JAVA Handwritten Notes

JAVA

Date ___ / ___ / _____

Ques What is Java ?

→ Java is a high-level object oriented programming language.

→ Java is a platform because java has its own runtime environment (JRE).

Ques. Editions of Java :-

① J2SE : Java 2 Standard Edition (Core Java)
② J2EE : Java 2 Enterprise Edition (Advance Java)
③ J2ME : Java 2 Micro Edition (replace by Android)

J2SE : → In this edition we learn java fundamental
Project → Standalone Application (execute on single machine)

J2EE : → In this edition we learn server-side programming (work on multiple machines)
Project → Enterprise Application (Banking)

J2ME : → In this edition we learn micro programming
Project → Mobile Based Application

Java History :-

→ Company :- Sun Microsystems (ORACLE)
→ Designed By :- James Gosling & his team
→ Idea :- Simple, platform Independent (work on all machine), tight coding (less number of lines)
→ Name :- Oak (Java)
→ Year :- 1991 (development) → 1995 (release)
→ Symbol :- Coffee
→ Long Term Service (LTS) Version :- Java 8, Java 11

Date ___/___/_____

Ques. What is the difference between C, C++ & Java ?

→ Programming Paradigm (Style) :-

C : Procedural language

C++ : OOP

Java : OOP

→ Origin :-

C : Based on Assembly language

C++ : Based on C language

Java : Based on C & C++

→ Translator :-

C : Compiler

C++ : Compiler

Java : Interpreter & Compiler

→ Static / Dynamic Programming languages

C : Static P.L

C++ : Static P.L

Java : Dynamic P.L

Static P.L :- In static programming language, the primitive data types allocates memory at the compile time.

Dynamic P.L :- In dynamic programming language, the primitive data types allocates memory at the run time.

Primitive data types :- byte, int, short, long, char, double

→ **Pre-Processor directives :-**

C: Support header files (# include, # define)

C++: Support header files (# include, # define)

Java: Use packages (import)

⇒ Java support packages that is why Java does not require pre-processor.

→ **Platform :-**

C: Platform dependent

C++: Platform dependent

Java: Platform independent

→ call by value / Reference

C: Support both

C++: Support both

Java: Support only call by value

→ **Inheritence :-**

C: No inheritence

C++: Support inheritence

Java: Support inheritence but not multiple inheritence

→ **Overloading :-**

C: Does not support

C++: Support overloading

Java: Support overloading but not operator overloading

Ques:- What is static and dynamic programming language?

Ques:- How java is dynamic programming language?

Ques:- Why java does not require pre-processor?

Ques:- How java is platform independent?

Ques: Why java does not support pointers ?

→ Java is dynamic programming language it means primitive data types allocates memory at the run time whereas pointer support static memory allocation while java does not support.

Ques. What is call by reference and call by value ?

→ call by reference :- When we call the function and provide the address of that variable that is known as call by reference.

→ call by value :- When we call the function and provide the value of that variable that is known as call by value.

Ques. Why java has good memory management as compared to C & C++ ?

Because of Java Garbage Collector

Package Statement :-

→ It is a group of similar type of classes or interfaces or packages

→ Advantages of package :-

- It prevents naming conflicts of classes or interfaces
- Easy to maintain
- Accessability to classes or interfaces can be controlled
- Reusability

→ Types of Packages :-

→ Pre-defined Package - lang, util, awt, swing

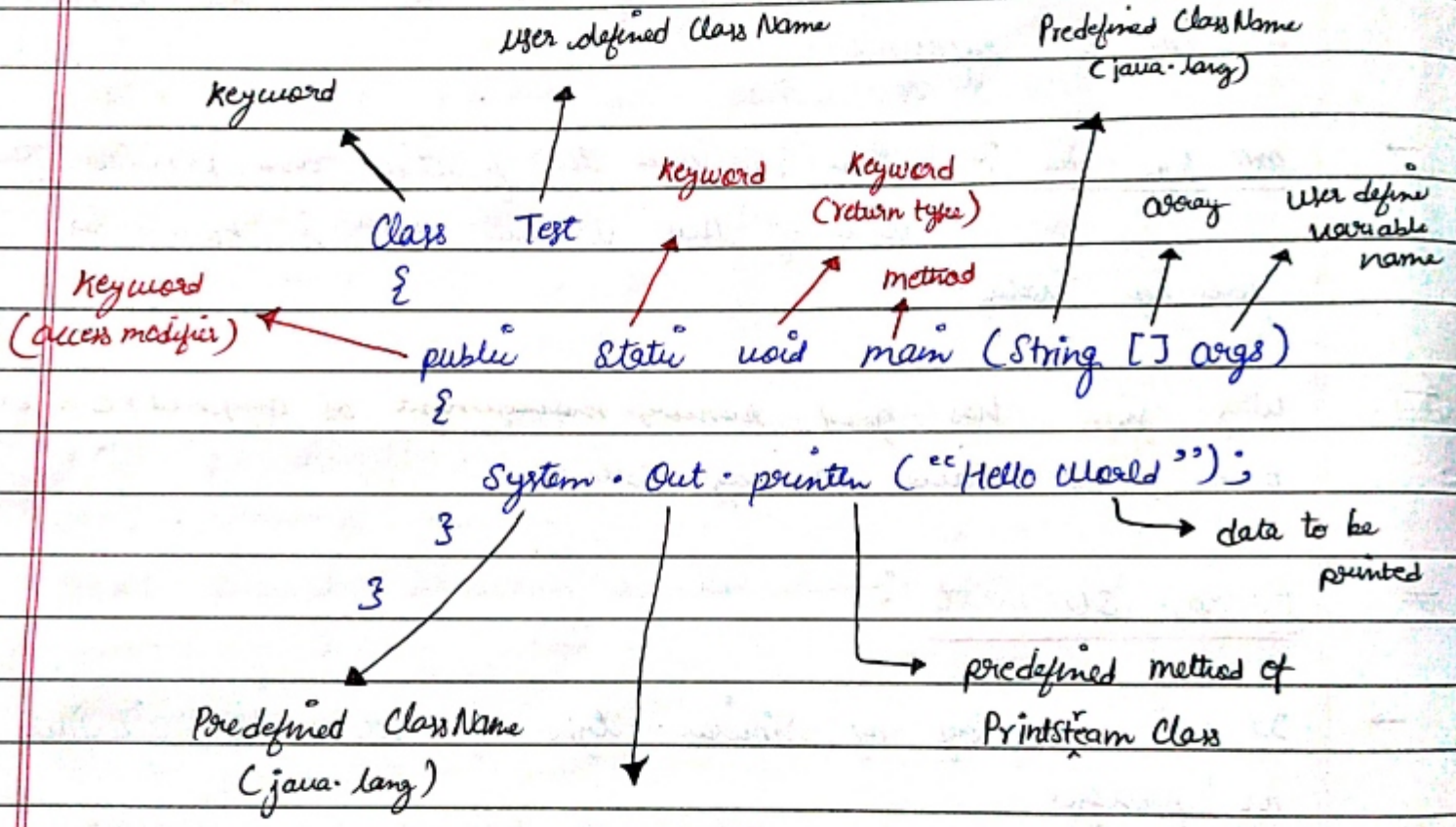
→ User-defined Package - created by the user

Date ___ / ___ / _____

→ Package statement should be only one in the single java file.

→ Package statement should be the first statement in java file.

→ Java.lang package is the only package which is imported by default




```
Class PrintStream
{
    void println()
    {
    }
}
```

```
Class System
{
    static PrintStream out;
}
```

main() method Syntaxes :-

→ public static void main (String [] args) - correct

→ public static void main (Integer [] args) - correct syntax for method but incorrect syntax for main method.

→ static public void main (String [] args) - correct

→ public void static main (String [] args) - Incorrect & compile time error because method syntax is incorrect as java need always method name after return type.

→ public static void args (String [] args) or
public static void xyz (String [] args) - correct syntax for method but incorrect for main method.

→ public static void main (String args () { }) - Incorrect

→ public static void main (String args) - correct syntax for method but incorrect for main method.

