**Java Day 1**

1. **What is object**

The object is an real word entity which has state and behavior.

The object is an instance of a class.

Objects are real world entity which have 4 characteristics – attribute, behaviour, state and identity

* Attributes defines the object.
* Behaviour defines how object is reacting with the system.
* State is the property which can able to take different value with respect to time.
* Identity is the unique property by which we can able to distinguish between two objects.

1. **What is class?**

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

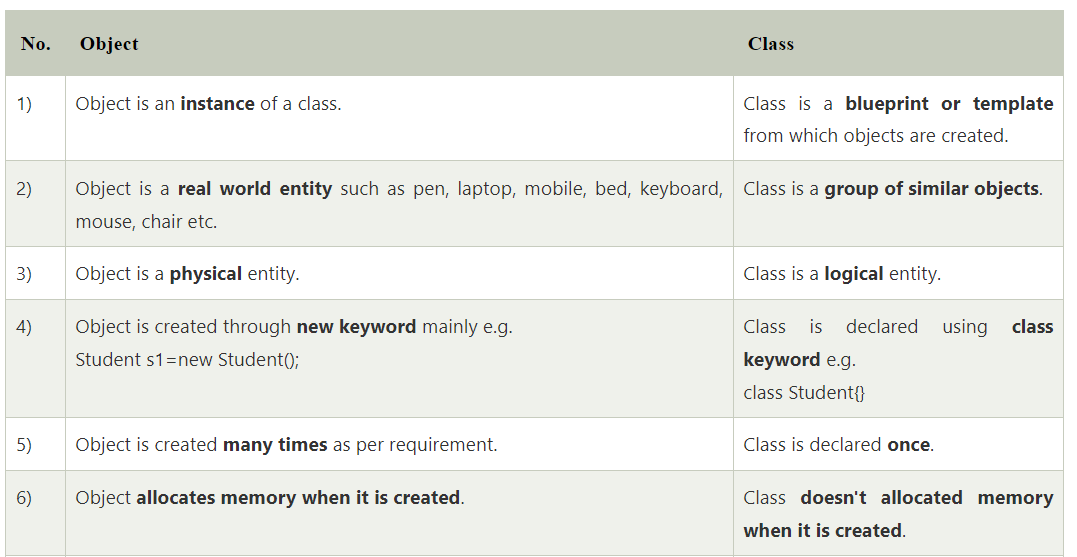
A class in Java can contain:

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

If we declare a variable within class, the variable is known as instance variable. And this instance variables are the attributes of an object.

If we declare a method within class, the method is known as instance method. And this instance methods are the behaviours of an object.

1. **Difference between object and class**

****

1. **Properties of object**

Attribute, behaviour, state and identity

* Attributes defines the object.
* Behaviour defines how object is reacting with the system.
* State is the property which can able to take different value with respect to time.
* Identity is the unique property by which we can able to distinguish between two objects.

1. **Relationship between object and class.**

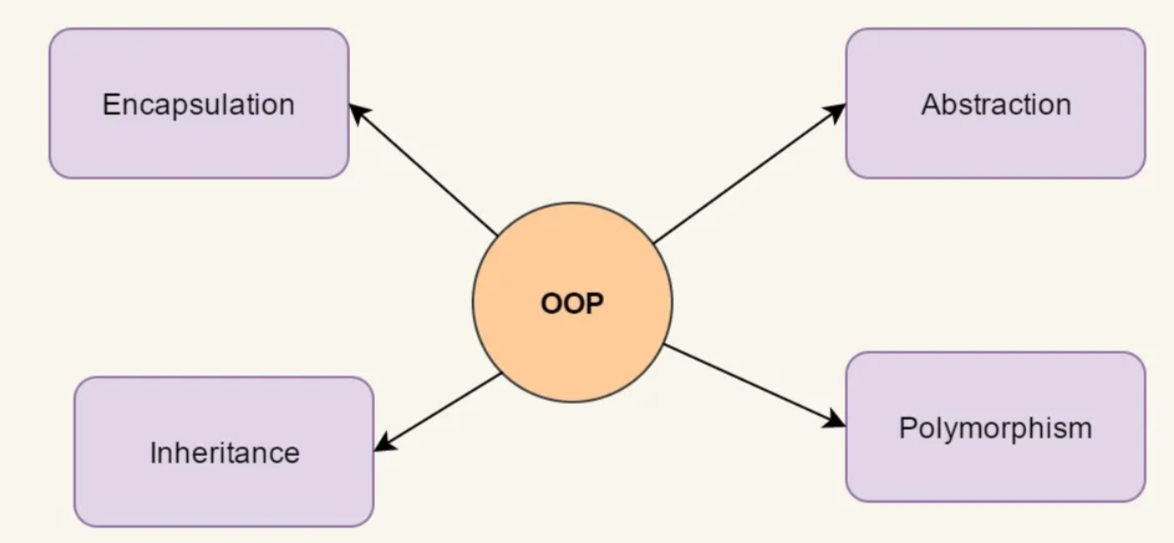
Object is an instance of a class and class is a blueprint from which the objects are created.

