever we require those lines of code.

Types of functions

1. Functions without argument and without return value
2. Functions without argument and with return value
3. Functions with argument and without return value
4. Functions with argument and with return value

defining or declaring a function

```
class LearningFunctions{
  public static void main(String args[]){

    public static void functionName(){
      System.out.println("this is a function");
    }

  }
}
```

the function definition is done exaclty like the main() method we use inside a class.

when we **call the function,** it runs the code inside the function also called the **function description.**

```
// calling the function
functionName();
```

```
// output : this is a function
```

Example function that adds two predefined numbers

```
public static void sum(){
  int n1 = 100;
  int n2 = 200;
  int res = n1 + n2;

  System.out.println(res);
}

sum();

// output : 300
```

Functions are also called as methods in object oriented programming languages. Method description/ definition is given inside the class and outside the main method and the function is called inside the main() method. This is called the **method flow.**

```
class methodCheck{
  public static void main(String args[]){
    display();
  }

  public static void display(){
    System.out.println("display");
  }
}
```

Q. Using type 1 function, create three methods. One to find out the sum of digits of given number. Second is to check the sum of digits of given number is lying between 100 and 200. Third function is to multiply sum of digits of a given number with 10 and count number of zeroes in it and print.

```
import java.util.*;

public class DayEight {
    public static void main(String args[]){
        sumOfDigits();
        sumInRange();
        zeroesInSum();
    }

    public static void sumOfDigits(){
        Scanner scan = new Scanner(System.in);

        System.out.print("enter your number: ");
        int number = scan.nextInt();
        int initNumber = number;
        int rem,sum = 0;

        while(number>0){
```

```
            rem = number % 10;
            number /= 10;
            sum += rem;
        }

        System.out.println("the sum of the digits of " + initNumber + " is " + sum);
    }

    public static void sumInRange(){
        Scanner scan = new Scanner(System.in);

        System.out.print("enter your number: ");
        int number = scan.nextInt();
        int initNumber = number;
        int rem,sum = 0;

        while(number>0){
            rem = number % 10;
            number /= 10;
            sum += rem;
        }

        if (sum>=100 && sum<=200){
            System.out.println("sum of " + initNumber + " : " + sum + " lies between 100 and 200");
        } else {
            System.out.println("sum of " + initNumber + " : " + sum + " does not lies between 100 and 200");
        }
    }

    public static void zeroesInSum(){
        Scanner scan = new Scanner(System.in);

        System.out.print("enter your number: ");
        int number = scan.nextInt();
        int rem,sum = 0;

        while(number>0){
            rem = number % 10;
            number /= 10;
            sum += rem;
        }

        sum *= 10;
        int initSum = sum;
        int zeroCount = 0;

        while (sum>0){
            rem = sum % 10;
            if (rem == 0){
                zeroCount += 1;
            }
            sum /= 10;
        }

        System.out.print("the sum multiplied by 10 : " + initSum + " has " + zeroCount + " zeroes");

        scan.close();
    }
}
```

we can create multiple classes inside a single .java file. Example:

we name the .java file the same name of the class that has the main() method.

when dealing with multiple classes, we can import them into each other using the new keyword as an object and then use its functions as methods by **ObjectNumber.MethodName();**

```
class PalindromeFunction{
  public static int palNumber(int n){
    int rem, rev = 0;
    while (n>0){
      rem = n % 10;
      rev = rev*10 + rem;
      n /= 10;
    }
    return rev;
  }
}

class CheckPalindromeNumber{
  public static void main(String args[]){
    PalindromeFunction pf = new PalindromeFunction();
    int n = 7890;
    if (pf.palNumber(n) == n){
      System.out.println("palindrome");
    } else {
      System.out.println("Not a palindrome");
    }
  }
}
```

Four core concepts of OOP

Before we learn about the four pillars of object oriented programming, we need to know some basic features of classes and objects that we will be using.

Constructors:

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called.

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

```
class Animal{
  Animal(){
    System.out.println("An animal is created");
  }

  public static void main(String args[]){
    Animal lion = new Animal();
  }
}
```

As we have created an object lion from the class Animal, the constructor of the class is called and we see the output as : *An animal is created*.