→ public static void main (String [] xyz) - correct

→ public static void main (String ... xyz) - correct but there should be 3 dots (Variable arguments)

→ protected / private static void main (String [] args) - Correct syntax for method but incorrect for main method

→ final / synchronized / strictfp public static void main (String [] args) - correct

Date ___ / ___ / _____

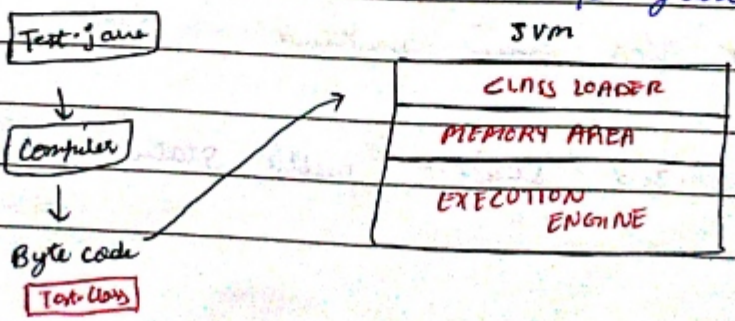
→ public void main (String [] args) – Correct Syntax for method but incorrect for main method.
Error: Main method is not static in class

→ Static void main (String [] args) – Correct Syntax for method but incorrect for main method.

Ques. Why we use Static keyword in main method ()?
Java is a Object oriented programming language, So, to call any method JVM requires objects and main method is the first method to be executed even before any object is being created So, all this happened because of Static keyword.
So, because of Static keyword JVM directly call the main method.

Ques. What is the role of compiler?
→ Compiler check the Syntax whether it is correct or not (Java is case-sensitive language thus it checks upper case and lower case keywords name).
→ Compiler ignore all the comments
→ Compiler generate the byte code

Ques. What is happens in execution phase?
→ Byte code is loaded in JVM.
→ Byte code verifier checks whether the byte code is correct or not.
→ memory allocation start
→ Interpreter and Compiler excutes our program.



Date ___ / ___ / _____

Ques. What is Virtual machine ?

→ It is a software simulation (copy) of a machine which performs operations similar to physical machine.
Ex: Calculator, JVM.

→ Types of Virtual machine :-

- Hardware Based VM or System Based VM
- Application Based VM or Process Based VM

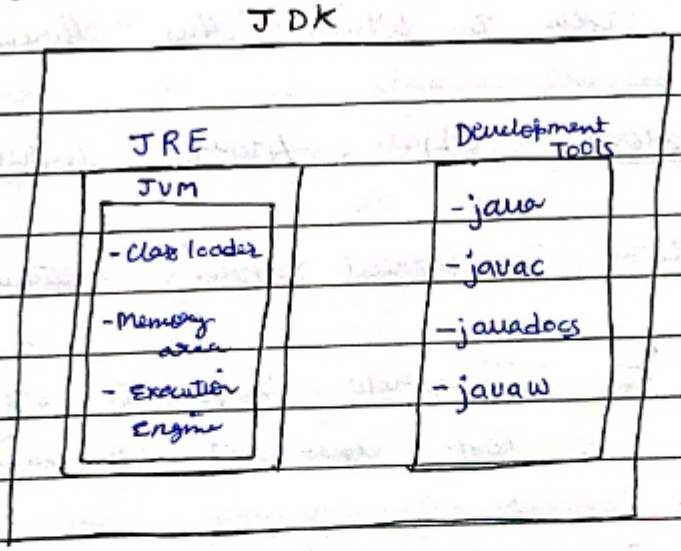
→ JVM is application based VM.

Ques. What is JVM ?

→ JVM is a Java Virtual Machine which is used to execute java byte code.

→ JVM is application based V.M

→ Architecture of JVM.



→ JDK contains tools need to develop java program and JRE to run the program

→ JDK = JRE + Development Tools

Date ___/___/___

loading, linking, initialisation

Class Loader → Class loader loads .class file into the method area
→ After that byte code verifier verify the syntax of the byte code and also check whether the byte is compiles through original java compiler or not. code file

Memory Areas : Total 5 types memory area —

1. Method Area
2. Heap Area
3. Stack Area
4. Pc Register
5. Native Method Area

→ In the method area .class file information and static variables are stored.

→ Method area is created when JVM is started.

→ Only one method area is created for one JVM and due to this multiple-threads altogether execute the method area due to which this thread is not safe.

* Heap area stored - Objects, Arrays, instance variable

* Stack area stored - Current running method, local variables

Execution Engine :- Interpreter helps to convert the byte code into the machine code.

→ JIT compiler :- Just In Time compiler, it is basically used to speed up the compilation of the code.

It basically look up to those functions/methods who are appearing frequently and store there threshold value and after that if exceed the threshold value then the JIT compiler directly executes those methods.

Data Types :- The type of data that we are specifying to java is known as Data type.

According to data types, languages are divided into 2 categories-

① Statically Type language : In this type of language we have to specify the type of each data.
Ex: C, C++, Java, Fortrain, Pascal

② Dynamically Typed language : In this type of language we donot have specify the type of data that we have provided.
Ex: Python, Javascript, Ruby.

Types of Data Types :-

① Primitive Data Types (Predefined Data Types) :-
The data types which are already provided by Java and whose size are fixed are known as primitive data type.
Ex:- There are 8 primitive data types -
boolean, char, byte, short, int, long, float, double

② Non-primitive Data Types (User Defined Data Types or Derived Data Type) :-
→ These data types are not pre-defined data types but are created by the programmer. These are sometimes known as "reference variable" or "object reference".
→ The size of non-primitive data types are not fixed.
Ex:- String, Array, collections, class, Abstract class, Interface etc.

Size — not fixed
Default value — null
Wrapper class — does not exist

Date ___ / ___ / _____

	boolean	Char	byte	Short	int	long	float	double
Size :-	not define	2 bytes	8 bits	2 bytes	4 bytes	8 bytes	4 bytes	8 bytes
default Value :-	false	'\u0000'	0	0	0	0.	0.0	0.0
Wrapper :- class	Boolean	Character	Byte	Short	Integer	long	Float	Double

Wrapper Class :-

The classes which are used to convert primitive into objects and objects into primitive.

Auto Boxing :-

Auto Boxing is the automatic conversion of primitive data types into its corresponding wrapper classes by Java Compiler.

Unboxing :-

Unboxing is the automatic conversion of an object of wrapper type to its corresponding primitive value.

Ques: What is ADT (Abstract Data Types) ?

→ ADT is a type (or class) which holds the different types of objects with some specifications

→ The defination of ADT only mentions what operations are to be performed but not how these operations implemented.
Ex:- Stack ADT, List ADT, Queue ADT

Ques: Why Java is not purely OOP's language ?

Java is not purely OOP's language because -

→ Usage of Primitive Data Types

→ Usage of Static members

Ques How primitive variable passed to methods - by value or by reference ?

Java supports only pass by value.

Ex:-

```
Class Test
{   public static void main(String[] args) {
        Test t = new Test();
        t.sum(10,20);        → Pass by value
    }
    void sum(int a, int b) {
    }
}
```

Type casting : The process of converting the data from one data type to another data type is known as type casting.

There are 2 types of Type-casting in Java:-

① "Primitive Data Type" Type casting

- Widening Type casting (Implicit Type casting)
- Narrowing Type casting (Explicit Type casting)

② "User Defined Data Type" Type casting

Widening Type casting : It is the process of converting data from lower data type to higher data type

Narrowing Type Casting : In this process of converting data type from higher to lower data type.

→ Narrowing type casting achieved through "cast operator".

Date ___ / ___ / _____

NOTE :- no1 & no2 , result

no1 + no2 = result

→ If no1 & no2 is byte, short or int then result will be always in int.

→ If no1 & no2 is long, float, dobble etc then result will be always in higher data type.

Ex:- byte no1 = 10; byte no2 = 20;

byte res = (byte)no1+ (byte)no2; //Error loosy conversion

byte res = (byte) (no1+no2); // correct.

② "User Defined Data Type" Type Casting :-

→ It is the process of converting data from one user defined data type to another user defined data type.

→ For "user defined data type" type casting, both data types should have relation (either extends or implements)

Ex:- → Superior class (Father/Parent) of all class

```
Class Object {
}

Class String extends Object
{
}

Class Test {
    String name = "deepak";
    Object O = (Object) name;
}
```

Variables :- Variable is the name of memory location that contains the data.

The variable value can be changed acc. to programming logic.