1. **What ia object oriented programming.**

**Is Java a pure object oriented programming? Explain in details.**

* Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
* **Reason why java is not 100% object oriented.**
* **Existence of Primitive data types**: int, char, float etc this primitive data types are predefined and present in Wrapper class. So, this are not objects.
* **Use of Static**: Whenever a class is declared static, it can be used **without an object** similarily, we can't call a static function or a static variable using a dot(.) operator or object which defies OOP principles.

1. **Properties of oops (basic principles of oops).**

****

Abstraction

*Hiding internal details and showing functionality* is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

### **Encapsulation**

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

Abstraction-> result. Encapsulation-> reason

### **Inheritance**

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism**

It allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

* **Compile-time Polymorphism:** Inheritance is not required.
* **Runtime Polymorphism:** Inheritance is required.

With out implementing run time polymorphism use of inheritance is nothing but misuse of inheritance.

1. **Define a ceiling fan using the basic 4 principle of OOPs.**

Ans:

* Abstraction: we don't know the inner mechanism of fan but we just know the speed, colour etc of a fan.
* Encapsulation: all the equipments of a fan like: wire, motor etc are wrapped in that object (Fan).
* Inheritance: This fan is being made by inheriting previous types of fans properties.
* Polymorphism: With the help of the features of this fan we can generate different types of fans.

1. **Why Java is platform independent?**

Java provides Platform Independence by making use of Java Byte Code. Java Byte Code or .class file is generated during the compilation of the source code. This Byte Code is platform-independent and can run on any system regardless of the platform it is built upon.

1. **Why JVM is platform dependent?**

The **JVM** is not platform independent. Java Virtual Machine (JVM) provides the environment to execute the java file(. Class file). So at the end it depends on  **kernel** , and kernel is differ from OS (Operating System) to OS. The JVM is used to both translate the **bytecode** into the machine language for a particular computer, and actually execute the corresponding **machine-language** instructions as well. Without the JVM, you can’t run a Java application.

1. **Why we install jdk over jvm ..as jvm is needed to run the program?**

****

**A. JDK** (Java Development Kit) is a Kit that provides the environment to **develop and execute(run)** the Java program. JDK is a kit(or package) that includes two things

* Development Tools(to provide an environment to develop your java programs, Like: Compiler).
* JRE (to execute your java program).

**B. JRE** (Java Runtime Environment) is an installation package that provides an environment to **only run(not develop)** the java program(or application)onto your machine.

**C. JVM:** Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an [**i*nterpreter***](https://www.geeksforgeeks.org/compiler-vs-interpreter-2/)**.**

**So, by default JVM is a part of JDK. That’s why we install JDK instead of JVM.**

1. **Write a normal Java prog and describe each and everything.**

**class** Simple{

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

     System.out.println("Hello Java");

   }

}

* **class** keyword is used to declare a class in Java.
* **public** keyword is an access modifier that represents visibility. It means it is visible to all. And the method can be accessed anywhere.
* **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
* **void** is the return type of the method. It means it doesn't return any value.
* **main** represents the starting point of the program.
* **String[] args** or **String args[] it is an array of String type** is used for [command line argument](https://www.javatpoint.com/command-line-argument).
* **System.out.println(" "); here, System is a predefined class of "java.lang" package. Out is a static reference variable of System class - which is used to hold the method. println() is a method which is to print anything in new line.**

String class is a predefined class which is available in "java.lang" package by using this we can use any methods of String class . And one more thing "java.lang" package imported by default in any program.

Scanner is also a predefined class which is available in “java.util” package, by using we can take input.

To take input by scanner class we have create object of a scanner class and pass ‘System.in’ as a parameter. Here ‘in’ is the static reference variable of a System class.

[Scanner sc = new Scanner(System.in)]

1. **Can we save the jave file with different name of class name where the main method is present?**

In a file there may be multiple class. There should be minimum zero public class and and maximum one public class. If one public class is present then we have to save the file with that public class name. If no public class is there we can save the file name as per our choice .

1. **Why we generally write a class name starting the first alphabet in capital?**

In java all predefined classes which are available in predefined packages all are started with capital letters. So when we create user defined class we should maintain this convention. (Not mandatory).

1. **WHY DO WE NEED WRAPPER CLASS ?**

A **Wrapper class** is a class which contains the**primitive data types** (**int, char, short, byte, etc)**. In other words, wrapper classes provide a way to use **primitive data types** **(int, char, short, byte, etc) as objects**. These wrapper classes come under**java.util package**.

1. **What will happen if use public static void main() instead of public static void main(String[] args).**

the program will compile successfully but it will not execute. It will give run-time error.

\*\*\* If we declare any methods or variables in a class , then they are known as instance variables or instance methods. And if we want to access those instance members we have to create objects.

