THE START

## **Quick word about java and why me…**

Java was initiated by James Gosling, when the term was coined by him after having a cup of coffee nearby his office.(as Java is an island in Indonesia where the first coffee was invented). It is also said that "Oak '' was initially the name given to the company but later changed to java as the workers also liked the name along with James(the founder).

These images are the proof of java referring to a coffee.

Then, in 1995 they got famous and then in 1996 they released their first jdk called jdk 1.0. Since then there has been a lot of development to this language and now(in 2023) I have come to learn all of java. For what it’s worth, I am trying to be a Software developer, and now I need to learn Java to survive the jobs and my higher studies.

## 

## 

## **The plan**

The plan may keep changing as I go along the path of my life and with this language. There is no specific pathing, but we can always add a strategy along the way.

So, this is how we go:

* Go to Javatpoint(<https://www.javatpoint.com/java-tutorial>) and learn the basics, which basically means just go through all of the topics and suffer eternal pain for hours.
* Spend at least 3 hours a day on learning the basics and getting comfortable with the language.
* Then do some projects related to Java.
* More tasks to be added soon…..

## **Task 1: Javatpoint**

### **1] Features:**

Once a movie taught me the features of java, “java is simple and powerful”. It was from the movie Premam.

But guess what, “Reality is often disappointing” like Thanos said. There are a lot of features apart from those 2 mentioned above.

And hence here we go:

* First and foremost **Java is simple.** That’s what someone who finished their college degree and going to job would say.

*“Simple my balls”*

* *Rishekesh.B*

Yeah that’s what I would quote. Sue me if you want, dickheads who say otherwise.

But anyways, let’s get “down” to business(pun intended).

* According to Sun Microsystem, Java is neat(or simple), clean and easy to understand(if you know some coding).
* Java syntax is based on C++, ha!, how ironic, cause the last time I met up with C++ it didn’t go as expected.(I failed in c++, the first sem of my cs degree.)
* Java, unlike C, has removed complicated and rarely-used features like explicit pointers and overloading, etc.
* And, I have no idea what this means(for now) but there is an automatic garbage collection in this program.

* Java is **Object-oriented programming.** Everything in java is an object.Object oriented means we organise our software as a combination of types of objects that incorporate both data and behaviour.

For now we don’t see the purpose of OOPs concept, as we are just beginners. We will just know the definition for now and then we will come back to this section later on.

Basic concepts include:

* Encapsulation
* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Java is **Platform independent**, meaning that java is a Write Once Run Anywhere(WORA) type of language. Programs like c,c++ needs to be run on an compiler to convert the human language code into a computer language code, but java doesn’t need to look out for that.

Basically there are 2 types of platforms:

* Software-based
* Hardware-based

Java provides a software-based platform that runs on a hardware-based platform, which again has 2 components:

* Runtime environment
* API(Application Programming Interface)
* Java is **secure** because it has no explicit pointers and it runs inside a virtual machine sandbox.
* Java is **robust** because it uses strong memory management, it lacks pointers that avoid security problems, there is exceptional handling and type checking mechanisms.
* Java is **architecture neutral** because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

* Java is **portable** because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.
* Java uses **Multi-threading**. Multi-threading is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multimedia, Web applications, etc.
* Java is a **dynamic language**. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

The above points are the features of java. Now, lets see if this comes in help

## **2] The universal first program in every programming language:**

Every programming language starts with a “hello, world” program by default. The words are the same but the structure or code differs. It is also written so that you don’t get your light diminished out of you, because you do not know this certain language you’re trying to understand.

Well, without further adieu, lets begin:

First of all I am using eclipse ide to code. If I ever forget how to do the setups again, then this is how. The 5 minute long video will help. [<https://www.youtube.com/watch?v=HmF3yQ_nj5Q>] .

Second, the code:

**public class firstProgram {**

**public static void main(String args[]) {**

**System.out.println("hello,world!");**

**}**

**}**

Third, the output:

**Hello, world!**

Fourth, Enjoyment:

Now I am a coder, I know, I must be feeling super proud of myself. I should tell everyone that I did it.

*“Wake up to reality”*

*-madara uchiha*

Well it's too late to celebrate your first program, cause no one gives a shit. Even a new born baby can use OOPs and DSA and write 500 lines of code in its sleep. And You just wrote a hello fucking world program and you wanna feel good about it.

OMG, get on track and do what you gotta do soldier!! Go!!

## **3] JDK, JRE, JVM:**

**(I don’t really understand it, so I will come back once I know it…)**

## **4] Variables:**

A variable is a container to hold a value in it, it holds data types.

There are 3 types of variables we use in java: static, local, instance.

And of course there are 2 types of data types: primitive and non-primitive.