Types of Variables —:

1. Local Variables
2. Instance Variables
3. Static Variables

(1) Local Variables :- Local variables are declared within the body of methods, constructor or blocks.

→ Scope :- Local variable can be used within the methods or constructor or blocks but not outside them.

→ When local variable gets memory allocated : Local variables gets allocated when the methods or constructors or blocks are executed and get deleted from memory when that method or block or constructor execution completes.

→ Stored Memory area :- Local variables get memory allocated in "stack Area".

→ Default Values :- Local variables does not have any default value, if we dont provide the value, and use them it will provide compile time error.

→ Access Modifier :- We cannot use access modifier i.e. public, protected and private with local variables.

→ How to access local variable :- Directly.

Saathi

Date ___ / ___ / _____

(2) Instance Variable :- Instance variable are declared within the class but outside the methods or constructors or blocks.

→ Scope :- Instance Variable can be used within the class and every method or block or constructor but not inside the static methods or static blocks.

→ When instance variable gets memory allocated :- Whenever new Object is created, instance variables get memory allocated and when that object is destroyed instance variable also get deleted.

→ Stored Memory Area :- Instance variable are stored in 'Heap area'.

→ Default Values :- Instance variable have default values.
Ex: int - 0 ; boolean - false ; float - 0.0 ; etc.

→ Access Modifier :- We can use access modifiers.

→ How to access instance variable :- directly, by using object name

(8) Static Variables :- Static variables are also declared within the class but outside the methods, or constructor or block and we also use "static" keyword with them.

→ Scope : static variables can be used in every method or static method or blocks or static blocks or constructor

→ When static variables get memory allocated :- When we run java program, .class file or byte code get loaded in JVM and at that time only static variable also get memory allocated. When .class get unloaded from JVM the variables got deleted.

Page No. ____

Date __/__/____ Aman Kumar Yadav Saathi

→ **Stored Memory Area :-** Stored in 'method Area'.

→ **Default values :-** have default values. Ex: int - 0; boolean - false; float - 0.0 etc.

→ **Access Modifier :-** It can use access modifiers

→ **How to access Static variable :-** directly, by Object name, by Class Name

Ex:-

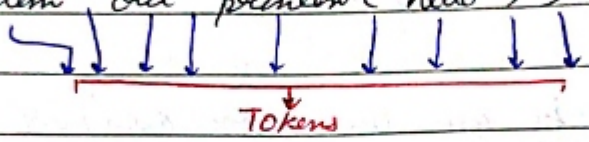
```
Class Variable {
    int d;                          // instance variable
    int f = 700;
    public Static int e = 500;      // static variable
    void sum () {
        int a = 100;
    }
    void mull () {
        int b = 400;
        // int c = a+b;             // Error we cannot use local variable inside mull()
        int c = d+b+c;
    }
    Static void divide () {
        int groceno = 100;
        // int res = d;             // Error because d is not static variable
    }
    public static void main (String [] args) {
        Variable ob = new Variable ();
        ob.sum ();
        S.O.P (e);
        S.O.P (ob.e);
        S.O.P (variable.e);
```

Page No.

Date ___ / ___ / _____

Tokens :- Token are the smallest unit or say small building blocks of java program that are meaningful to the java compiler.

Eg:- `System . out . println ("hello") ;`



→ Our java program converts into tokens and then java compiler converts these tokens into java bytecode.

Different types of tokens :-

① Literals
② Operators
③ Separators
④ Punctuators
⑤ Comments
⑥ Keywords / Reserved words
⑦ Identifiers

1/ **Literal** :- Any content value assigned to the variable is known as literal. For Eg:- `int a = 10;` 10 is literal
`Char c = 'x';` x is literal

2/ **Operators** :- Operators are the special symbols which are used to perform any operations on one or more operands.

Eg: `c = a+b` (+ is a operator and a, b are operands)

Date ___/___/_____ Aman Kumar Yadav Saathi

Types of Operators :-

① Arithmetic Operators → +, -, *, /, %

② Unary Operators → postfix : no++, no--
prefix : ++no, no--, +no, -no, ~no, !no

③ Assignment Operators → =, +=, -=, *=, /=, &=, ^=, |=, >>=

④ Relational Operator → ==, !=, <, >, <=, >=, instanceof (to check whether the object you created belong to that class.)
(Provide the output in true or false)

⑤ Bitwise Operators → ① Bitwise logical Operator
& (bitwise AND), | (bitwise OR), ^ (bitwise exclusive OR)

② Shift Operator - >> (right shift) (remove 2's zeros), << (left shift) (add 2's zero), >>> (zero fill right shift)

⑥ Logical Operators → && (logical AND), || (logical OR)

⑦ Ternary Operators → ? :

Que. What is difference b/w & and && operator ?

→ & - Bitwise Operator
&& - Logical AND Operator

→ & - Operates on bit values
&& - Operates on boolean values

Date ___ / ___ / _____

3. **Separator** : A separator is a symbol that is used to separate a group of code from one another.

Types of Separators :-

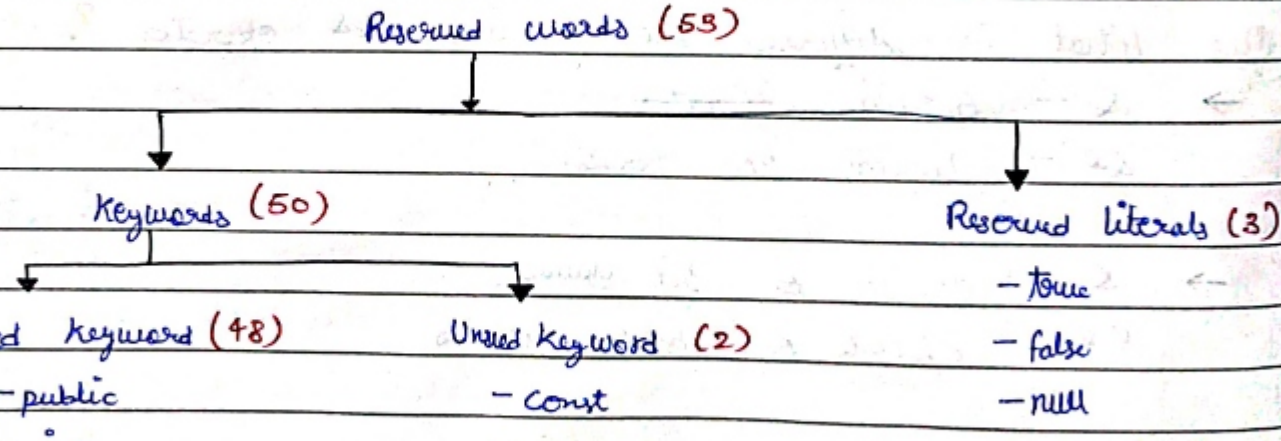
- → ( ) – Paranthesis
- → { } – Braces
- → [ ] – Big Brackets
- → ; – Semi colon
- → , – Comma
- → . – Period

4. **Punctuators** : Punctuators are the symbols or tokens that has semantic meaning to the compiler.

Types of punctuators :-

- → ? – Question Mark
- → : – Colon
- → :: – Double colon (used to create method or Constructor references)

5.



Keywords :- Keywords are predefined words having any specific meaning.

Date __/__/____

Programming Paradigm :-

→ Programming Pradigm is a way or an approach to solve any problem or to achieve any task using any programming language.

There are 2 classification of programming paradigm :-

① Imperative Programming Paradigm
② Declarative Programming Paradigm

Imperative	Declarative
→ We have to specify step by step every task	We have to define the problem to achieve the task.
→ User makes the decision and command to the compiler	Allows compiler to make decision.

→ Real World Eg: By a PC, Proposal to a girl, read mails.

OOPs :-

→ In OOP program is divided into parts i.e. objects
→ In OOP bottom up approach is used
→ have dot of access modifiers
→ OOPs deals with data
→ It needs more memory as compared to Procedure Oriented Progm.
Ex:- Java, C++, C#

Note :- One language can use multiple programming paradigm

Ques. What is diff. b/w OOP language and Object Based Programing language?

In OOP language inheritance feature is mostly used but in case of OBPL inheritance is not used.
OOP – JAVA ; OBPL – JAVASCRIPT

Saathi

Date ___ / ___ / _____

OOP's :-

→ Full form is Object Oriented Programming.