We can also created a **parameterized constructor** by giving in arguments during the constructor defintion, this allows us to provide different values for the class properties to distinct objects during object

creation.

```
class Animal{
  // properties of the Animal class
  int population;
  String species;

  Animal(int p, String s){
    population = p;
    species = s;
  }

  // method to display the properties
  void display(){
    System.out.println("species is: " + species);
    System.out.println("population is: "+ population);
  }

  public static void main(String args[]){
    Animal lion = new Animal(45000, "Panthera leo");
    Animal giraffe = new Animal(1200, "Giraffa camelopardalis");

    lion.display();
    giraffe.display();
  }
}
```

Here we create a class Animal that has two properties population and species. In the constructor we give p and s as variables and assign them to the population and species properties of the class. So that whenever we are creating an object of the class Animal, we can pass in the values of population and species as shown above.

static keyword:

The **static keyword** in Java is used for memory management mainly. We can apply static keyword with variables, methods and blocks. The static keyword belongs to the class than an instance of the class.

The static can be:

▼ Variable (also known as a class variable)

The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc. The static variable gets memory only once in the class area at the time of class loading.

▼ Method (also known as a class method)

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

▼ Block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}

// output:
// static block is invoked
// Hello main
```

⇒ **The four core concepts or pillars of OOP are as given below:**

▼ Inheritance

Inheritance is a mechanism that allows child classes to access the properties and methods of the parent class they were extended from. To create a child class from a parent class, we use the **extends** keyword while class creation.

```
class Parent{
  public void helloParent(){
    System.out.println("Hello I am a parent");
  }
}

public class Child extends Parent{
  public void helloChild(){
    System.out.println("Hello I am a child");
  }
}

public class Main{
  public static void main(String args[]){
    Child obj = new Child();
    obj.helloParent();
    obj.helloChild();
  }
}
```

Here, we create a parent class first with the method **helloParent()** then we create a child class extending from the parent class, the child class contains the method **helloChild().** In the main method we create an object of the child class. Now, by inheritance the child class object not only has the **helloChild()** method but it also inherits the **helloParent()** method.

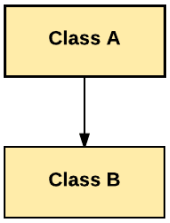
This passing down of properties and methods from the parent class to the child class is called Inheritance.

The parent class in this case is also called the base class or the super class. The child class is also called the derived class or the sub class.

There are five types of Inheritance in java:

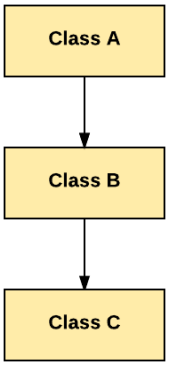
▼ Single-level Inheritance

As the name suggests, this type of inheritance occurs for only a single class. Only one class is derived from the parent class.



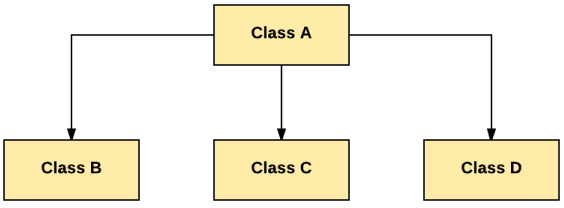
▼ Multi-level Inheritance

The multi-level inheritance includes the involvement of at least two or more than two classes. One class inherits the features from a parent class and the newly created sub-class becomes the base class for another new class.



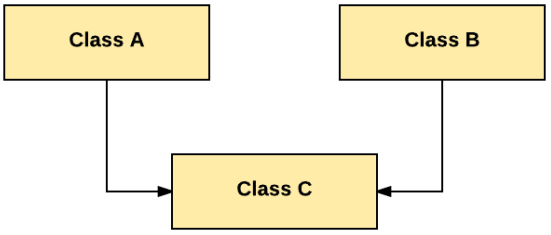
▼ Hierarchical Inheritance

The type of inheritance where many subclasses inherit from one single class is known as Hierarchical Inheritance.



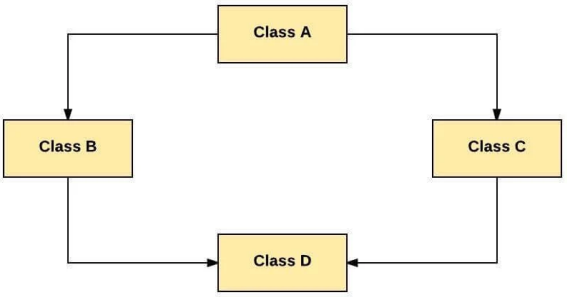
▼ Multiple Inheritance

Multiple inheritances is a type of inheritance where a subclass can inherit features from more than one parent class. In java, multiple inheritances can be achieved through interfaces.



▼ Hybrid Inheritance

Hybrid inheritance is a combination of more than two types of inheritances single and multiple. It can be achieved through interfaces only as multiple inheritance is not supported by Java.



▼ Polymorphism

Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations, we can have different functionality for the same method name. The word “poly” means many and “morphs” means forms.

⇒ Java has two types of polymorphism:

▼ Compile time polymorphism

It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