A variable allocates a space in the memory for the certain data type to be stored. And if you notice closely variable can be split as “vari + able”, which means the value can be changed.

Types of variables explained with example:

* Local variable:

A variable declared inside the body of the method is called a local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.A local variable cannot be defined with "static" keyword.

* Instance variable:

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

* Static variable:

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Example for the above:

public class A

{

static int m=100; //static variable

void method()

{

int n=90; //local variable

}

public static void main(String args[])

{

int data=50; //instance variable

}

} //end of class

Practice:

1] Write a program to add two number

Program:

public class firstProgram {

public static void main(String args[]) {

int a=10;

int b=12;

int c = a+b;

System.out.println(c);

}

}

Output:

22

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2] program to show “widening”

Program:

public class firstProgram {

public static void main(String args[]) {

int a=10;

float b = a;

System.out.println(a);

System.out.println(b);

}

}

Output:

10

10.0

3] program to show “Narrowing”

Program:

public class firstProgram {

public static void main(String args[]) {

float f = 20.5f;

int newn = (int)f;

System.out.println(newn);

}

}

Output:

20

4] program to show “overflow”

Program:

public class firstProgram {

public static void main(String args[]) {

int a = 130;

byte c = (byte)a;

System.out.println(a);

System.out.println(c);

}

}

Output:

130

-126 (rounds of the value closest to it)

5] Adding 2 lower types

Program:

public class firstProgram {

public static void main(String args[]) {

byte a = 10;

byte b = 10;

byte c = (byte)(a+b);

// byte c = a+b; throws an error, "Type mismatch: cannot convert from int to byte"

System.out.println(c);

}

}

Output:

20

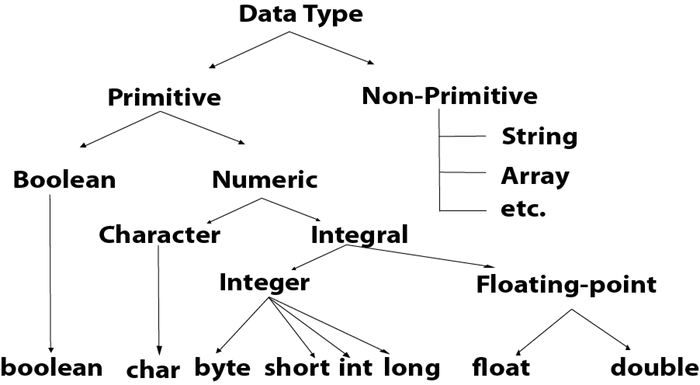
## 

## **5] Data types:**

There are 2 types of data types in java, and almost every language know until now, called:

* Primitive data type
* Non-primitive data type

Primitive data types are the ones mentioned below:



| **DATA TYPE** | **DEFAULT VALUE** | **DEFAULT SIZE** | **VALUE RANGE** |
| --- | --- | --- | --- |
| boolean | fales(0) | 1 bit | true/false |
| char | ‘\u0000’ | 2 byte | ‘\u0000’ to ‘\uffff’ |
| byte | 0 | 1 byte | -128 to 127 |
| short | 0 | 2 byte | -32,768 to 32,767 |
| int | 0 | 4 byte | (-2^31) to ( 2^31-1) |
| long | 0L | 8 byte | (-2^63) to (2^63 -1) |
| float | 0.0f | 4 byte | unlimited |
| double | 0.0d | 8 byte | unlimited |

## **6] Operators and Keywords:**

Since operators are common in all languages, i'm gonna skip the boring explanation part.

But operators are important and some are worth looking up during dire times of need, and so: [<https://www.javatpoint.com/operators-in-java>] this link will help.

This table below took me 10 mins to make(ig), better it be worth it!!

| Operator type | category | precedence |
| --- | --- | --- |
| unary | postfix | expr++ , expr– – |
| prefix | ++expr , - - expr , + expr , – expr , ~, ! |
| Arithmetic | multiplicative | \* , / , % |
| additive | + , – |
| Shift | shift | << , >> , >>> |
| Relational | comparison | < , > , <= , >= |
| equality | == , != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive XOR | | |
| Ternary | ternary | ? : |
| Assignment | assignment | = , += , -= , \*= , /= , %= , &= , ^= , |= , <<= , >>= , >>>= |

**Keywords** are reserved words. They act as a key to the code. You can’t use them as variable names, because they are predefined by the creators to do a specific function when called for. Eg:- Boolean, byte, case,catch,try. Etc.

Now there are more keywords with definition, here: <https://www.javatpoint.com/java-keywords> , check them out.

**7] Control statements:**

When it comes to coding, there is a structure and a flow. What I mean by flow is that every program executes its function from the top to bottom.