→ OOP is the programming paradigm based on the concept of objects which contains the data and methods.

→ It is the most popular programming paradigm used by the programmers.

→ Examples:- Java, Python, C++, etc.

→ Features of OOPs :-

(i) Class, Objects & Methods
(ii) Message Passing
(iii) Inheritance & Composition
(iv) Polymorphism
(v) Encapsulation
(vi) Abstraction

Class :- A class is a user-defined blueprint or prototype which is used to create an object.

→ Simply we can say that a class is a group of objects having common properties, behaviour, relationships.

Eg:-

```
Class Animal
{
    int age = 10;
    String color = "Black";
}
```

Methods :- A set of codes which performs a particular task.

Advantages :- Code Reusability → code Optimization

Eg:- void cat () { → Method declaration
S.O.P("I am caling"); } → Method definition
}

Objects :- Object is an instance of class

→ Object is physical entity or object is real world entity.
→ An object have 3 characteristics :-
(i) State (represents the data (value) of an object)
(ii) Behaviour (represents the functionality of an object)
(iii) Identity (represents the unique id of an object which is created automatically by JVM).
→ Object is simply a memory block.

Eg:-
```
className Object_Name = new ClassName ();
```
→ Calling a variable or methods from object
```
Object_Name . variable_name ;
Object_Name . methodName ();
```

Constructors :- Constructors are the special methods having same name as that of class name and does not have any return type.

Eg:-
```
Class Animal {
    Animal ()
    {
    }
}
```
→ Constructor can be public, private protected or default.
→ We cannot use abstract, final, static, synchronized etc keywords with constructors.

Date ___ / ___ / _____

→ Use of constructor :- Constructor are used to initialize an object but not for object creation.

→ When constructor are excelled :- Constructor are executed exactly at the time of object creation, not before or after object creation.

→ How constructor are excuted :- Constructor are executed automatically when we create an object.

→ Syntax : access-modifiers ClassName (list of Parameter) throws Exception

```
{
    // initiazation code
}
```

→ Types of constructor :- There are 3 types of constructor :-

① Default Constructor (compiler)
② 0- Argument Constructor (programmer)
③ Parametrized Constructor (programmer)

① Default Constructor :- Whenever we donot create any constructor in class, then compiler will always create a constructor which is known as default constructor

Note :- If programmer creates any one constructor then compiler will not generate default constructor.

→ Access modifier of default constructor will be same at that of class access-modifier.

→ Access-modifier of default constructor cannot be private or protected because outer class cannot be private or protected.

```
Class Test {
    Test () {
      Super ();
    }
}
```

Parametrized Constructor :- These constructors are created by the programmers.

```
Class Test {
    Test (int a, int b) {
    }
}
Class Animals {
  int age;
  String color;
  Animals ( int age1, String color1) {
      age = age1;
      color = color1;
    }
Void eat () {
   S.O.P ("I am eating");
  }
Void run () {
   S.O.P ("I am running");
  }
public static void main (String[] args) {
    Animals buzo = new Animals (10, "Brown");
     S.O.P ("Age" + buzo.Age);
     S.O.P ("Color" + buzo.color);
     buzo.eat ();
     buzo.run ();
   }
}
```

Saathi

Date ___ / ___ / _____

Ques What is difference b/w Methods & Constructors :-

→ Methods always have return type.
Constructors does not have any return type even void.

→ Methods can have any valid name.
Constructors always have same name as that of class name.

→ Methods are used to perform any particular task.
Constructors are always used to initialize an object.

→ We have to call the methods explicitly by using object name or class Name
Constructors are called automatically when we create an object.

→ If we donot create any method then compiler will not generate any method.
If we donot create any constructor then compiler will generate default constructor.

NOTE:-

→ Constructor is profined class present in java.lang.reflect package.

→ This constructor class is used to get constructor related information.

```
Class Student {
    int rollno;
    String name;

    Student (int rollno1, String name1) {
        rollno = rollno1;
        name = name1;
    }
    void show () {
        S.O.P("Rollno:" + rollno + "Name:" + name);
    }
}
Class StudentMain {
    P.S.V.m (String[] args) {

        Student S1 = new Student (01, "Aman");
        Student S2 = new Student (02, "Amar");

        S1.show();
        S2.show();
    }
}
```

Object creation :-

① Hashcode :- As soon as heap manager creates an object, a unique integer value will be assigned to the object which is known as "hashcode".

② Now JVM convert this hashcode (1235) into hexadecimal code (ab123) which is also known as reference value.

③ Now this "reference value" is assigned to the object and this object is known as "reference variable".

Date ___ / ___ / _____

Relationship Between Classes :-

→ Use of relationship between java classes :-

(i) Code reusability
(ii) less Execution Time
(iii) less Memory Time

→ Types of relationship :-

① IS-A Relationship (Inheritance)
② HAS-A Relationship (Association)
③ USES-A Relationship (Dependency)

① IS-A Relationship :- It is one in which data members of one class is obtained into another class through the concept of inheritance.

```
Class A {
-------
}
```

ISA Relationship

```
Class B extends A
{
}
```

② HAS-A Relationship :- It is one in which an object of one class is created as a data member into another class.

```
Class A {
-----
}
```

HAS-A Relationship

```
Class B {
A Obj = new A();
---
}
```

(3) USES-A Relationship :- It is one in which a method of one class is using an object of another class.

```
class A {
------
}
Class B {
void show() {
A obj = new A();
}
}
```

USES-A Relationship

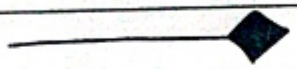
UML Notation in JAVA :-


Inheritance

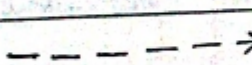

Interface Inheritance


Association


Aggregation


Composition


Direct Association


Dependency

Date ___ / ___ / ___

IS-A Relationship :- (Inheritance)

→ It is also known as "Inheritance".

→ IS-A Relationship or Inheritance is achieved by using "extends" Keyword.

→ All java classes except Object class will always have one parent class thus we can say that the total java API is implemented based on inheritance concept.

Eg:-

```
Class Animal {
 void eat() {
     S.O.P ("i am eating");
    }
}
Class Humans extends Animal {
 }
Class InheritanceMain {
   P.S.V.m ( String [] args ) {
    Humans deepak = new Humans ()
      deepak.eat();
    }
}
```

→ Use for code reusability.

→ For method overriding to achieve runtime polymorphism.

Types of Inheritance :-

① Single Inheritance
② Multiple Inheritance
③ Hierarchical Inheritance
④ Multilevel Inheritance
⑤ Hybrid Inheritance

Date ___/___/_____

Notes :-

→ By default if any java class does not inherit any parent class then it inherits Object class.

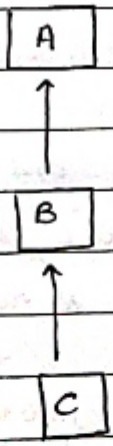
→ There can be only one parent class for every class and due to this java does not support multiple inheritance.

→ Private members of parent class is not inherited in child class and constructor are not inherited because constructor are not the part of class members (fields, methods, nested class).

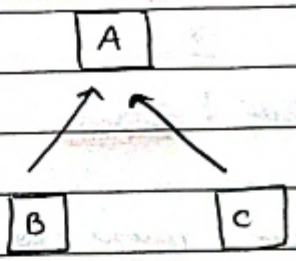
→ Cyclic inheritance is not possible.

