**JAVA 2**

Constructor:

Whenever we create objects or instance of a class at that particular moment instance variables of the class are required to initialize. Java provides one inbuilt mechanism by which at the time of object creation the instance variables are initialised, that mechanism is known as constructor.

Constructor are look like method but **doesn't have any return type.** It's name is same as class name.

public class MyClass{

//This is the constructor

MyClass(){

}

..

}

\*\*\*\*Use of Constructor: So constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

Type :

**@ Argumented Constructor:**

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with your own values, then use a parameterized constructor.

// parameterized constructor.

import java.io.\*;

class A

{

// data members of the class.

String name;

int id;

// constructor would initialize data members

// with the values of passed arguments while

// object of that class created.

A(String name, int id)

{

this.name = name;

this.id = id;

}

}

class Demo

{

public static void main (String[] args)

{

// this would invoke the parameterized constructor.

A x = new A("adam", 1);

System.out.println("Name :" + x.name +

" and Id :" + x.id);

}

}

Output:

Name :adam and Id :1

**@ Default Constructor(Non argumented)**

A constructor that has no parameter is known as default constructor. If we don’t define a constructor in a class, then compiler creates default constructor(with no arguments) for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.

Default constructor provides the default values to the object like 0, null, etc. depending on the type.

// no-argument constructor

class A

{

int num;

String name;

// this would be invoked while an object

// of that class is created.

A()

{

System.out.println("Constructor called");

}

}

class Demo

{

public static void main (String[] args)

{

// this would invoke default constructor.

A x = new A();

// Default constructor provides the default

// values to the object like 0, null

System.out.println(x.name);

System.out.println(x.num);

}

}

Output :

Constructor called

null

0

**@ Copy Constructor:**

In Java, a copy constructor is a special type of constructor that creates an object using another object of the same Java class. It returns a duplicate copy of an existing object of the class.

class A

{

int a,b;//instance variable

A(int a,int b)//argument constructor

{

this.a=a;

this.b=b;

System.out.println("Argument Constructor");

}

A(A x)//copy constructor

{

this.a=x.a;

this.b=x.b;

System.out.println("copy constructor");

}

public static void main(String args[])

{

A m=new A(12,13);

A n=new A(m);

System.out.println(m.a);

System.out.println(m.b);

System.out.println(n.a);

System.out.println(n.b);

}

}

**When is a Constructor called ?**

Each time an object is created using new() keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the data members of the same class.

A constructor is invoked at the time of object or instance creation. For Example:

class A

{

.......

// A Constructor

A() {}

.......

}

// We can create an object of the above class

// using the below statement. This statement

// calls above constructor.

A obj = new A();

**This Constructor:**

The this keyword refers to the current object in a method or constructor.

The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

If you omit the keyword in the example above, the output would be "0" instead of "5".

public class Main {

int x;

// Constructor with a parameter

public Main(int x) {

this.x = x;

}

// Call the constructor

public static void main(String[] args) {

Main myObj = new Main(5);

System.out.println("Value of x = " + myObj.x);

}

}

**Rules for writing Constructor:**

Constructor(s) of a class must has same name as the class name in which it resides.

A constructor in Java can not be abstract, final, static and Synchronized.

Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

**Does constructor return any value?**

There are no “return value” statements in constructor, but constructor returns current class instance. We can write ‘return’ inside a constructor.

**How constructors are different from methods in Java?**

Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.

Constructor(s) do not return any type while method(s) have the return type or void if does not return any value.

Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.

**Constructor Overloading**

Like methods, we can overload constructors for creating objects in different ways. Compiler differentiates constructors on the basis of numbers of parameters, types of the parameters and order of the parameters.

// Java Program to illustrate constructor overloading

// using same task (addition operation ) for different

// types of arguments.

class A

{

// constructor with one argument

A(String name)

{

System.out.println("Constructor with one " +

"argument - String : " + name);

}

// constructor with two arguments

A(String name, int age)

{

System.out.println("Constructor with two arguments : " +

" String and Integer : " + name + " "+ age);

}

// Constructor with one argument but with different

// type than previous..