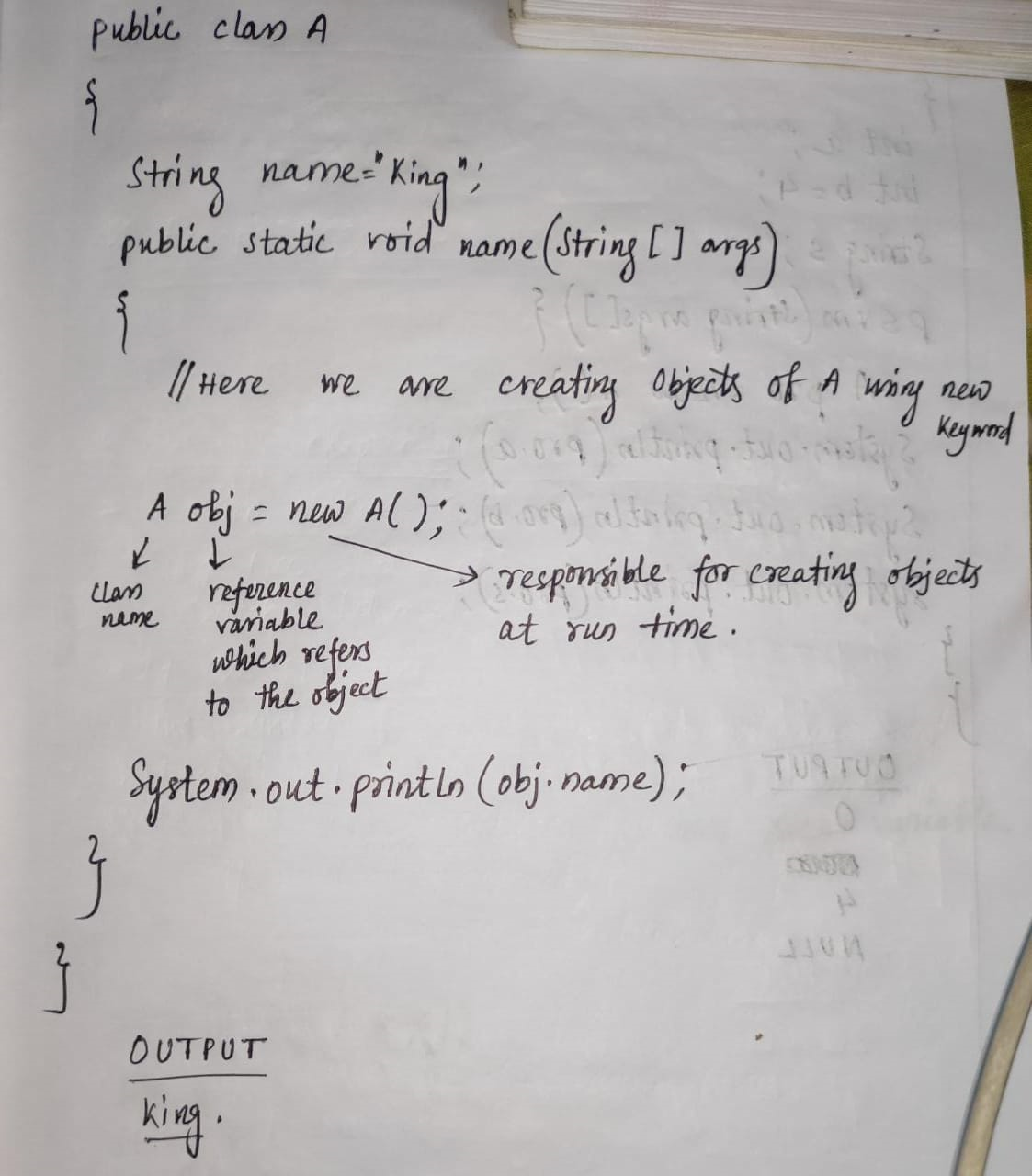
\*\*\* Class acts as a container where we store attributes and behaviour of an object.