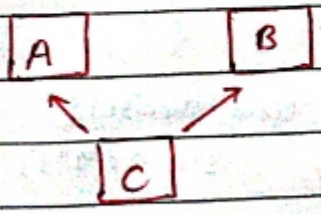
Single Inheritance

Multilevel

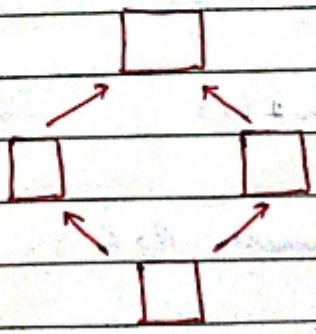


Hierarchical Inheritance





Multiple Inheritance



Hybrid Inheritance

Date ___ / ___ / ____

Multiple Inheritance

```
Class A {
    void show1() {
        S.O.P("1");
    }
}

Class B {
    void show1() {
        S.O.P("2");
    }
}

Class C extends A,B {
    }

Class Main {
    P.S.V.M (String[] args) {
    C obj = new C();
    obj.show1();
    }
```

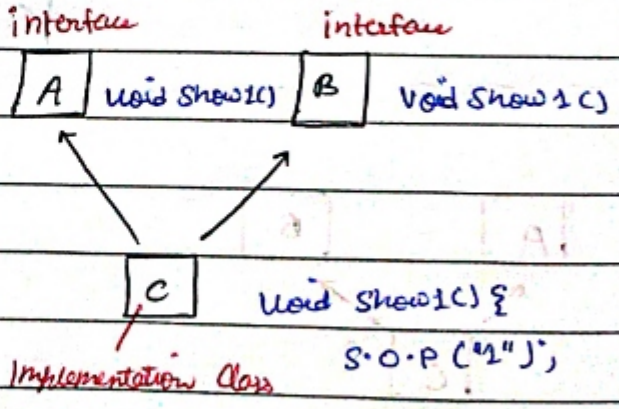
Output : Error

→ If we want to achieve Multiple inheritance then we can achieve by interfaces but it is not strictly inherited and Hybrid

```
interface A {
    void show1();
}

interface B {
    void show1();
}

Class C implements A,B
{
    void show1() {
    }
```



HAS-A Relationship (Association) :-

→ Association is relation between two separate classes which establishes through their objects.

→ Association has 4 types :-

① One to one
② One to many
③ many to one
④ many to many

→ Association has 2 forms :-

① Composition (STRONG BONDING)
② Aggregation (WEAK BONDING)

⇒ One-to-one :-

a) One person has one passport
b) One employee have one employee id
c) One student has one roll no.

⇒ One-to-many :-

a) One person have multiple phone numbers
b) One student can have multiple courses
c) One person can have multiple bank accounts.

→ many-to-one :-

a) Multiple cities exist in one state
b) Many ports can exist in one extention

→ Many-to-many :-

a) Multiple customers can buy multiple products and multiple products can be bought by multiple customers.

It is relationship b/w entities where one instance of an entity should be mapped only to one instance of another entity.

Saathi

Date ___ / ___ / _____

One - to - One Example :-

```
Class Person {
String name ;
String city ;
Passport pwd ;

    Person ( String name1, String city1, Passport pwd1 ) {
      name = name1 ;
      city = city1 ;
      pwd = pwd1 ;
      }
void showDetails () {
      S.O.P("Name" + name) ;
      S.O.P("city" + city) ;
      S.O.P("Passport id" + pwd.passId ) ;
      S.OP ("Passport valid" + pwd.validity) ;
      }
}
Class Passport {
   String passId ;
   String validity ;
      Passport ( String passId1, String validity1) {
            passId = passId1 ;
            Validity = validity1 ;
            }
      }
Class Main {
      P.S.V.M (String [] args) {
      Passport pass = new Passport ("321", "01-07-2027") ;
      Person p1 = new Person ("Aman", "Punjab", pass) ;
      p1.showDetails() ;
      }
}
```

If is the relation b/w entity classes where one instance of an entity should be mapped with multiple instance of another entity.

One to Many Example :-

```
Class Student {
String id ;
String name ;
Courses [] courses;
Student ( String id1 , String name1 , Courses [] Courses1 ) {
  id = id1 ;
  name = name1 ;
  courses = courses1 ;
  }
  void ShowDetails () {
      S.O.P ( " Student ID" + id );
      S.O.P( " Name " + name);
      S.O.P( " Courses Detail" );
      for ( int i=0 ; i< courses.length; i++) {
          S.O.P ("Courses" + (i+1)+ ": " + Courses [i]. coursename);
      }
  }
Class Courses {
  String branch ;
  String coursename ;
  Courses (String branch1 , String coursename1 ) {
      branch = branch1 ;
      coursename = coursename1 ;
      }
  Class OneToManyDemo {
      P.S.V.M( String [] args ) {
      Courses C1 = new Courses ("CSE", "Java");
      Courses C2 = new Courses ("CSE", "C++");
      Course C3 = new Courses ("CSE", "Python");
      Courses [] Courses1 = { C1, C2, C3 } ;
      Student S1 = new Student ("101", "deepak", Courses1);
      S1. ShowDetails ();
  }
}
```

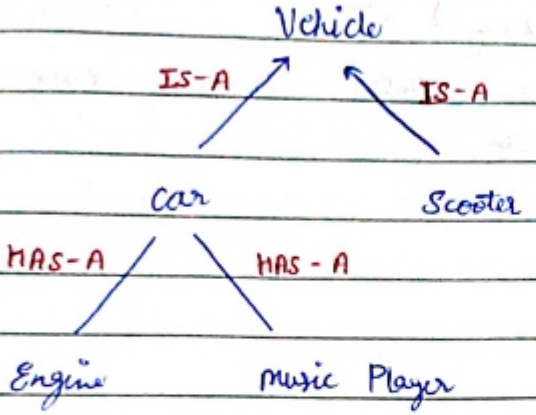
Dependency Injection

MANY-TO-ONE :
Date __/__/____

```
Class Student {
String rollno;
String name;
Branch branch;
    Student (String rollno1, String name1, String branch1) {
        rollno = rollno1;
        name = name1;
        branch = branch1;
    }
    Void ShowDetails () {
        S.O.P("Student Rollno:" + rollno);
        S.O.P("Student name:" + name1);
        S.O.P("Student branch Code:" + branch.branchCode);
        S.O.P("Student branch Name:" + branch.branchname);
    }
Class Branch {
String branchCode;
String branchname;
  Branch (String branchCode1, String branchname1) {
        branchCode = branchCode1;
        branchname = branchname1;
    }
}
Class manyToOne {
    P.S.V.m (String[] args) {
    Branch b1 = new Branch ("CSE01", "CSE");
    Student S1 = new Student ("01", "Aman", b1);
    Student S2 = new Student ("02", "Amit", b1);
    Student S3 = new Student ("03", "Amar", b1);
        S1.ShowDetails();
        S2.ShowDetails();
        S3.ShowDetails();
    }
}
```

dependency Injection

Date ___ / ___ / _____



Car HAS-A Engine (STRONG - COMPOSITION)
Car HAS-A MusicPlayer (WEAK - AGGREGATION)

Composition : Part of
Aggregation : has

Q. Where we have to use IS-A Relationship and HAS-A relationship ?
If we want to inherit all the properties of parent class into child class then we can use IS-A relationship but if we want to only partial inheritance then we have to use HAS-A relation.

Dependency Injection :-

→ DI is the concept in which object get the other required objects from outside entity.

→ The process of injecting or inserting dependent (contained) object into the container object is known as Dependency Injection.

→ Types of Dependency Injection —:

1. Constructor dependency injection
2. Setter method dependency injection

USES-A Relationship (Dependence) :-

It is one in which a method of one class is using an object of another class.

```
Class Account {
    String accno ;
    String name ;
    int balance ;
  Account (String accno1, String name1, int balance1) {
        accno = accno1 ;
        name = name1 ;
        balance = balance1 ;
    }
Class Transaction {
    void deposit ( Account account, int amount ) {
        account.balance = account.balance + amount ;
      S.O.P ("Name" + account.name ) ;
      S.O.P ("ACCNO" + account.accno) ;
      S.O.P( "T.Balance" + account.balance) ;
    }
}

Class BankApp {
        P.S.V.M ( String[] args) {
      Account ob1 = new Account ("1234", "Aman", 10,000) ;
      Transaction ob2 = new Transaction ()
        ob2.deposit ( ob1, 5000) ;
  }
}
```

Polymorphism :- Concept by which we can perform a single action in different ways.

→ Poly (many) + Morphism (forms, structure)

→ Real world Example :- Sound, water, many brands in one shop.

→ Advantage :- It provides the flexibility to develop an application i.e. it allows us to perform a single task in different ways.

→ Types of Polymorphism :

1. Compile Time Polymorphism.
2. Runtime Polymorphism

* Compile Time Polymorphism :-

→ It is also known as Static Polymorphism or Early binding.

→ If the polymorphism is achieved at compile time then it is compile time polymorphism.

