Java

Date _/_/_

* Language :- Language is a medium which is used to communicate between two entities.
(Entities are nothing but communities (group of people))
Ex- English, hindi, kannada etc

* Programming language is a formal language that specifies a set of instructions for a computer to perform specific tasks.
Example ⇒ C, C++, C#, Java, Python etc

How the languages is built :-

Alphabets
(A-Z) :- uppercase
(a-z) :- lowercase
letter → word → sentence → paragraph → story

How the programming languages is built?

(A-Z) :- uppercase
(a-z) :- lowercase
(0-9) : Numeric values
$@#_ : Special Symbols
→ Tokens → Statements → Programs/code → Software

Tokens :→ Token is the smallest unit/part of java programming language.

↳ Classification of Tokens :- (i) Identifier
(ii) Literals
(iii) Operators
(iv) Comments
(v) Separators
(vi) Keywords

(i) Identifiers :- A name in java program is called identifier, it can be class name or variable name or method name or label name.

Rules :- (1) The only allowed characters in java identifier are:

A-Z
a-z
0-9
_
$

→ If we are using any other character we will get compile time error.

Date _/_/_

Rule-2 Identifier can't start wit digit. but can start with special symbol $ and _.

Rohit ✓	9Wrong ✗
Priyanshu ✓	$Money ✓
5total ✗	_income ✓

Rule-3 Java Identifiers are case sensitive.

int Number = 10; } Both are different
int number = 5; }

Rule-4 True, false and Null can't be used as identifier.

Rule-5 Keywords are not be used as identifier in Java.

(ii) Keyword :- Keyword are the predefined words (inbuilt words) in Java.
↳ Each and Every keywords has its own meaning.
↳ There are 50 keywords in Java.

Abstract	Continue	for	new	switch
Assert	default	goto	package	Synchronized
Boolean	do	if	private	this
break	double	implements	protected	throws
byte	else	import	public	throw
case	enum	instanceof	return	transient
Catch	extends	int	Short	try
char	final	interface	Static	void
class	finally	long	StrictJP	volatile
const	float	native	Super	while

(iii) Literals :- Literals are the values given in the java program.

(a) Numeric literal → (i) Integer literal :- Any no which is not having decimal value.
e.g - 10, 20, -30 etc

(ii) Decimal literal :- Any no. which is having decimal values.
e.g - 9.86, 7.21, etc

(b) String literal :- Anything which is enclosed within double quotes it will be called as string literal.
e.g → "Priyanshu"

Date _/_/_

(c) Character Literal :- Anything which is enclosed within single quote it will be called as character literal.

e.g ⇒ 'A' '$' ' ' etc.

(d) Boolean Literal :- true and false are boolean literals.

(iv) Operators :-

(v) Comments :- Comments are used to provide additional information or to skip any particular line of code.

(i) Single line comment (//) :- It is used to skip single line.

(ii) Block comment (/* --- */) :- It is used to skip multiple line.

(vi) Separator :- If we want to separate one line from the another line of code we will go for separators.

, : ; { } [ ] ( )

Note :-

Q. What is an application ?

⇒ An application is a computer software package (set of programs) that performs a specific function directly for an end user or in some case for another application.

Application
- Stand Alone (Doesn't need Internet to operate)
- Web application (need Internet to operate)

Q. Explain translator and its types

⇒ Translator typically refers to a software tool that converts code written in one programming language into another form.

There are three common types of translator.

(i) Compiler :

A compiler translates the entire source code of a program written in high-level programming to machine code or an intermediate code.

(ii) Interpreter

An interpreter translates and executes the source code line by line.

(iii) Assembler

An assembler translates assembly language code into machine code.

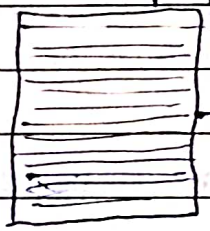
Date _/_/_

Q. What is Java?

→ Java is a popular programming language used to create software applications. It is known for its versatility, allowing developers to write code that can run on different devices without needing to be rewritten. (or Platform independent)
Java is object oriented, meaning it organizes code into reusable components called objects. It's widely used in web development, mobile apps, enterprise software and more.

Java Architecture

Notepad/Editplus

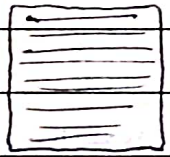


→ Save → filename.java → e.g Sum.java

Q. TE and developer, we will write the code by looking of source code

Command Prompt (2 operation)

Compilation



(i) It will check for the syntax of the code.

(ii) It will check for the rules of the code.