A(long id)

{

System.out.println("Constructor with one argument : " +

"Long : " + id);

}

}

class Demo

{

public static void main(String[] args)

{

// Creating the objects of the class named A

// by passing different arguments

// Invoke the constructor with one argument of

// type 'String'.

A x= new A("raj");

// Invoke the constructor with two arguments

A y = new A("rahul", 10);

// Invoke the constructor with one argument of

// type 'Long'.

A z= new A(325614567);

}

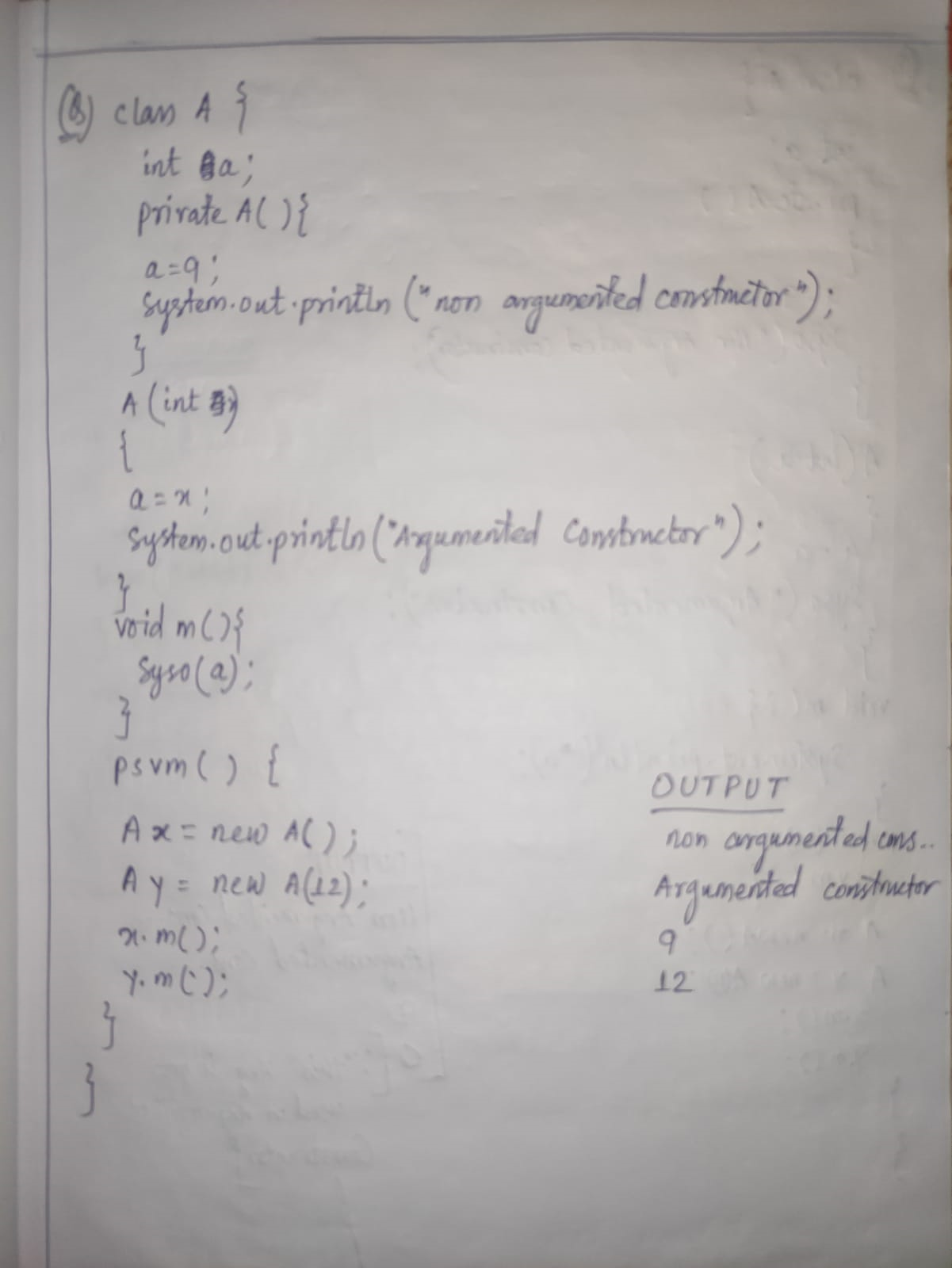
}

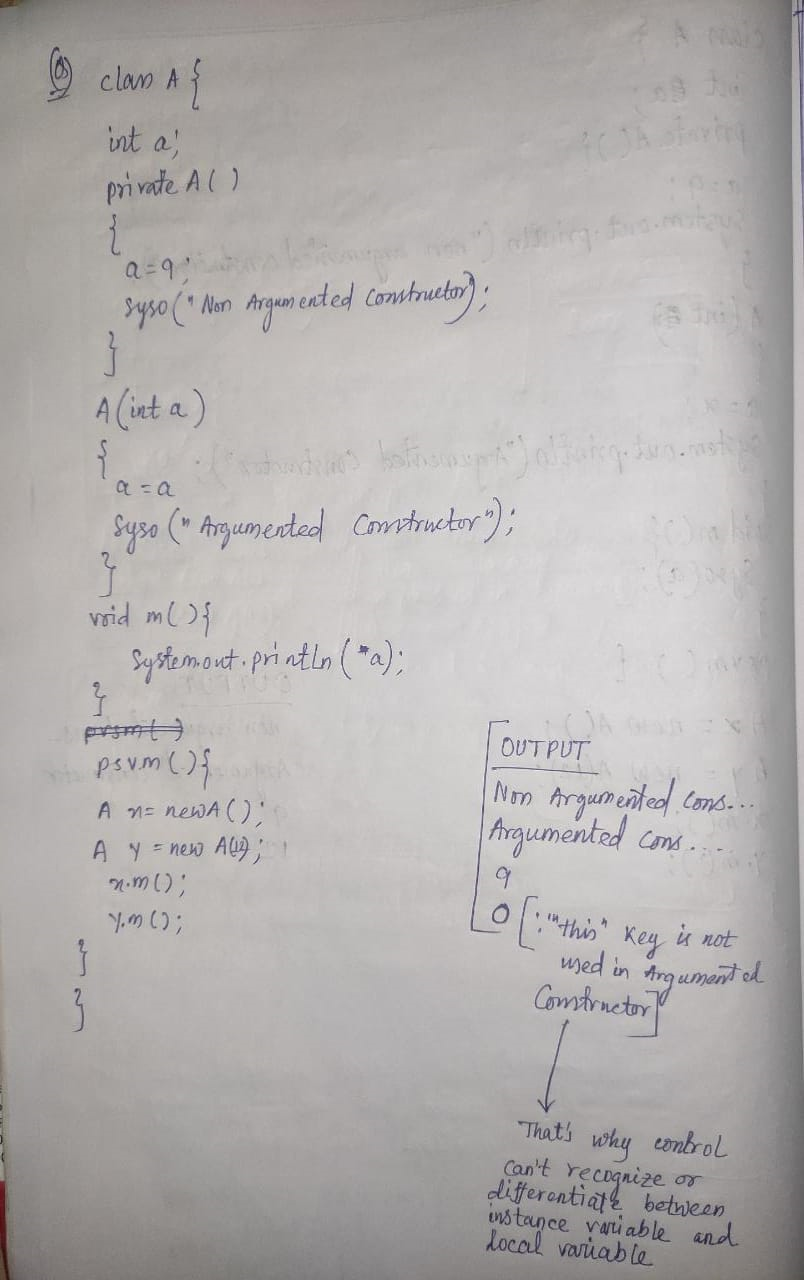
Output:

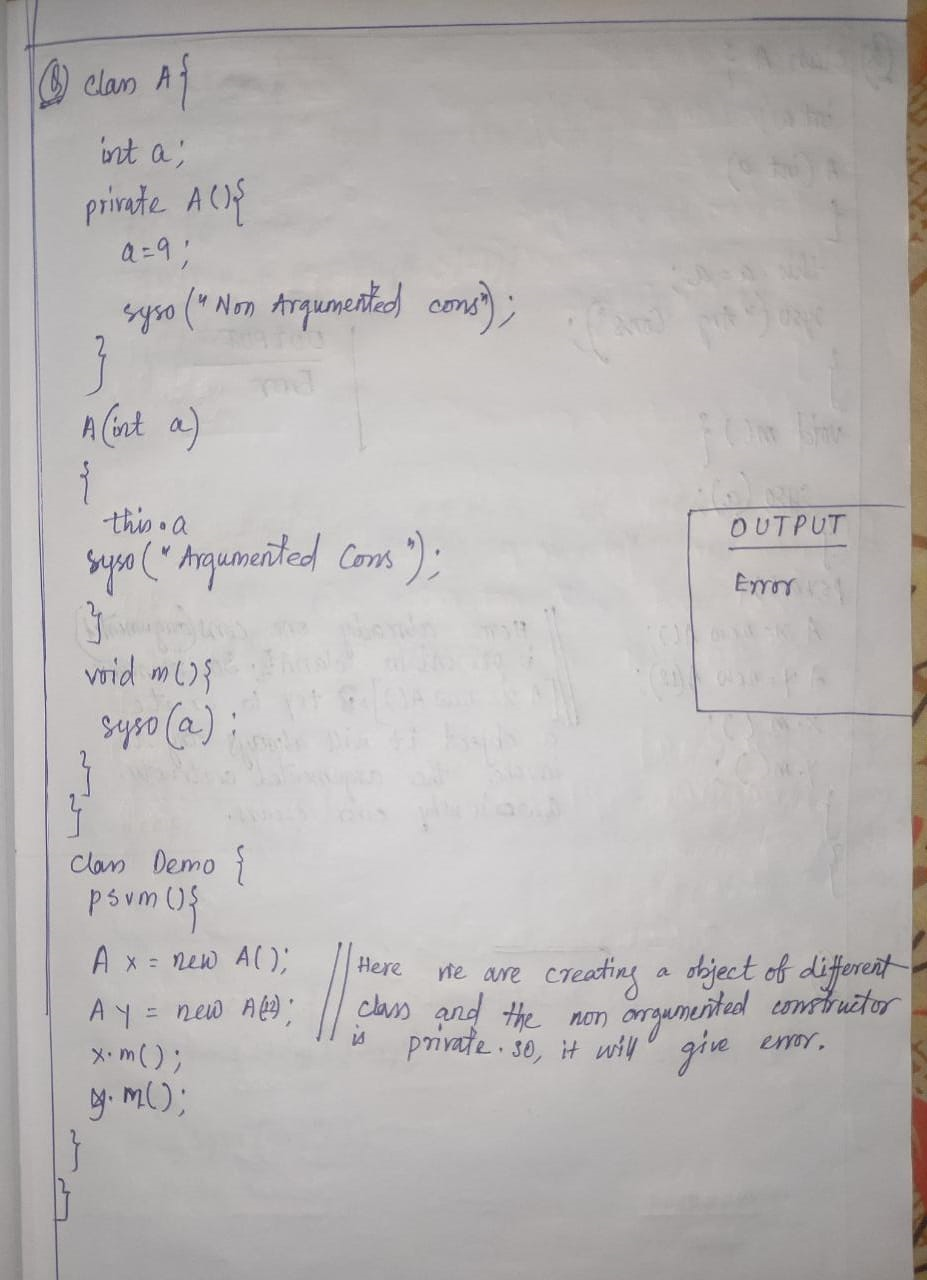
Constructor with one argument - String : raj