But there is a way to control the flow, and these methods fall into the control statements such as:

* **Decision making statements:**

As the name suggests the decision making statement helps us give a decision of yes or no, which means in terms of boolean. Now, there are 2 types in this decision making statements, namely:

* **If-statement**:

The control of the program is diverted depending upon the specific condition. The “if” statement gives a boolean output. It has 4 types:

* **Simple “if” statement:**

It evaluates a boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Eg: program to see if a student age is above 20.

Program:

public class firstProgram{

public static void main(String args[]) {

int x = 19;

if(x>18) {

System.out.println("The student age is right");

}

}

}

Output:

The student age is right

* **“If-else” statement:**

The if-else condition is an extension to the if condition. The else block executes if the condition of the if block is evaluated to false.

Eg:- Program to know if a person is infected or not.

Program:

public class firstProgram{

public static void main(String args[]) {

String health = "infected";

if(health != "infected") {

System.out.println("this person is helathy");

}

else {

System.out.println("this person is infected");

}

}

}

Output:

this person is infected

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

It is a chain of if else statements that create a decision tree where the program may enter in the block of code if the condition is true.

eg: - A program to know the largest between 2 numbers

Program:

public class firstProgram{

public static void main(String args[]) {

int a = 1;

int b = 1;

if(a>b) {

System.out.println("a is greater than b");

}

else if(a==b) {

System.out.println("they both are equal values");

}

else {

System.out.println("b is greater than a");

}

}

}

Output:

they both are equal values

* **Nested if:**

In nested if statement, there can be multiple “if” or “if-else” statements.

Eg: program to find the city you live in using nested if.

Program:

public class firstProgram{

public static void main(String args[]) {

String address = "coimbatore, tamilnadu";

if(address.endsWith("tamilnadu")) {

if(address.contains("chennai")) {

System.out.println("you live in chennai");

}

else if(address.contains("tiruchi")) {

System.out.println("you live in tiruchi");

}

else {

System.out.println(address.split(",")[0]);

}

}

else {

System.out.println("you don’t live in tamilnadu");

}

}

}

Output:

Coimbatore

* **Switch case:**

Switch statements are like if-else statements but use cases instead of if and else. People usually go for switch statements if there are multiple cases and can make the code look shabby and long. It enhances the readability og]f the code. Cases cannot be duplicate.

Eg:- A program to check how switch cases work

Program:

public class firstProgram{

public static void main(String args[]) {

int number = 10;

switch(number) {

case 1:

System.out.println("1");

break;

case 2:

System.out.println("2");

break;

default:

System.out.println(number);

break;

}

}

}

Output:

10

* **Loop statements:**

Sometimes we need to execute a block of code repeatedly while some conditions evaluate to be true. However, loop statements are used to execute a set of instructions over and over again, until it turns out to be false. This execution of the set of instructions requires a specific condition to be true.

There are 3 types of loops:

* **For loop:**

Like all other programming languages, for loop completes the task of looping in one single line. It consists of loop variables, check condition, and increment/decrement in a single line of code. We only use the loop when we know the exact number of times we want to repeat the loop.

Eg: To find the sum of first 10 natural number:

Program:

public class firstProgram{

public static void main(String args[]) {

int sum = 0;

for(int i=1; i<=10; i++) {

sum = sum + i;

}

System.out.println("the sum of first 10 natural numbers is: "+sum);

}

}

Output:

the sum of first 10 natural numbers is: 55

* **For-each loop:**

For-each loop is an advanced loop for DSA. The catch is that we don’t need to update the loop variable.

Eg: program to demonstrate the working of for-each loop.

Program:

public class firstProgram{

public static void main(String args[]) {

String[] names = {"rishe","bishe","shushi","mushy"};

System.out.println("the names are:");

for(String name:names) {

System.out.println(name);

}

}

}

Output:

the names are:

rishe

bishe

shushi

mushy

* **While loop:**

If we don’t know the number iterations needed beforehand we use a while loop. It is also called entry controlled loop as the while loop checks the condition before entering into the loop, like a bodyguard in front of a clubhouse.

Eg: print the list of first 10 even numbers

Program;

public class firstProgram{

public static void main(String args[]) {

int i = 0;

while(i<20) {

i = i+2;

System.out.println(i);

}

}

}

Output:

2

4

6

8

10

12

14

16

18

20

* **Do-while:**

The difference between while and do-while loop is while loop checks and executes later, but do-while loop executes the loop and then checks the condition. It is also called exit control loop .