→ Compile time Polymorphism can be achieved by 2 - ways :-

① Method Overloading

② Operator Overloading (is not supported in java except + Symbol)

* Runtime Polymorphism :-

→ It is also known as Dynamic Polymorphism or Late Binding

→ If the polymorphism is existed at runtime then it is known as Runtime Polymorphism.

→ Runtime Polymorphism can be achieved by "Method Overriding".

* Method Overloading :-

The process of compiler trying to resolve the method call based on reference type is known as method Overloading.

→ Rules for method Overloading :

1) Same name
2) Within the same class
3) Different parameters
   - → No. of Parameters
   - → Type of Parameters
   - → Sequence of Parameters

```
Class Test {
    void sum(int a, int b) {
        int res = a+b;
        S.O.P(res);
    }
    void sum(float a, float b) {
        float res = a+b;
        S.O.P(res);
    }
}

Class MethodOverloading {
    P.S.V.M(String[] args) {
        Test t = new Test();
        t.sum(10,20);
        t.sum(10.4f, 20.4f);
    }
}
```

Q. can we overload main method ?
Yes, we can.

Q. can we overload constructor ?
Yes, we can.

* Method Overriding :-

The process of JVM trying to resolve the method call based on reference type is known as method overriding.

→ Overriding is the feature by which child class trying to change the implementation of parent class method.

→ Rules for method overriding :-

1) Same name
2) Within different class
3) Same parameters
4) IS-A Relationship

```
Class A
{
   Void Show()
    {
      S.O.P("1");
    }
}
Class B extends A
 {
   Void Show()
    {
      S.O.P("2");
    }
 }
Class Main {
   P.S.V.M(String[]args)
    {
       A Ob1 = new B();
         Ob1.Show();
    }
}
```

Method overriding

```
Class A
{
   Void Show1()
    {
      S.O.P("1");
    }

Class B extends A
 {
    Void Show2()
     {
       S.O.P("2");
     }
 }
Class Main {
   P.S.V.M(String[] args)
    {
    A Ob1 = new B();
     Ob1.Show1();
     Ob1.Show2(); //Error Cannot find Symbol .Show2

    B Ob1 = new A()
```

↳ Implicit down casting

```
    A Ob1 = new B()
    B Ob1 = (B) Ob1
```

Cases for method overriding :-

→ If we change the return type in method overriding then it will provide compile time error.

→ We can provide child class as a return type for overriding method and this concept is known as covarent return type.

→ Child class method should have equal or higher access modifiers as compared to parent method access modifier in method overriding.

→ We cannot override private methods.

→ And final method cannot override

→ And static also. And constructor also cannot override

Data Hiding :-

→ Data-Hiding is the process of hiding the data from outside users.

→ It is achieved by private access-modifiers.

Eg:-
```
class Account
{
  private int balance ;   // data hiding
}
```

→ To access or modify the private variables then java provides special methods i.e getter and setter methods.

→ It is highly recomended to declare variables as private.

Abstraction :-

→ Abstraction is hiding the details (hiding the implementation part) and just highlight the main services.

→ Real World Example :- Car (internal working of breaks, gears etc are hidden from user) etc

→ It is achieved by abstract class & interfaces.

Abstract Method :-

→ Abstract Methods are those which does not have body or implementation part.

→ Abstract Method are those which have only declaration part, not implementation.

For ex:- abstract void sum() ;

Abstract Class :-

→ Abstract class are those which can contain both concrete methods and abstract methods.

Ex:
```
abstract class Test {
    void show() {
    }
    abstract void Print();
}
```

POINTS TO REMEMBER :-

1. If any class contains abstract methods then that class should be declared as abstract class.
2. If we declare any abstract class, then it can contain both concrete methods as well as abstract method.
3. We cannot create an object of abstract class but we can declare variable for abstract class.
4. If any class inherits abstract class then it should implement all the abstract methods or that class should also be declared as abstract.
5. Whenever we use abstraction concept we are using method overriding concept also.
6. Abstract class can have constructors.
7. Abstract class can inherit concrete class.

Ex:-

```
Class Vehicle
 {
    void start ()
     {
        S.O.P("start with key");
        S.O.P("start with kick");
     }
 }
Class Car extends Vehicle
 {
 }
Class Scooter extends Vehicle
 {
 }
Class AbstractDemo {
    P.S.V.M(String[] args) {
        Car ob = new Car();
        ob.start();
    }
}
```

→ Here, only one problem that whenever I want to make an object of scooter class and call the start method it print both the statement but I want only one with kick. So, here comes the role of "Abstraction".

```
abstract class Vehicle
{
   abstract void start();
}
Class car extends Vehicle
 {
    void start()
      {
        S.O.P("Starts with key");
      }
 }
Class Scooter extends Vehicle
 {
    void start()
      {
        S.O.P("Start with kick");
      }
 }
Class AbstractDemo
 {
    P.S.V.M(String[] args)
    {
       car ob = new car();
          ob.start();
       Scooter ob1 = new Scooter();
          Ob1.start();
       }
    }
```

Interface :-

→ Interface are similar to abstract class which can contain variables and methods but having all the variables as "public static final" and methods as "public abstract".

→ Interface is the blueprint of class which specifies what must do and not how.

Ex:

```
interface Vehicle
{
  void start();
  void changeGear();
}
```

→ Uses of Interfaces :-

→ It is used to achieve total abstraction.
→ It is used to achieve multiple inheritance.
→ It is used to achieve loose coupling.

Ex:-

```
interface Vehicle
 {
    void start();
    void noOfGears(int a);
 }
Class Bus implements Vehicle
 {
     public void start()
      {
        S.O.P(" bus starts with key");
      }
     Public void noOfGears (int gears)
      {
        S.O.P("Bus has" + gears + " gears");
      }

  Class Car implements Vehicle
   {
     public void start()
      {
        S.O.P(" Car start with key");
      }
      Public void no.of Gears (int gears)
       {
         S.O.P(" Car has" + gears +" gears);
       }

   Class Main {
        P.S.V.M (String[] args) {
        Bus b = new Bus();
         b.Start();
         b. noofGears(6);
        }
   }
```

Ques. What is difference b/w Concrete, Abstract Class and Interface Class?

→ Concrete Class : Class className { }
Abstract Class : abstract class ClassName { }
Interface : interface InterfaceName { }

→ Concrete Class : We can declare only concrete method
Abstract Class : We can declare concrete methods and abstract methods
Interface : We can declare only abstract method

→ Concrete Class : We can create objects
Abstract Class : We cannot create an object but we can declare variable name
Interface : We cannot create an object but we can declare variable name

→ Concrete Class : cannot achieve Abstraction
Abstract Class : can achieve partial abstraction
Interface : can achieve full abstraction

→ Concrete Class : Methods & variables are same as we declare
Abstract Class : Methods & variables are same as we declare
Interface : Methods → "public abstract" & variable → "public static final".

→ Interfaces cannot be private, or protected but nested interface can be anything.

* Interface New features :-

(i) We can create default methods in an interface which have implementation part.
(ii) We can create static methods in an interface.
(iii) We can create private methods or private static methods in interface.

marker Interface :- Any interface which does not contain any abstract method or any variable is known as marker interface.

interface it {
}
→ Used to provide extra features or information to the object at runtime.

Encapsulation :-

It is the process by which variables and methods are wrapped or binded into a single unit.

→ Technically hiding the data from other classes and these data can be accessed only through the member functions of its own class.

→ Real World Ex :- capsule (Medicine is encapsulated inside the capsule), Mobile, Java Class.

→ Encapsulation is achieved by declaring variables as private and public getter and setter methods.

Ex :-
```
Class Employee
{
    private int salary;
    public void setSalary(int salary)
    {
        this.salary = salary;
    }
    public int getSalary()
    {
        return salary;
    }
}
Class Main {
  P.S.V.M(String[] args) {
    Employee e1 = new Employee();
      e1.setSalary(10,000);
      S.O.P(e1.getSalary());
  }
}
```

Advantages :-

① Data Hiding
② Increase Flexibility
③ Reusability
④ Testing code is easy.

Java Bean Class :

Java Bean class is used to encapsulate the data into single object.

→ Rules for Java Bean :

(i) Must implement Serializable interface
(ii) Class must contain public non-argument constructor
(iii) Class must have all private variables
(iv) Class must contain public getter and setter methods

⇒ Tightly Encapsulated Class :-

→ A class is tightly encapsulated class if and only if it have all the variables as private.