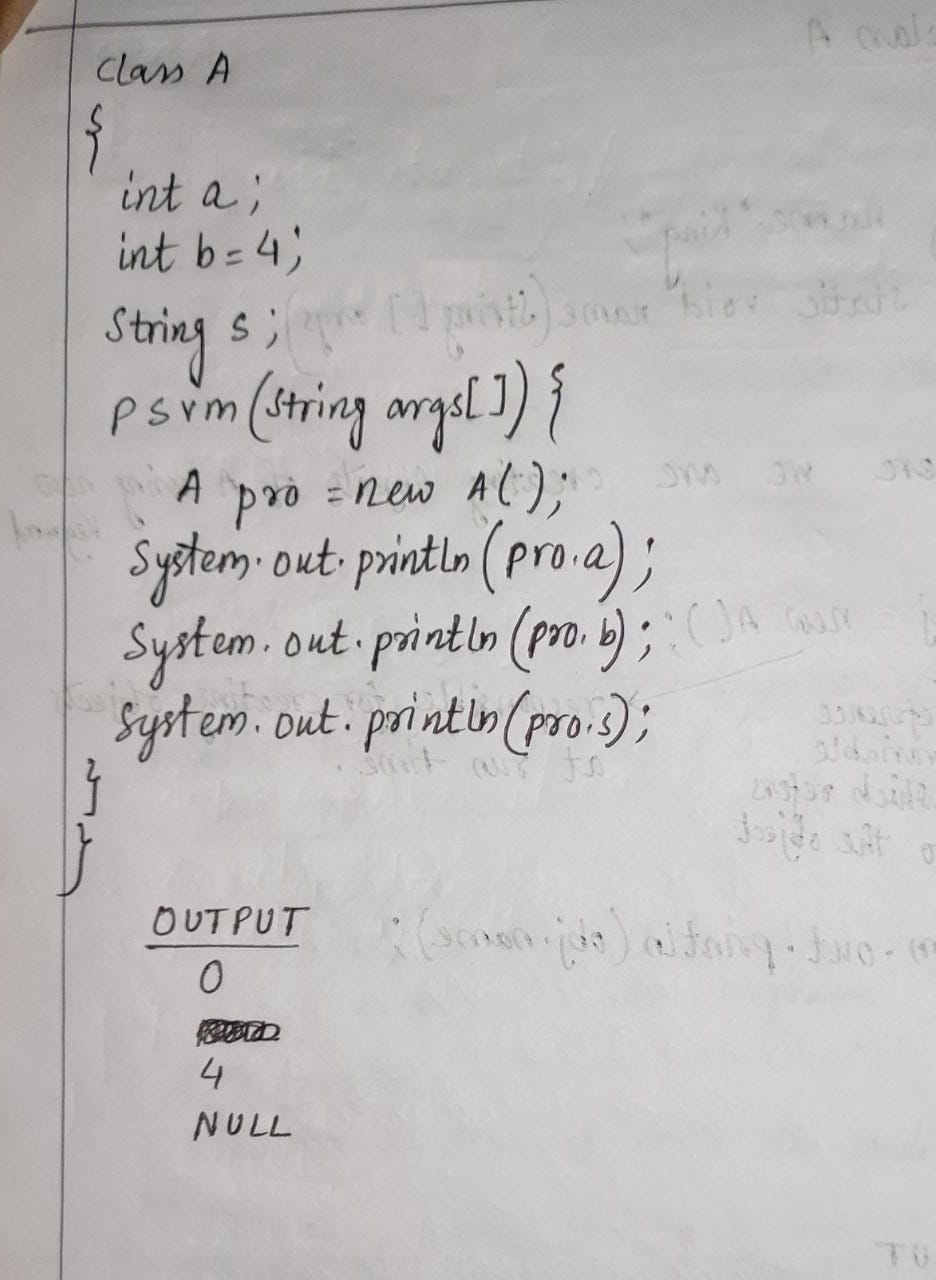
\*\*\* new key word is used to create objects.

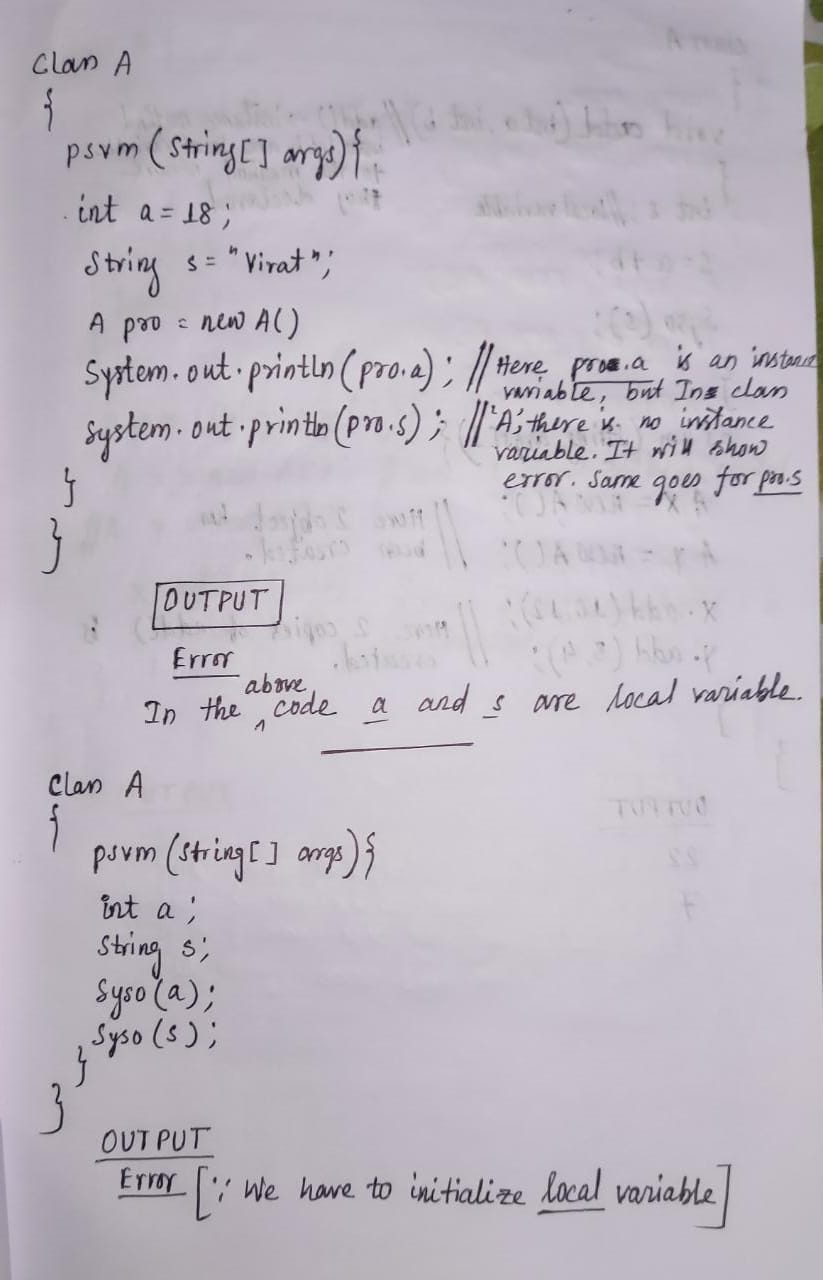
\*\*\* Constructor is used to initialize the instance variables. If we don't initialise the instance variables by default "default constructor" will create and initialise the instance variables with default value(0).