Eg: print first 10 odd numbers

Program:

public class firstProgram{

public static void main(String args[]) {

int num = 1;

do {

System.out.println(2 \* num - 1);

}while(++num<=10);

}

}

Output:

1

3

5

7

9

11

13

15

17

19

**Practice:**

1] Program to check if a number is even or odd [if-else]

Program:

public class firstProgram{

public static void main(String args[]) {

int num = 1;

if(num%2 == 0) {

System.out.println("it is an even number");

}

else {

System.out.println("it is odd number");

}

}

}

Output:

it is odd number

2] program to check even or odd [using Ternary operation]

Program:

public class firstProgram{

public static void main(String args[]) {

int num = 10;

String output = (num%2 == 0)? "even number" : "odd number";

System.out.println(output);

}

}

Output:

even number

3] program to check if a year is a leap year or not [if-else]

Program:

import java.util.Scanner;

public class firstProgram{

public static void main(String args[]) {

Scanner myObj = new Scanner(System.in);

System.out.println("enter the year: ");

int year = myObj.nextInt();

if(((year%4 == 0) && (year%100 != 0)) || (year%400 == 0)) {

System.out.println(year + " is a leap year");

}

else {

System.out.println(year+" is a regular year");

}

}

}

Output:

enter the year:

2020

2020 is a leap year

4] Program to check if the student passed or failed the exam [if-else-if ladder]

Program:

import java.util.Scanner;

public class firstProgram{

public static void main(String args[]) {

Scanner myObj = new Scanner(System.in);

System.out.println("enter the mark you got: ");

int mark = myObj.nextInt();

if(mark<50) {

System.out.println("You failed");

}

else if(mark>=50 && mark<60) {

System.out.println("You got B");

}

else if(mark>=60 && mark<70) {

System.out.println("You got B+");

}

else if(mark>=70 && mark<80) {

System.out.println("You got A");

}

else if(mark>=80 && mark<90) {

System.out.println("You got A+");

}

else if(mark>=90 && mark<=100) {

System.out.println("You got O");

}

else {

System.out.println("Invalid score");

}

}

}

Output:

enter the mark you got:

100

You got O

5] Program to check positive or negative number. [if-else-if ladder]

Program:

import java.util.Scanner;

public class firstProgram{

public static void main(String args[]) {

Scanner myObj = new Scanner(System.in);

System.out.println("enter the number to check positive or negative: ");

int number = myObj.nextInt();

if(number>0) {

System.out.println(number + " is positive.");

}

else if(number<0) {

System.out.println(number + " is negative.");

}

else {

System.out.println(number + " is zero.");

}

}

}

Output:

enter the number to check positive or negative:

0

0 is zero.

6] Program to check if a person is eligible to donate blood or not [nested if-else]

Program:

import java.util.Scanner;

public class firstProgram{

public static void main(String args[]) {

Scanner myObj = new Scanner(System.in);

System.out.println("enter the age: ");

int age = myObj.nextInt();

System.out.println("enter the weight: ");

int weight = myObj.nextInt();

if(age>=18) {

if(weight>=60) {

System.out.println("You are eligible to donate blood.");

}

else {

System.out.println("you are not eligible to donate blood. ");

}

}

else {

System.out.println("you are not eligible to donate blood");

}

}

}

Output:

enter the age:

19

enter the weight:

4

you are not eligible to donate blood.

7] program to know the day based on the number given [Switch case]

Program:

import java.util.Scanner;

public class firstProgram{

public static void main(String args[]) {

Scanner myObj = new Scanner(System.in);

System.out.println("enter the number for the day name: ");

int dayNumber = myObj.nextInt();

String dayName = "";

switch(dayNumber) {

case 1:

dayName = "1-Monday";

break;

case 2:

dayName = "2-Tuesday";

break;

case 3:

dayName = "3-Wednesday";

break;

case 4:

dayName = "4-Thursday";

break;

case 5:

dayName = "5-Friday";

break;

case 6:

dayName = "6-Saturday";

break;

case 7:

dayName = "7-Sunday";

break;

default:

System.out.println("enter a number between 1-7 ");

}

System.out.println(dayName);

}

}

Output:

enter the number for the day name:

3

3-Wednesday

8] program to find if a character has a vowel or not. [Switch case]

Program:

import java.util.Scanner;

public class firstProgram{

public static void main(String args[]) {

Scanner newObj = new Scanner(System.in);

System.out.println("enter the vowel you want to check: ");

char vowel = newObj.next().charAt(0);

switch(vowel) {

case 'a':

System.out.println("vowel");

break;

case 'e':

System.out.println("vowel");

break;

case 'i':

System.out.println("vowel");

break;

case 'o':

System.out.println("vowel");

break;

case 'u':

System.out.println("vowel");

break;

case 'A':

System.out.println("vowel");

break;

case 'E':

System.out.println("vowel");

break;

case 'I':