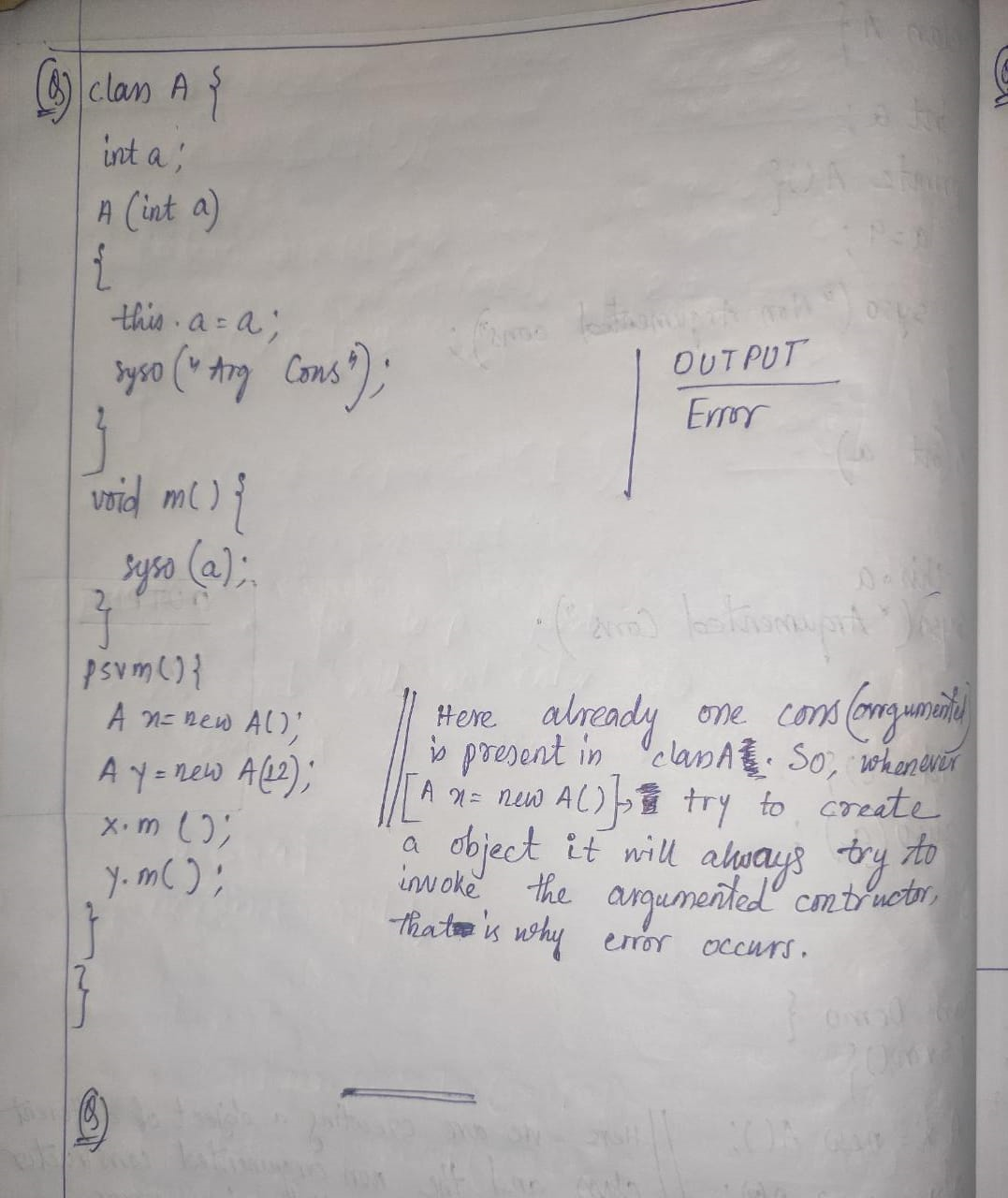
Constructor with two arguments - String and Integer : rahul 10