```
class Helper {
    // Multiplication of 2 numbers
    static int Multiply(int a, int b)
    {
        return a * b;
    }

    // // Multiplication of 3 numbers
    static int Multiply(int a, int b, int c)
    {
        return a * b * c;
    }
```

```
}

class GFG {
    public static void main(String[] args)
    {
        System.out.println(Helper.Multiply(2, 4));
        System.out.println(Helper.Multiply(2, 7, 3));
    }
}
```

We use same method name but different arguments

▼ Run time polymorphism

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

We create sub classes of a parent class that share a common method but have different implementation in each child class and during object creation, the class chosen decides the functionality of the common method.

```
class Bank{
  float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
  float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
  float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
  float getRateOfInterest(){return 9.7f;}
}
class TestPolymorphism{
  public static void main(String args[]){
    Bank b;
    b=new SBI();
    System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
    b=new ICICI();
    System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
    b=new AXIS();
    System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
  }
}
```

here, although the method used **getRateOfInterest()** is the same in all cases but since we are using different child classes, the output will be different.

We use same method name and same arguments but objects of different classes.

▼ Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

⇒ Abstract classes

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

```
abstract class Parent{}
```

⇒ Abstract method

A method which is declared as abstract and does not have implementation is known as an abstract method. The implementation for an abstract method should be given in a child class of the parent class.

```
abstract class Parent{
  abstract void showResult();
}

class Child extends Parent{
  void showResult(){
    System.out.println("Result");
  }
}
```

Since an abstract method has no body, it does not need {} curly braces. Instead of that, we put a semicolon after the parenthesis like so: **abstract void methodName();**

⇒ Interface

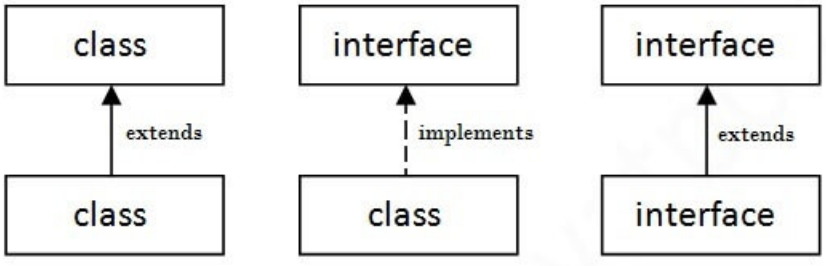
An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

When creating a child class for an interface we use the keyword implements. Given below is a diagram that shows the keywords used for creating a child element.



Syntax for Interfaces:

```
interface printStatement{
  void print();
}

class A6 implements printable{
  public void print(){System.out.println("Hello World");}

  public static void main(String args[]){
    A6 obj = new A6();
    obj.print();
  }
}
```

▼ Encapsulation

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

Encapsulation provides control over the data since you hard-code the setter methods and it provides data hiding since we can use private properties in classes.

```
public class Student{
  private String name;

  public String getName(){
    return name;
  };

  public void setName(String name){
    this.name = name
  }
}
```

In the above Student class, we have encapsulated the data of the student name which can be set or accessed using the getter and setter methods in the class. Lets create an object and try it out.

```
public class testStudent{
  Student safwan = new Student;
  safwan.setName("mohammed safwan");
  System.out.println(safwan.getName());
}

// output
// mohammed safwan
```

We can also make the class read-only by just creating a getter method. We can also make the class write-only by just creating a setter method.