System.out.println("vowel");

break;

case 'O':

System.out.println("vowel");

break;

case 'U':

System.out.println("vowel");

break;

default:

System.out.println("It is a consonant");

break;

}

}

}

output:

enter the vowel you want to check:

A

vowel

9] Program to get a syllabus for a specific stream in a specific year of college. [Switch case]

Program:

import java.util.Scanner;

public class firstProgram{

public static void main(String args[]) {

Scanner myData = new Scanner(System.in);

System.out.println("enter the branch[CSE,Mechanical,Civil] to get the syllabus: ");

String branch = myData.nextLine();

System.out.println("enter the year[1,2,3,4] to get specific result: ");

int year = myData.nextInt();

switch(year) {

case 1:

System.out.println("Science,maths,english");

break;

case 2:

switch(branch)

{

case "CSE":

System.out.println("maths,english,data science");

break;

case "Mechanical":

System.out.println("maths,engine,hydro fluid");

break;

case "Civil":

System.out.println("maths,bricks,building plan");

break;

}

break;

case 3:

switch(branch)

{

case "CSE":

System.out.println("maths,Software,ML");

break;

case "Mechanical":

System.out.println("motor,rotor,engine");

break;

case "Civil":

System.out.println("architecture,chemical stuff,cement");

break;

}

break;

case 4:

switch(branch)

{

case "CSE":

System.out.println("data analyst, development");

break;

case "Mechanical":

System.out.println("cars,bike,flight,boat");

break;

case "Civil":

System.out.println("make a house 101, stop breaking stuff 911");

break;

}

break;

default:

System.out.println("select a year and branch");

break;

}

}

}

Output:

enter the branch[CSE,Mechanical,Civil] to get the syllabus:

CSE

enter the year[1,2,3,4] to get specific result:

2

maths,english,data science

10] Switch statement to demonstrate the use of **enum.** [switch case]

Program:

import java.util.Scanner;

public class firstProgram

{

public enum Day

{

sun,mon,tues,wed,thurs,fri,sat

}

public static void main(String args[])

{

Day[] toDay = Day.values();

for(Day day:toDay)

{

switch(day)

{

case sun:

System.out.println("sunday");

break;

case mon:

System.out.println("monday");

break;

case tues:

System.out.println("tuesday");

break;

case wed:

System.out.println("wednesday");

break;

case fri:

System.out.println("thursday");

break;

case sat:

System.out.println("FRIDAY");

break;

}

}

}

}

Output:

sunday

monday

tuesday

wednesday

thursday

friday

11] program to demonstrate a wrapper class in java. [switch case]

Program:

12] program for basic for loop [for loop]

Program:

public class firstProgram{

public static void main(String args[]) {

for(int i=0;i<3;i++) {

System.out.println(i);

}

}

}

Output:

0

1

2

13] program to show nested for-loop [nested for loop]

public class firstProgram{

public static void main(String args[]) {

for( int i=1;i<=3;i++) {

for( int j=1; j<=3;j++) {

System.out.println(i+" "+j);

}

}

}

}

Output:

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

14] program to print a right pyramid. [for-loop]

Program:

public class firstProgram{

public static void main(String args[]) {

int i,j;

for(i=1; i<=5 ; i++) {

for(j=1; j<=i; j++) {

System.out.print("\* ");

}

System.out.println();

}

}

}

Output:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

15] program to print an inverted right triangle [for-loop]

Program:

public class firstProgram{

public static void main(String args[]) {

int i,j,term = 6;

for (i=1; i<=term;i++) {

for(j=term; j>=i ; j--) {

System.out.print("\* ");

}

System.out.println();

}

}

}

Output:

\* \* \* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

16] Program to print all the elements of the array separately. [for-each]

Program:

public class firstProgram{

public static void main(String args[]) {

int array[] = {12,34,56,78,97};

for(int i : array) {

System.out.println(i);

}

}

}

Output:

12

34

56

78

97

17] program to demonstrate and infinite loop. [for-loop]

Program:

public class firstProgram{

public static void main(String args[]) {

for(;;) {

System.out.println("69");

}

}

}

Output:

69……..until terminated

18] program to demonstrate the use of labelled for loop. [labelled for loop]

Program:

public class firstProgram{

public static void main(String args[]) {

int i,j;

aa:

for(i=1;i<=3;i++) {

bb:

for(j=1;j<=3;j++) {

if(i==2 && j==2) {

break aa;

}

System.out.println(i+" "+j);

}

}

}

}

Output:

1 1

1 2

1 3

2 1

## **Programs**

1] program to demonstrate fibonacci series without recursion.