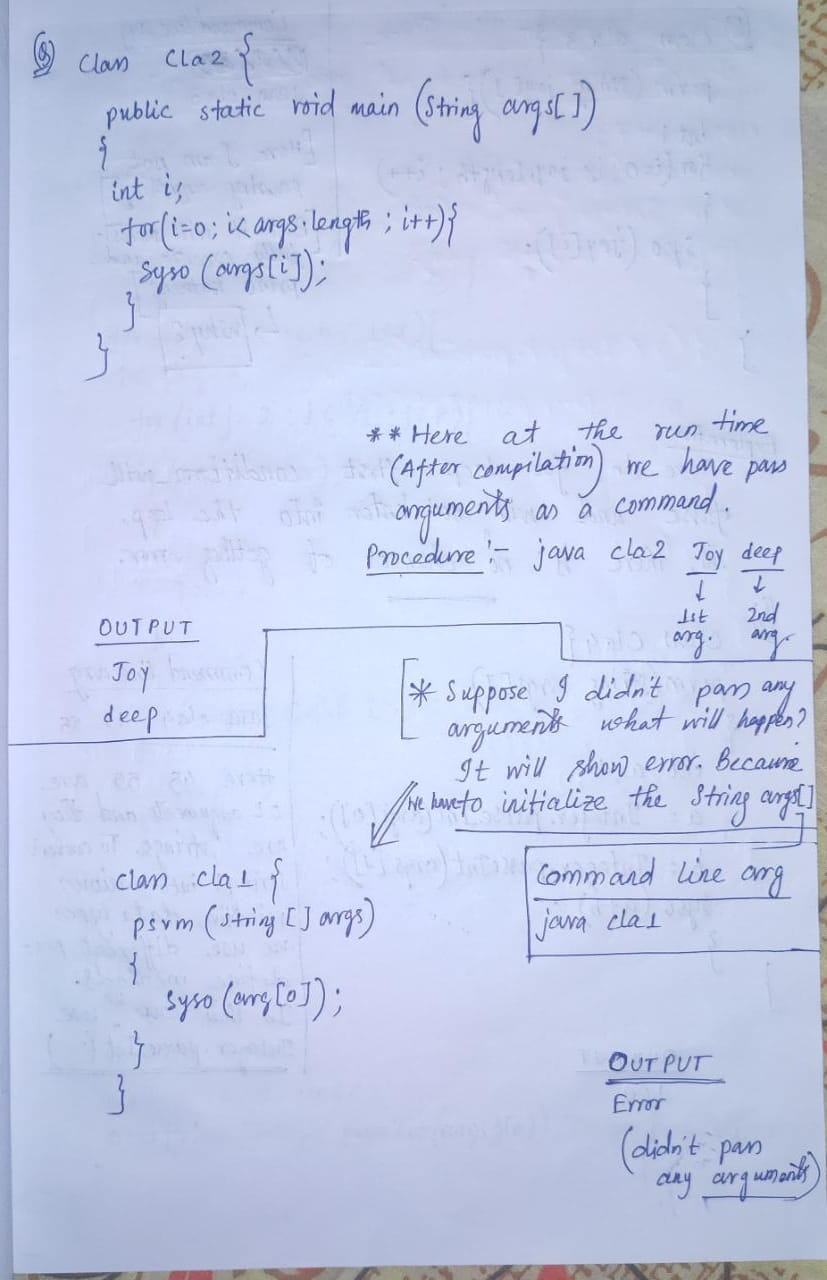
Constructor with one argument - Long : 325614567