Also we can initialise the instance variables by providing value.

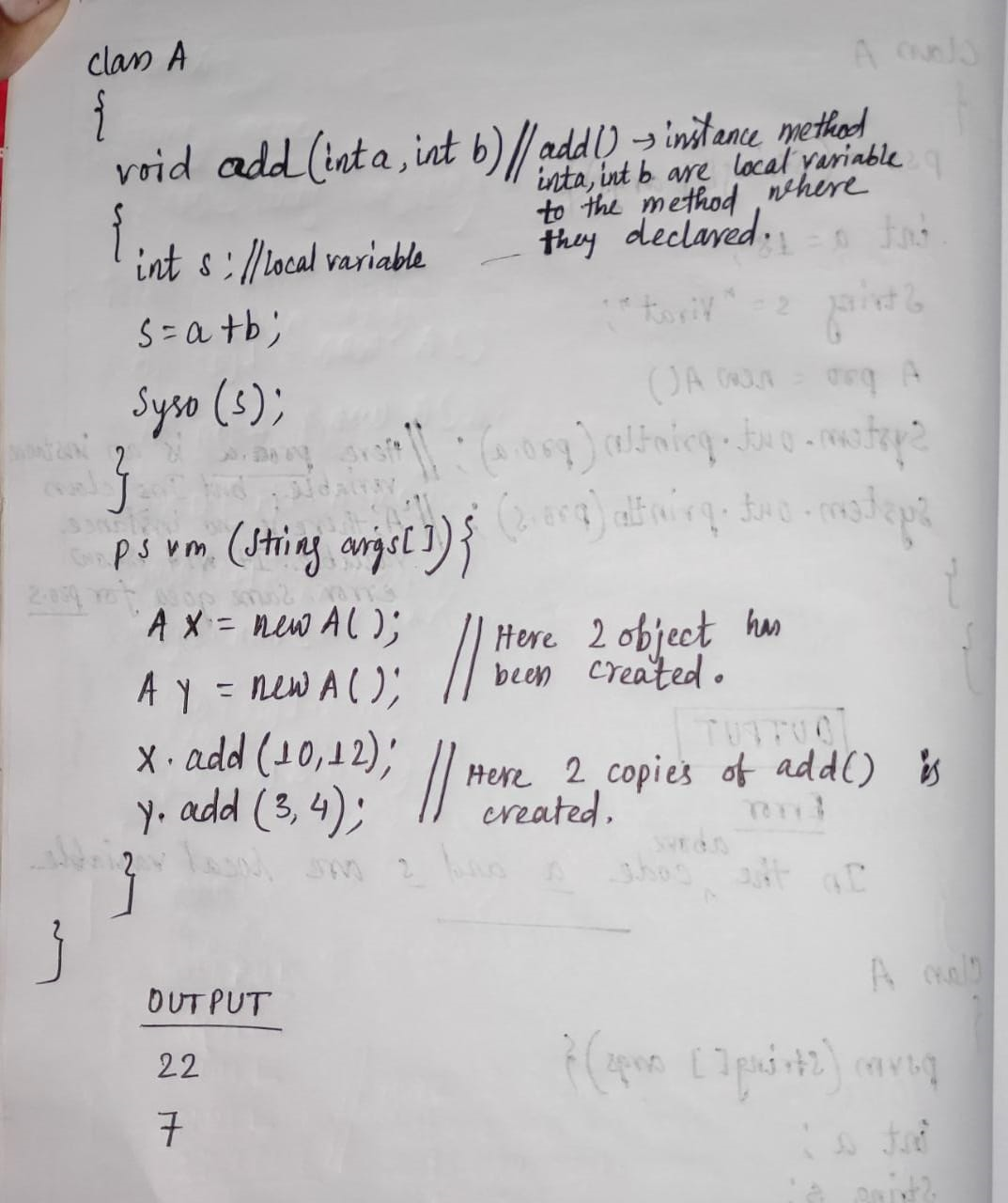
Example:



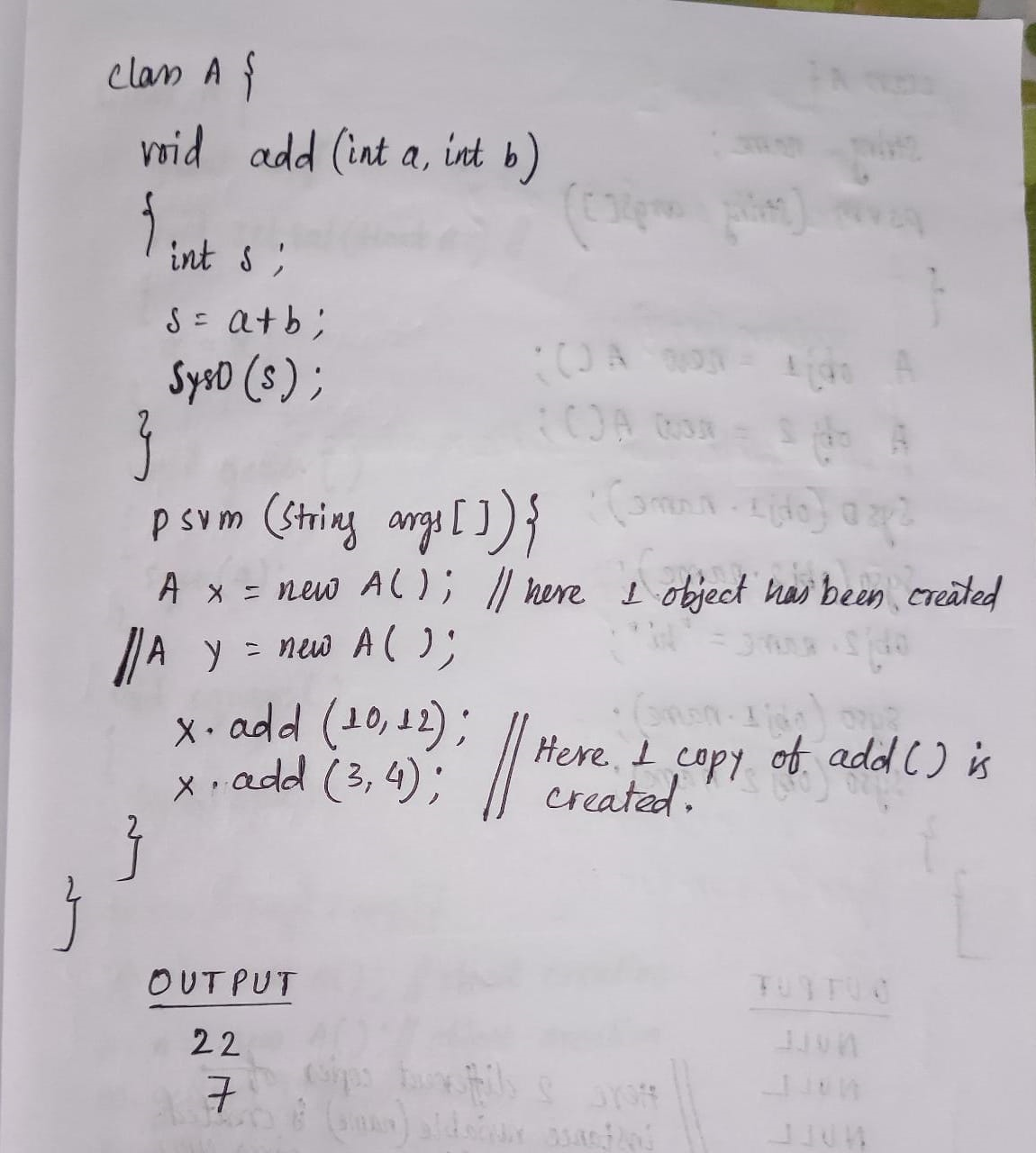


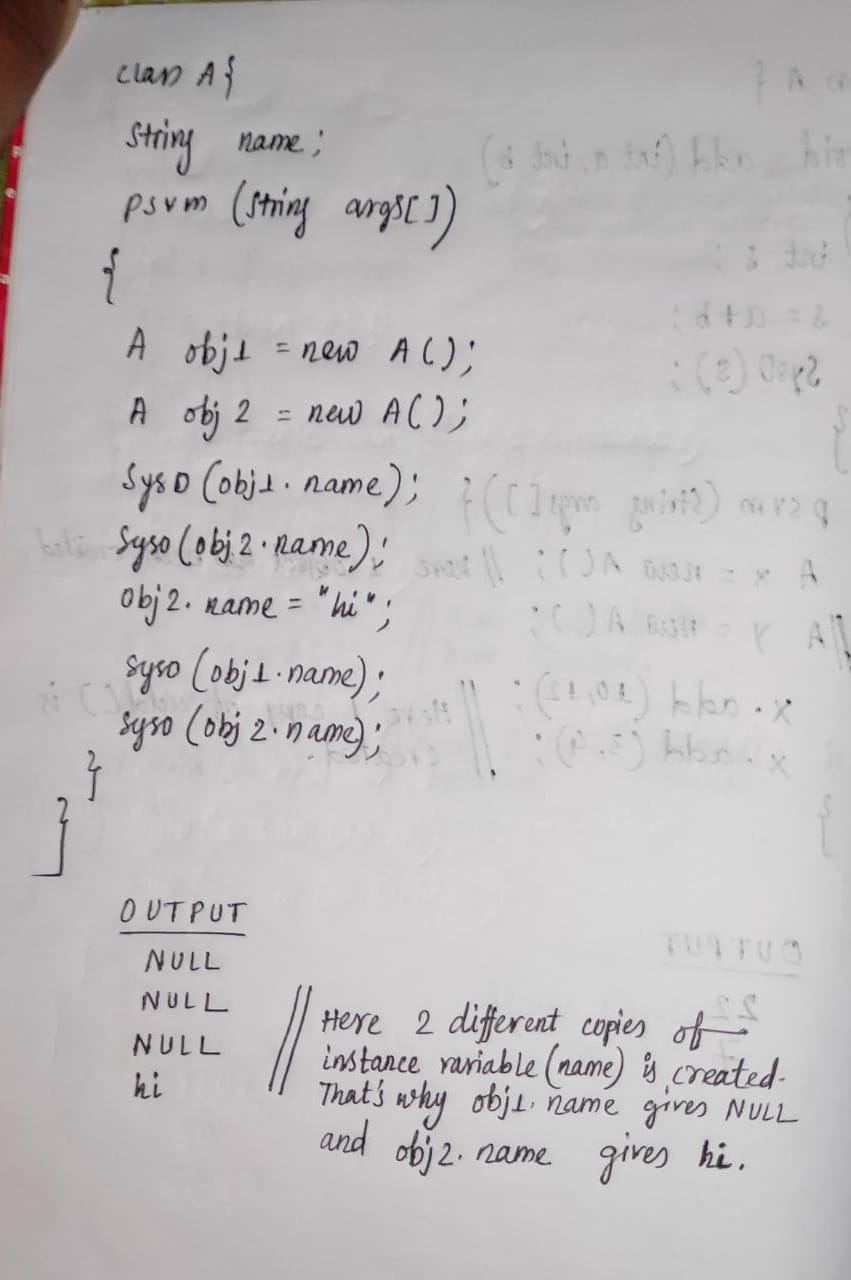


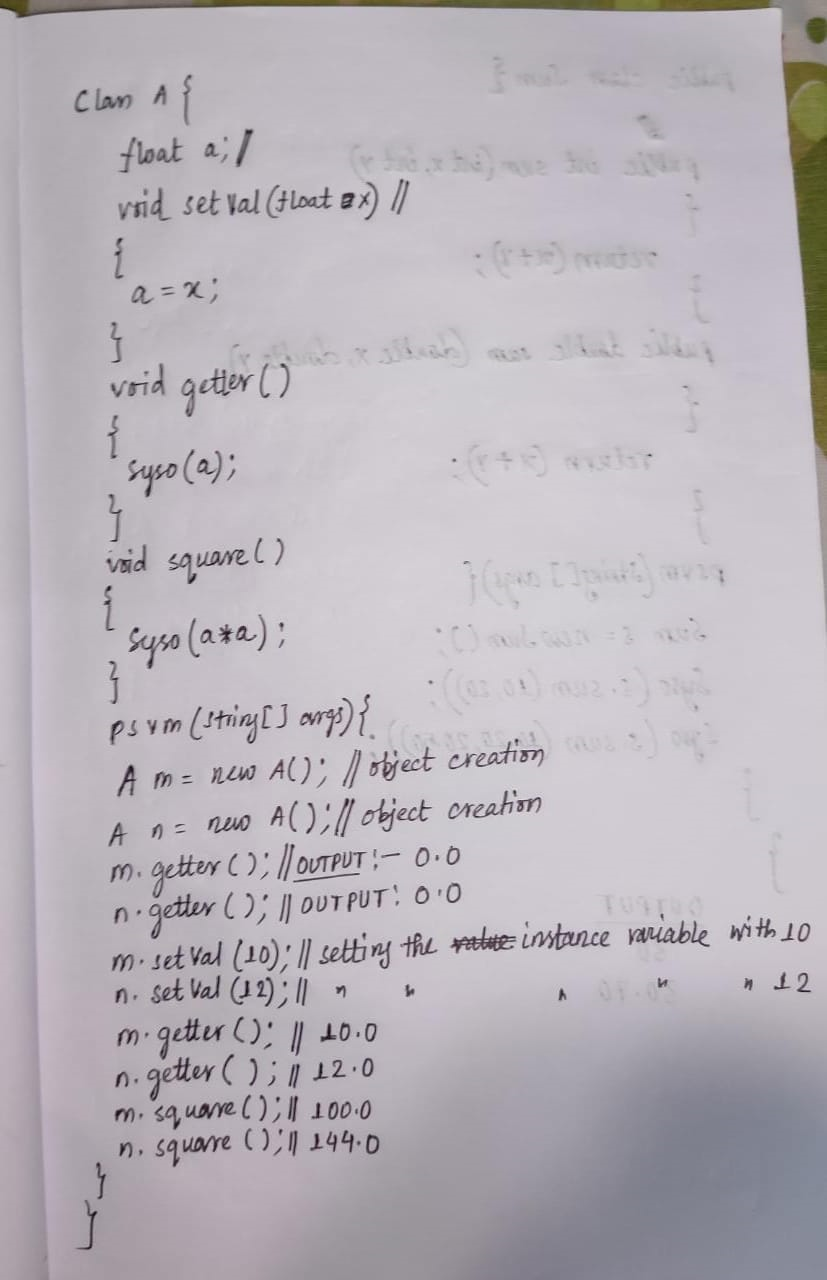
\*\*\* For a particular class we can create n number of instances. For every instance of a class separate copy of instance member will create.