Program:

import java.util.Scanner;

public class firstProgram{

public static void main(String args[]) {

Scanner myCount = new Scanner(System.in);

System.out.println("enter the count of the series to be printed: ");

int count = myCount.nextInt();

int term1 = 0,term2 = 1, term3,i;

System.out.print(term1+" "+term2);

for(i=2;i<=count;i++) {

term3= term1 + term2;

System.out.print(" "+term3);

term1 = term2;

term2 = term3;

}

}

}

Output:

enter the count of the series to be printed:

10

0 1 1 2 3 5 8 13 21 34 55

2] fibonacci series with recursion.

Program:

public class firstProgram{

static int n1=0,n2=1,n3=0;

static void myFibbo(int count){

if(count>=0) {

n3=n1+n2;

n1=n2;

n2=n3;

System.out.print(" "+n3);

myFibbo(count-1);

}

}

public static void main(String args[]) {

int count=10;

System.out.print(n1+" "+n2);

myFibbo(count-2);

}

}

Output:

0 1 1 2 3 5 8 13 21 34 55

3] program to find prime number or not.

Program:

public class firstProgram{

public static void main(String args[]) {

int i,m=0,flag=0;

int n=3; //user input

m = n/2;

//1

if(n==0||n==1) {

System.out.println(n+" it is not a prime number");

}

//2

else {

for(i=2;i<=m;i++) {

if(n%i == 0) {

System.out.println(n+ " is not a prime number");

flag = 1; //true

break;

}

}

//3

if(flag == 0) {

System.out.println(n+ " is a prime number");

}

}

}

}

Output:

3 is a prime number

**8] Object Oriented Programming(OOPs):**

Object means a real world entity like chair,table and car,etc. OOPs is a methodology to design a program using classes and object. It helps simplifies coding by providing concepts such as:

* **Object**:

Any entity that has a state and behaviour is called an object. It can be physical or logical. For example, chair and table. An object is an instance of a class. An object contains a address and takes some space in the memory. Objects can communicate without knowing each others data or code.

* **Class**:

A collection of objects is called class. A class can also be called a blueprint through which we can create individual object. Classes don’t consume any space.

* **Inheritance:**

When one object acquires all the properties and behaviours of its parent object, it is called inheritance. It provides code reusability and also helps us achieve runtime polymorphism.

* **Polymorphism:**

If one task can be performed in different ways, then it is called polymorphism. In java, using method overloading and method overriding we can achieve polymorphism.

Eg:- A dog barks, A cat meows, these are all different methods of speech.

* **Abstraction:**

Hiding internal details and showing functionality is called abstraction. We use interfaces and abstract classes to achieve abstraction.

Eg:- A phone call, we don’t know the internal processing.

* **Encapsulation:**

Binding code and data into a single unit is called encapsulation.

Eg:- A java class is encapsulated.

Apart from the ones mentioned above, there are additional concepts to be known to achieve object oriented design:

* **Coupling:**

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other.

* **Cohesion:**
* **Association:**
* **Aggregation:**
* **Composition:**

<https://www.javatpoint.com/java-oops-concepts> – go back to complete

**9] Objects and Class:**

Object oriented programming is based on the concept of designing a program using objects and classes.

An object in java is a physical and logical entity, but a class in java is just a logical entity only.

**Object:**

An entity that has state and behaviour is known as an object.

Eg:-

A book can be read,

A bag can hold books,

A pen can write words, etc.

It can be physical or logical(tangible and intangible)

An object has 3 characteristics:

* State: represents the data of an object. eg:book,pen,etc.
* Behaviour: represents the behaviour(functionality) of the object.

Eg:- book can be read(read,is a behaviour of the object).

* Identity: An object identity is typically implemented via a unique id. This identification is not visible to the external user, but within JVM, it is used to identify each object uniquely.

Other definitions of object include:

* An object is a real-world entity.
* An object is a runtime entity.
* The object is an entity which has state and behaviour.
* An object is an instance of class.

We can create objects in java by using: new Keyword, newInstance() method, clone() method, deserialisation, factory method, etc.

**Class:**

A class is a group of objects that have some common properties. It is a logical entity and cannot be physical.

A class in java contains:

* Fields
* Methods
* Constructors
* Blocks
* Nested class and interface

**Instance variable**:

A variable that is created within a class but outside the method is called a instance variable. It doesn’t get memory at compile time. It gets memory at runtime when object or instance is created.

**Method**:

A method is like a function used to expose the behaviour of an object. We use method for the code optimisation and reusability.

**“New” keyword:**

The “new” keyword is used to allocate memory at runtime. All objects get memory at heap memory area.

**Examples:**

1] Program to demonstrate - Main with the class.