Q. What is the difference b/w Abstraction & Encapsulation ?

→ Abstraction : It hides the implementation (details)
Encapsulation : It hides the data (information)

→ Abstraction : It is achieved by "abstract" class and "Interface".
Encapsulation : It is achieved by using "JavaBean" class.

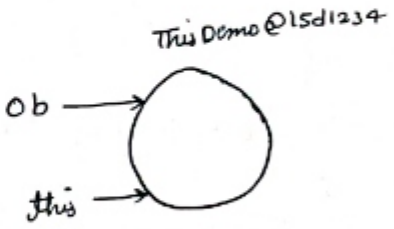
→ Abstraction : We have to use "abstract" keyword.
Encapsulation : We have to use access modifiers (private & public)

this Keyword :

→ this keyword is "reference variable" that refers to the current object.

→ Example of this keyword and reference variable are pointing to the same address of an object.

```
Class ThisDemo
{
    void m1()
    {
        S.O.P("I am in m1 method" + this );
    }
    P. S. V. M (String [] args)
    {
        ThisDemo ob = new ThisDemo ();
        S.O.P("I am in main method" + ob);
        ob.m1();
    }
}
```



→ Uses of this keyword :—

(i) this keyword is used to refer the current class instance variable. this.method_name

(ii) this keyword is used to invoke current class method.

(iii) this keyword is used to invoke current class constructor. this();

(iv) this keyword must be the first statement in the constructor call.

(v) this keyword can be used to pass as an argument in the method
↳ This case is mainly used in event handling.

(vi) this keyword can be used to pass as an argument in the constructor

(vii) this keyword can be used to return current class instance.

```
Class ThisDemo5
{
    ThisDemo5 ()
    {
        S.O.P("1");
    }
    ThisDemo5 (int no)
    {
        this ();
        S.O.P("2");
    }
    P.S.V.M (String [] args)
    {
        ThisDemo5 o62 = new ThisDemo5 (10);
    }
}
```

Super Keyword :

→ Super Keyword is a reference variable which is used to refer immediate parent class object.

→ Uses of super keyword :

(i) Super keyword can be used to refer the immediate parent class instance variable.

(ii) Super keyword can be used to invoke parent class method.

(iii) Super keyword is used to invoke parent class constructor.

(iv) We cannot use this() and super() together.

Q What is the difference b/w this and this() ?

```
Class A {
  int no=10;
}
Class B extends A
{
   int no = 20;
   void show (int no)
   {
       S.O.P(no);
       S.O.P(this.no);
       S.O.P(super.no);
   }
}
Class SuperTest {
   P.S.V.M (String[] args) {
     B ob = new B();
      ob.show(30);
   }
}
```

Output :
30
20
10

Example 2 :

```
Class A {
  A()
   {
     S.O.P("I am A Class constructor");
   }
Class B extends A
   {  super();
      S.O.P("I am B Class constructor");
   }
}
Class SuperTest {
   P.S.V.M (String [] args) {
      B Ob = new B();
   }
}
```

final :

→ final keyword is used to provide restrictions to the users.

→ final keyword can be used with :

① Variable (variable value cannot be changed, we cannot reassign final variable value).

② method (method cannot be overrided)

③ Class (class cannot be inherited)

Access - Modifier :-

→ Public

→ Protected

→ Private

→ default or no - modifier

Non- access - Modifier :-

→ Abstract

→ final

→ static

→ strictfp

→ synchronized

→ transient

→ volatile

Static Keyword :

→ Static keyword is non-access modifier

→ Static keyword can be used with

1. Variables
2. block
3. methods
4. nested class or inner class (not outer class).

→ Use of static keyword :-

1. It is used to improve share-ability
2. It is used for memory management

→ Static members belong to the class, not objects

Static variables :- If we declare any variable as static, it is known as static variables.

→ Static variables get memory allocated in method area at the time of class loading.

Example :-

```
Class Employee {
  int empid;
  String empname;
  String empcomname; // static String ecomname = "Smart Programming";
  Employee ( int empid, String empname, String empcomname)
  {
      this.empid = empid;
      this.empname = empname;
      this.empcomname = empcomname;
  }
  void display()
  {
      S.O.P ("Employee ID:" + empid);
      S.O.P ("Employee Name:" + empname);
      S.O.P("Employee Company Name:" + empcomname);
  }
}
```

```
Class StaticDemo {
    P.S.V.M ( String [] args )
    {
        Employee e1 = new Employee (101, "Deepak", "Smart Programming");
        Employee e2 = new Employee (102, "Deepak", "Smart Programming");
        e1.display();
        e2.display();
    }
}
```

⇒ for static ~~method~~ int :-

```
Class PageVisitors {
Static int count = 0;
    PageVisitors ()
    { count = count + 1;
    }
    void noofVisitors ()
    {
        S.O.P (count);
    }
Class StaticDemo2 {
    P.S.V.M ( String [] args )
    {
        PageVisitors V1 = new PageVisitors ();
            "        V2 =   "          "
            "        V3 =   "          "
        V1.noofPageVisitors ();
    }
}
```

NOTE :→ We cannot create static local variable because main use of static variable is improve Share-ability but local variable have limited Share-ability thus it violates the rule of static keyword.

→ If we declare any variable in static method, then it will be treated as local variable only.

→ We cannot use instance variable inside static method but we can use static variable inside instance method.

STATIC BLOCK : A block created using static keyword is known as static block.

→ Static block is executed at the time of class loading.

→ Uses of static block :-

(i) We can create static block to initialize static variables.

ERRORS AND EXCEPTION :

Q. What is the difference b/w Error & Exception ?

Error	Exception
→ Error is the problem which will not allow us to compile and execute the program.	→ Exception is the problem which is occurred at runtime and which we can handle programmatically.
→ Error is occurred due to lack of java concepts or lack of system resources.	→ Exception occurs due to our programs.
→ Types of Error :- 1) Compile Time Error 2) Runtime Error	→ Types of Exception :- 1) Predefined Exception 2) User-defined Exception

Compile Time Error :

→ Lexical Error : (If we have any mistake in java keywords. Ex: Stato, iiod)

→ Syntax Error : (If we have any mistake in java predefined Syntax for ex: variable declaration)

→ Semantic Error : (It is the meaning-less statement)