(iii) .java → .class
↓
intermediate code / bytecode

→ It can't be understood by human and also not understood by machines

Interpretation

→ Input

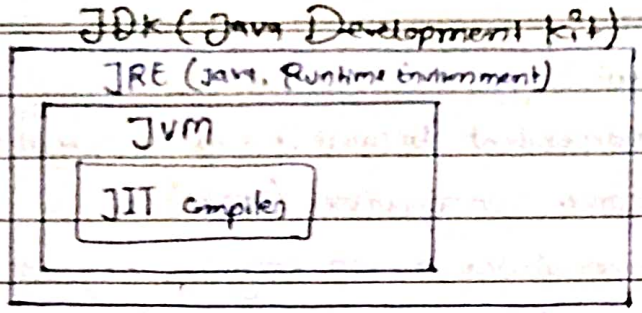
(i) It will read the code line by line and

(ii) execute the code.

(ii) .class file → Binary format [machine readable or machine understandable format].

C.TE (compile Time Error):- If developer or TE does not follow rules and syntax, then the code throws the Compile Time Error.

Date _ / _ / _



Java architecture refers to the structure and components that makes up the Java programming language and its runtime environment.

(i) Java Source code

→ Developers write programs in the Java programming language. This source code contains instructions that define how the program should function.

(ii) Compilation:

→ The Java source code is then compiled into an intermediate form called bytecode. This bytecode is not machine specific and can run on any device that has a Java Virtual Machine (JVM)

(iii) Java Virtual Machine (JVM)

→ The JVM is like a virtual computer that executes Java bytecode. It translates the bytecode into machine code that the hardware can understand.

(iv) Java Runtime Environment (JRE):

→ The JRE includes the JVM along with libraries and other files needed for running Java applications. It provides the necessary runtime environment to execute Java programs.

(v) Java Development kit (JDK):

→ The JDK is a comprehensive package that includes the JRE and additional tools needed for Java development. Developers use the JDK to write, compile and debug Java applications.

(vi) Just-In-Time Compiler (JIT):

→ Within the JVM, the JIT compiler optimizes and translates bytecode into machine code at runtime.

Date _/_/_

Q. Why Java is platform independent?

A. Java is considered platform independent because it uses a special approach in its execution called "Write once, run anywhere." (WORA).

(i) Write once ⇒ When a programmer writes a Java program, they use a Java compiler to convert the human readable code into a special type of code. called bytecode. This bytecode is not specific to any particular operating system or hardware.

(ii) RunAnywhere ⇒ Instead of directly running the bytecode on a specific machine. Java uses a virtual machine called Java Virtual machine (JVM) The JVM is specific to each operating system and its job is to interpret and execute the bytecode. Therefore as long as there is a JVM available for a particular operating system, the Java programs can run on it without any modification in code.

History of Java

Founder → James Gosling

Year → 1991

↳ The name of James Gosling Team is green team. (In green team there are four members along with James)

↳ Firstly, it was called "Greentalk".

Oak tree ← OAK

Symbol of strength in germany

↳ later it have legal issue with Oak Technologies files the case against James gosling & team.

Java → 1995

Class :- Class is a blueprint/Template which is used to create an object.

Syntax:- class classname → identifier

e.g.:- class Rohit

Class Action → class declaration

{

class body { Main declaration → public static void main(String[] args) → main method declaration (for main method signature)

Arguments

Date _/_/_

{ access specifier, modifier, return type, method name

Main body { System.out.println("Hello world"); → print statement

}

}

Steps :-

(i) Save the program → filename.java → Action.java

(ii) Compilation

javac filename.java

e.g. javac Action.java

(iii) Interpretation

java filename

e.g. java Action

Q. WAP to print your name by using java.

```
=> Class Name {
       public static void main (String [] args)
       {
           System.out.println (" Rohit Gupta");
       }
   }
```

Q. WAP to demonstrate String literal, character literal, Integer literal, decimal literal, boolean literal.

```
=> Class literal
   {
     public static void main (String [] args)
     {
        System.out.println ("Hello world");
        System.out.println ("1");
        System.out.println (420);
        System.out.println (11.11);
        System.out.println (false);
     }
   }
```

(i) Integer + Integer = Summation / Addition

e.g → 20 + 20 = 40

(ii) String + Integer = Concatenation

e.g → "Rohit" + 20 = "Rohit20"

(iii) Integer + String = Concatenation

e.g → 40 + "Rohit" = "40Rohit"

(iv) String + String = Concatenation

e.g → "Rohit " + "Gupta" = "Rohit Gupta"

(v) $\frac{\text{Decimal}}{\text{Decimal}}$ = Decimal

e.g → $\frac{1.0}{4.0} = 0.25$

(vi) $\frac{\text{Decimal}}{\text{Integer}}$ = Decimal

e.g → $\frac{1.0}{4} = 0.25$

(vii) $\frac{\text{Integer}}{\text{Decimal}}$ = Decimal

e.g → $\frac{1}{4.0} = 0.25$

(viii) $\frac{\text{Integer}}{\text{Integer}}$ = Integer

e.g → $\frac{1}{4} = 0$

Date _ / _ / _

(#) Member of the class :-

(i) Variables :-

(ii) Method :-

(iii) Constructor

(1) Variable

Variable is a named memory location which is used to store some value and it can be changed n- no of time during execution.