Program:

public class Student //defining class

{

int id; //defining fields

String name;

public static void main(String args[]) //creating a main method within a class

{

Student s1 = new Student(); //creating an object or instance

System.out.println(s1.id); //accessing member through reference variable

System.out.println(s1.name);

}

}

Output:

0

null

2] program to demonstrate - main outside the class.

Program:

package javatpoint;

class Student1

{

int id;

String name;

}

class Student

{

public static void main(String args[])

{

Student1 s1 = new Student1();

System.out.println(s1.id);

System.out.println(s1.name);

}

}

Output:

0

Null

3] There are 3 ways to initialise a object:

1. **By reference variable:**
   1. Program to initialise an object through reference:

Program:

class Student1

{

int id;

String name;

}

class Student

{

public static void main(String args[])

{

Student1 s1 = new Student1();

s1.id = 1;

s1.name = "rishe";

System.out.println("roll.no "+s1.id+" is "+s1.name);

}

}

Output:

roll.no 1 is Rishe

* 1. Program to create multiple objects and store value in them

Program:

class Student1

{

int id;

String name;

}

class Student

{

public static void main(String args[])

{

Student1 s1 = new Student1();

Student1 s2 = new Student1();

s1.id= 1;

s1.name="Rishe";

s2.id= 2;

s2.name = "Brush";

System.out.println(s1.id+" "+s1.name);

System.out.println(s2.id+" "+s2.name);

}

}

Output:

1 Rishe

2 Brush

1. Initialisation through method:
   1. Program to maintain student records.

Program:

class Student1

{

int rollno;

String name;

void insertRecord(int r, String n)

{

rollno = r;

name = n;

}

void result()

{

System.out.println(rollno +" "+ name);

}

}

class Student

{

public static void main(String args[])

{

Student1 s1 = new Student1();

Student1 s2 = new Student1();

s1.insertRecord(111,"Karan");

s2.insertRecord(222,"Aryan");

s1.result();

s2.result();

}

}

Output:

111 Karan

222 Aryan



Objects get stored in heap memory. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 are reference variables that refer to objects allocated in memory.

* 1. Program to maintain records of rectangle class.

Program:

class Rectangle

{

int length;

int width;

void insert(int l, int w)

{

length=l;

width=w;

}

void calculate()

{

System.out.println("The area of rectangle is: " + length \* width);

}

}

class Student

{

public static void main(String args[])

{

Rectangle r1 = new Rectangle();

Rectangle r2 = new Rectangle();

r1.insert(2, 3);

r2.insert(3, 4);

r1.calculate();

r2.calculate();

}

}

Output:

The area of rectangle is: 6

The area of rectangle is: 12

4] Program to explain how anonymous objects are used.

Program:

class Calculation

{

void fact(int n)

{

int fact = 1;

for(int i=1; i<=n; i++)

{

fact = fact\*i;

}

System.out.println("fact of "+n+" is "+ fact);

}

}

class Student

{

public static void main(String args[])

{

new Calculation().fact(5);

}

}

Output:

fact of 5 is 120

5] Program to demonstrate banking system.

Program:

class Bank

{

int acc\_no;

String name;

float amount;

void insert(int acc, String n, float amt)

{

acc\_no = acc;

n = name;

amount = amt;

}

void deposit(float amt)

{

amount = amount + amt;

System.out.println(amt + " was deposited");

}

void withdraw(float amt)

{

if(amount<amt)

{

System.out.println("insufficient funds!");

}

else

{

amount = amount - amt;

System.out.println("Your new balance is, "+ amount);

}

}

void checkBalance()

{

System.out.println("The balance for the account number: "+acc\_no+" is "+amount);

}

void accountHolderDetails()

{

System.out.println(acc\_no+" "+ name +" "+amount);

}

}

class firstProgram

{

public static void main(String args[])

{

Bank b1 = new Bank();

b1.insert(696969, "Rishe", 100000);

b1.accountHolderDetails();

b1.checkBalance();

b1.deposit(50000);

b1.checkBalance();

b1.withdraw(25000);

System.out.println("Thank you");

}

}

Output:

696969 null 100000.0

The balance for the account number: 696969 is 100000.0

50000.0 was deposited

The balance for the account number: 696969 is 150000.0

Your new balance is, 125000.0

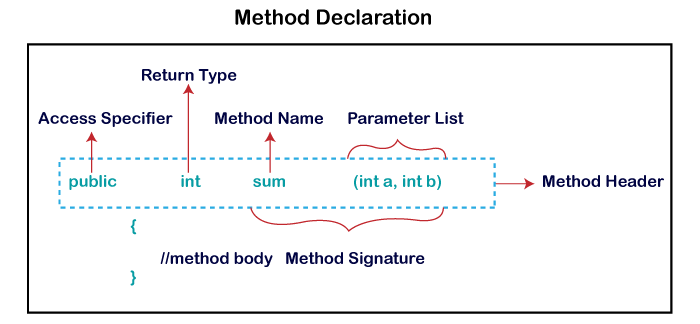
Thank you

**10] Method**

Method is a way in which we can perform certain task. It is a collection of instructions to perform a specific task. It provides the reusability of the code. It is also easy to modify. Method is only executed when called or invoked.