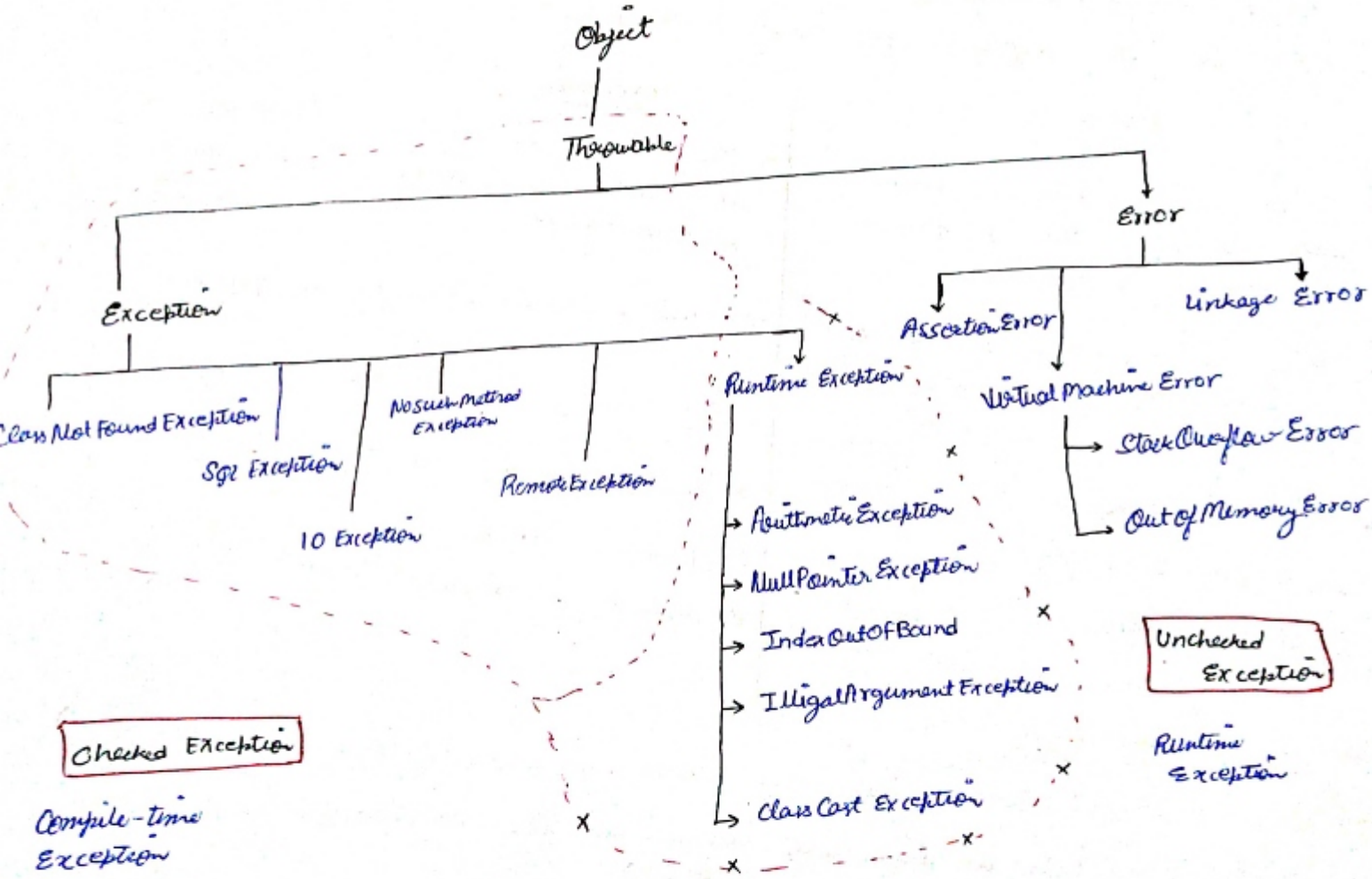
Runtime Error :-

→ Runtime errors cannot be handled programatically.
→ Runtime errors occurs because of JVM. (memory error)

Exception: Exception is any unwanted event which disturb the normal flow of the program.

Types of Exception :

→ Predefined Exception
→ User-Defined Exception
(Above both can be of 2 types — Checked Exception & Unchecked Exception)



Checked and UnChecked Exception :-

Ques: What is Checked and Unchecked Exception ?

→ Checked exception is the exception which can be checked by the compiler.
Unchecked exception is the exception which compiler cannot check.

→ If we donot support checked exception, our program will not compile.
In case of Unchecked exception, if we donot support the exception then our program can compile

→ In case of checked exception we have to use throws or try-catch keyword.
In case of unchecked exception its not compulsory to use throws or try-catch keyword.

Ex: of Checked Ex: :-

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
Class Test {
  P.S.V.M (String [] args) {
    try {
        FileInputStream fis = new FileInputStream("e://aaa.txt");
        int i = fis.read();
        S.O.P(i);
      }
    Catch (FileNotFoundException e)
      {
        S.O.P (e);
      }
  }
}
```

Ex: of Unchecked Except: :-

```
Class Test2 {
  P.S.V.M(String[] args)
  {
    int a=100; int b = 0, c;
    c= a/b;
    S.O.P(c);
  }
}
```

NOTE: Exceptions are always occured at runtime, no exception is occured at compile time.

Working of Exception :-

→ If we dennot handle the exception, our program will terminate abnormally.
→ If any exception is occured in any method, then that method will create an exception object and that method has the responsibility to handle the exception.
→ If the method is not handling the exception, then caller method has the responsibility to handle the exception.
→ If any caller method is not handling the exception object then that object will be transferred to JVM and then Default Exception Handler.
→ Then Default Exception Handler will print the exception object and JVM will terminate the exception object abnormally.

Throw Keyword: throw keyword is used to create an exception object programatically.

Syntax: throw new ExceptionClassName();

Ques. How to handle the exceptions ?
→ We can handle the exceptions by 2 ways -
1. by using "throws" keyword
2. by using "try-catch" block

Throws Keyword :
→ throws keyword is used to inform the caller method that this method can throw one of the listed type of exceptions.
→ throws keyword bypass the generated exceptions from present method to caller method.
→ If caller method wants to handle the exception then it has to use try-catch block.

NOTE: throws keyword cannot handle the exceptions.

Syntax: throws ExceptionClassName, ExceptionClassName -,-,-,
(It is used with method signature)

Ques. What is difference b/w throw & throws keyword ?

→ throw keyword is used to create an exception object manually.
throws keyword is used to inform that this method can raise listed type of exceptions.

→ throw keyword is used in method body.
throws keyword is used with method signature

→ throw keyword is able to allow only one exception class name at one time.
throws keyword is able to allow more than one exception class name at one time.

→ throw keyword is mainly used for unchecked exceptions
throws keyword is mainly used for checked exceptions

try-catch block :

try block : try block is used to provide the risky code that is the code in which there are chances for exception.

catch block : catch block is used to provide maintenance code or alternative code to handle the exceptions.

Java 7 features for try-catch block :

1. Multi-catch block
2. try with resources

finally block :

→ finally is the block which is always executes after try and catch block
→ finally block contains resources closing code for example database connection, reading or writing from file etc.

finally Example :-

```
Public class Test {
  P.s.v.m ( String [] args ) {
    try
    {
      S.o.P(100/0);
    }
    Catch ( Exception e)
    {
      S.o.P(e);
    }
    finally
    {
      S.o.P("Hii aman");
    }
  }
}
```

Multi catch Block Example :-

```
Class MultiCatchBlock
{
  P.s.v.m ( String [] args)
  {
    try
    {
      S.o.P(100/0);
      int [] arr = {10, 20, 30, 40};
      S.o.P(arr[5]);
    }
    Catch (ArithmeticException | ArrayIndexOutOfBoundException e)
    {
      S.o.P(e);
    }
  }
}
```

try with resource :-


```
Class Test3
{
   p.s.v.m ( String [] args)
   {
      try ( FileInputStream fis = new FileInputStream ("");)
       {
          fis.read();
       }
       catch ( exception e ) {
          s.o.p(e);
       }
   } }
```

⇒ try with resource is used to automatically close the resources that the programmer has created.

⇒ try with resource can contain only that class which inherits AutoCloseable interface.

⇒ Due to "try-with resource" the importance of "finally" block has decreased.

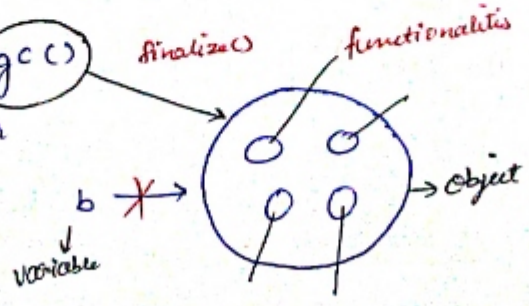
Ques. What is the difference b/w final, finally and finalize ?

→ final is the keyword
finally is the block
finalize is the method

→ 'final' keyword can be used with variables (the value cannot be changed or re-assigned of the final variable), method (we cannot override the final method) and class (we cannot inherit the final class).

'finally' block is used to close the resources.

'finalize' method is executed just prior to the garbage collector (or gc() method) It is used to perform the clean-up activity related to the object.



* how does gc() know to delete which object?

→ If none of the variable is not referencing to that object, then gc() thought that object is waste so we have to delete it.

User Define Exception :-

→ User Define exception can be checked or unchecked exception.

→ To create Checked Exception we have to use "Exception" Class.

→ To create Unchecked Exception we have to use "RuntimeException" Class.

Ex:-

```
Public class InvalidAgeException extends RuntimeException {
    Public InvalidAge Exception ()
    {
        super();
    }
    Public InvalidAgeException (String message){
        super (message);
    }
```

```
Class voting {
    P.S.V.M ( String [] args) {
    Scanner sc = new Scanner (System.in);
     S.O.P(" Enter your age");
    int age = sc.nextInt();
    if (age < 18) {
        try {
            throw new InvalidAge Exception ();
            }
         catch ( InvalidAgeException e) {
             S.O.P(e.getMessage());
        }
    }
    else {
        S.O.P (" you can vote");
        }
    }
}
```