\*\*\* We can able to invoke one member of a class n number of times as per our requirement by using same object reference.







* **Method Overloading:**

With in a class if we define more than one method with same name but different signature that indicates method overloading.

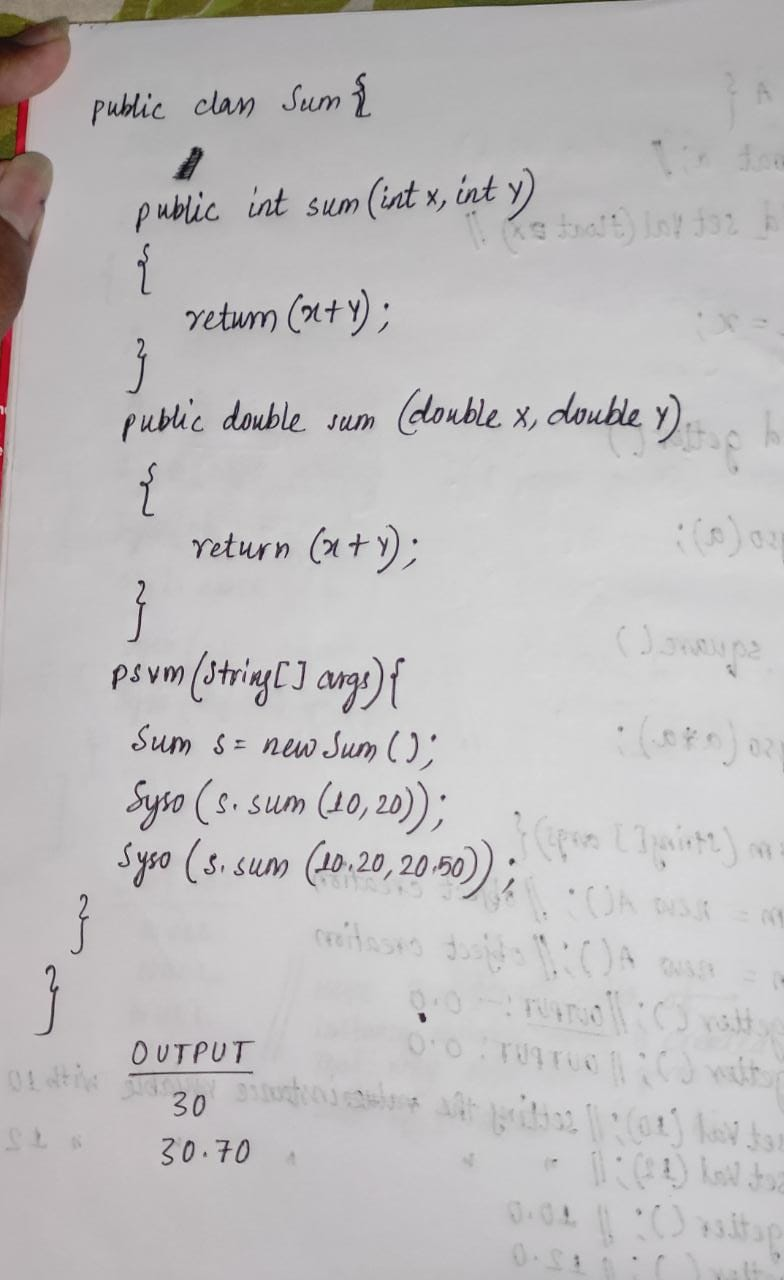
* **Method signature:**

It indicates type of arguments or sequence of arguments or no of arguments.

\*\* We can create multiple methods in a same class with same name with different signature.

* **Why method overloading is an example of compile time polymorphism?**

Method overloading is a compile time polymorphism. Because at the time of compilation of a program, we can able to identify which method has been called by seeing the signature of a method.



Example:2

public class Sum {

// Overloaded sum(). This sum takes two int parameters

public void sum(int x, int y)

{

int s;

s=x+y;

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

}

public void sum(int x, double y)

{ double s;

s=x+y;

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

}

public void sum(double x, int y)

{ double s;

s=x+y;

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

}

// Overloaded sum(). This sum takes three int parameters

public void sum(int x, int y, int z)

{

int s;

s=x+y+z;

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

}

// Overloaded sum(). This sum takes two double parameters

public void sum(double x, double y)

{

double s;

s=x+y;

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

}

public void sum(float x, float y)

{

float s;

s=x+y;

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

}

public static void main(String args[])

{

Sum s = new Sum();

s.sum(7,6,2);

s.sum(10, 20);

s.sum(10.20, 30);

s.sum(10, 20.50);

s.sum(4.5f,7.8); // As there is no method for sum(float, double) it will invoke the sum(double, double) method.

}

}