Eg:- main() method.

**Method declaration:**



The method declaration provides us with the features present in the method. Such as:

* Method signature: provides the method name and parameter list.
* Access Specifier: It specifies the visibility of the method. It has 4 types:
  + Public: The method is accessible by all classes.
  + Private: The method is accessible in only the classes it is defined.
  + Protected: The method is accessible within the same package or subclass in different package.
  + Default: When we don’t use any access specifier we use default, the default is only visible to the same package only.
* Return Type: It is the data type the method returns. It can be a data type,object,collection or void. If the return type is of no value then it is void.
* Method name: It is the name of the method that the method corresponds to.
* Method body: Contains the actions performed by the method.
* Parameter list: It contains data types and variable name separated by commas, if no parameters, then it is left blank.

**Method overloading:**

It is possible that one method has the same name as another method in the same class, this is called method overloading.

**Types of methods:**

* **Predefined method**:

Predefined methods are methods that are already defined in the java library. It is also called standard-library method or built-in method. We can use them in out program at any point of the code. Some of the examples are length(), sqrt(), print(), max(), etc.

Every predefined method is defined inside of a class. Such as print() method is defined by java.io.printStream class.

Example:

1. Program to find the maximum between 2 numbers.

Program:

class Student{

public static void main(String args[]) {

System.out.println("the max between the 2 numbers is "+ Math.max(3,5));

}

}

* **User defined methods:**

Methods written by programmers are called user defined methods. These methods are modified according to our requirements.

Eg:- A method that checks the number is even or odd.

Program:

import java.util.Scanner;

public class Student

{

public static void evenOrOdd(int num)

{

if(num%2 == 0)

{

System.out.println("it is even");

}

else

{

System.out.println("it is odd");

}

}

public static void main(String args[])

{

Scanner scan = new Scanner(System.in);

System.out.println("enter the number: ");

int num = scan.nextInt();

evenOrOdd(num);

}

}

* **Static method:**

A method which has a static keyword is called static method. A method that belongs to a class rather than the instance of the class is known as static method.

The main advantage of static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

Eg:- To show the working of static method

Program:

public class Student

{

public static void simplePrint()

{

System.out.println("this is dummy");

}

public static void main(String args[])

{

simplePrint();

}

}

Output:

this is dummy

* **Instance method:**

Method of a class is called instance method. It is non-static method defined in class. Before calling it, we need to create an object of class.

Eg:- Program to see the working of instance method.

Program:

class Student

{

public static void main(String args[])

{

Student class\_method = new Student();

System.out.println("The result is: "+class\_method.sum(23,20));

}

int s;

public int sum(int a,int b)

{

s=a+b;

return s;

}

}

Result:

The result is: 43

There are 2 types of instance methods:

* **Accessor method:**

The variables that read the instance variables are known as Accessor method. It is easy to identify the accessor method as they are prefixed by the word ‘get’, hence they are also know as ‘getters’. It returns the value of the private field.

Eg:-

public getId()

{

return id;

}

* **Mutator method:**

The methods that read the instance variables and modify the values are known as mutator method. These are easy to identify as they are prefixed by the word ‘set’ hence these are also known as ‘setters’ or ‘modifiers’. It does not return any value. It accepts the same parameters as the same data type that depends on the field. It is used to set the private field.

Eg:-

public void setRoll(int roll)

{

this.roll = roll;

}

* **Abstract method:**

The method that does not have a method body, i.e, without an implementation is called an abstract method. To use this we use ‘abstract’ keyword. It means that the class must itself be abstract if it has abstract method.

Eg:- program to demonstrate abstract method.

abstract class Demo //abstract class

{

abstract void display(); //abstract method declaration

}

public class Student extends Demo

{

void display() //method implementation

{

System.out.println("abstract method...");

}

public static void main(String args[])

{

Demo obj = new Student(); //creating object of abstract class

obj.display();

}

}

Output:

abstract method...

**11] Constructors:**

A constructor is a block of code similar to methods. It is called when instance of class is created. At the time of calling constructors, memory for the object is allocated in the memory.

Everytime a object is created with the new() keyword, at least one constructor is called.

It calls a default if no constructor is available in the class.

There are 2 types of constructors:

* No-arg constructor
* Parameterized constructor