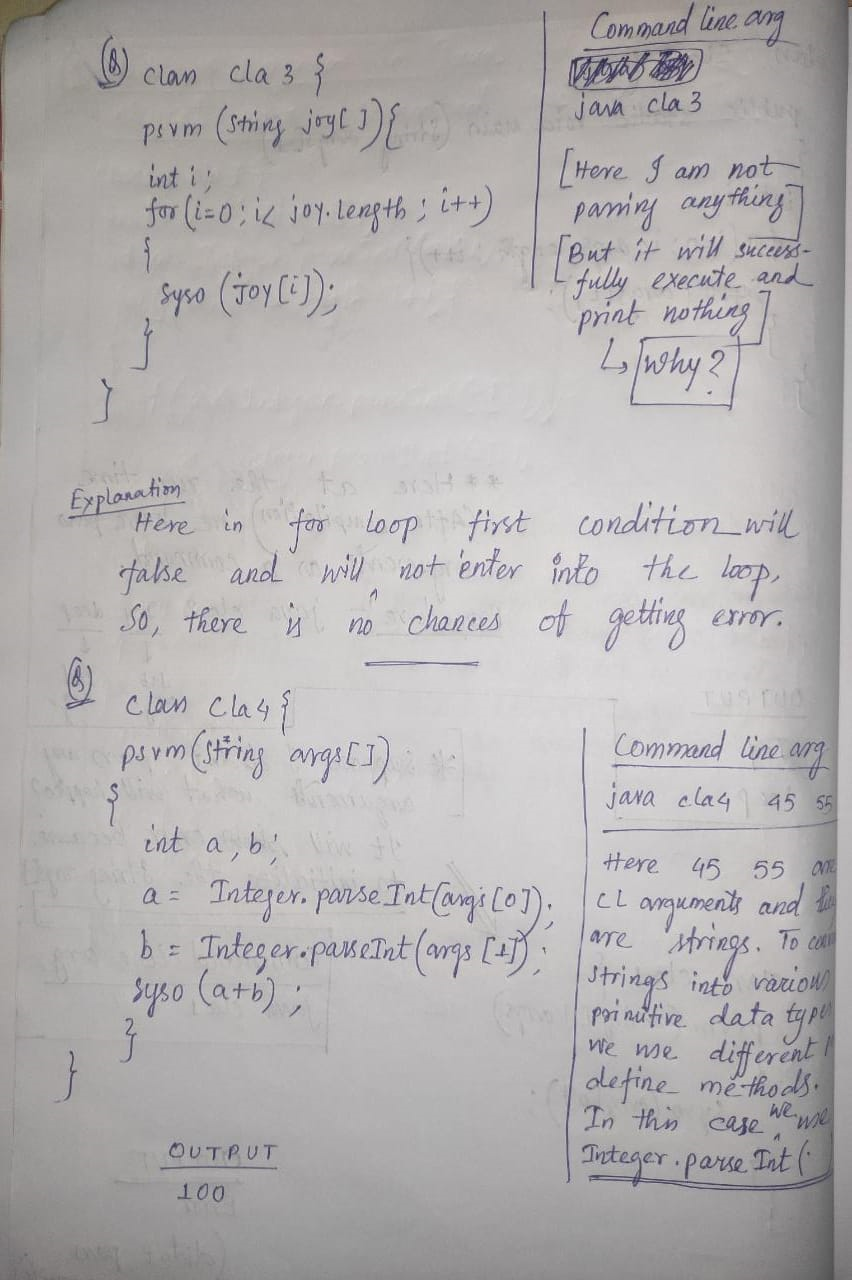












Command Line arguments:

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behaviour of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

All the command line arguments are by default String Arguments.

**Simple example of command-line argument in java**

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

class CommandLineExample{

public static void main(String args[]){

System.out.println("Your first argument is: "+args[0]);

}

}

compile by > javac CommandLineExample.java

run by > java CommandLineExample king

Output: Your first argument is: king

**Example of command-line argument that prints all the values**

In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

class A{

public static void main(String args[]){

for(int i=0;i<args.length;i++)

System.out.println(args[i]); }

}

compile by > javac A.java

run by > java A king messi 10

Output:

king

messi

10

**Addition of two integer numbers by using command line arguments**

class sum

{

public static void main(String ar[])

{

int x,y,s;

x=Integer.parseInt(ar[0]);

y=Integer.parseInt(ar[1]);

s=x+y;

System.out.println("sum of " + x + " and " + y +" is " +s);

}

}

run by > java sum 10 2

sum of 10 and 2 is 12