Syntax (i) Variable Declaration :

Syntax : Datatype variablename ;

e.g → int a ;

(ii) Variable Initialization :-

Syntax : variablename = value ;

e.g → a = 100 ;

(iii) Variable Utilization :-

Syntax :- System.out.println (variablename) ;

e.g → System.out.println (a) ;

(iv) Variable declaration and initialization

Syntax :- Datatype variablename = value ;

e.g → int a = 5 ;

(v) Variable reinitialization

Syntax :- variablename = new value ;

a = 35 ;

(vi) Copying the one variable value to another variable value :-

Syntax :- int x = 25 ;
int y = x ;

Date _/_/_

(11) Lifecycle of Variables :-

Variable Declaration
↓
Variable Initialization
↓
Variable Utilization

Classification of variables :- (a) Local Variable
(b) Global Variable

(a) Local variable :- → Any variable which is declared inside the method it will be called as local variable.
→ Scope/visibility of local variable is inside the method.
→ Local variable can not be classified as static or non-static.
→ Local variable does not have any default value.
→ once after local variable is declared, it has to be initialized before Utilization.

(b) Global variable :→ Any Variable which is declared outside the method inside the cl it will be called as global variable.
→ Scope/visibility of global variable from starting of the class till end the class.
→ Global variable can be classified as static or non-static.
→ Global Variable will have default value.
→ once after global variable is declared, it can not be initialized or re-initialized immediately in the next line.

```
Class Type1
{ Static int x;        → Global variable
  public static void main(String [] args)
  { int a;             → local variable
    a = 50;
    x = 30;
    System.out.println(a+x);
  }
}
```

Date _/_/_

Primitive datatypes

- **Numeric datatypes** (To represent Number)
  - **Integer datatypes**
    - byte
    - short
    - int
    - long
  - **Floating-point datatypes**
    - float
    - double
- **Char datatypes** (To represent character)
- **Boolean datatypes** (To represent logical value)

Datatype	Default value (→ Global variable)
byte	0
short	0
int	0
long	0
float	0.0f
double	0.0 or 0.0d
char	Unicode or '\u0000'
boolean	false
String	Null

```
class classname
{   static int a=5;
    public static void main(String[]args)
    {   a=10;
        System.out.println(a); // 10
    }
}
```

```
class classname
{   static boolean a;
    public static void main(String[] args)
    {   System.out.println(a); //false
    }
}
```

```
class classname
{   public static void main(String []args)
    {   static int a=5;
        System.out.println(a);
    }
}
```

o/p- Compile Time Error

Static can't be use with local variable

Q. What is the difference b/w each and every primitive datatype.

	Size (bits)	Range or Capacity (-2^{n-1} to $2^{n-1}-1$)
byte	8	-128 to 127
short	16	-32768 to 32767
int	32	-2^{31} to $2^{31}-1$
long	64	-2^{63} to $2^{63}-1$
float	32	-2^{31} to $2^{31}-1$
double	64	-2^{63} to $2^{63}-1$
char	16	-2^{15} to $2^{15}-1$
boolean	1	true false

Note :→ In C language, the size of int is varied from platform to platform. For 16 bit processor it is 2 bytes but for 32 bit processor it is 4 byte. but in Java the size of int is always 4 bytes irrespective of any platform

Date _/_/_

e.g:
```
Class Smartphone
{
    public static void main(String[] args)
    {
        String mob_brand = "Motorola";
        String mob_color = "Blue";
        float mob_price = 12499.00f;
        System.out.println("Mobile brand = " + mob_brand);
        System.out.println("Mobile color = " + mob_color);
        System.out.println("mob_price = " + mob_price);
    }
}
```

o/p
```
Mobile brand = Motorola
Mobile color = Blue
Mob_price = 12499.00
```

(11) Operator → operator is a predefined symbol which is used to perform specific task

→ (2) (1) (3) operands
operator

Types of operator on no. of operands
(i) Unary operator
(ii) Binary operator
(iii) Ternary operator

Types of operator
(i) Arithmetic operators
(ii) Unary operators
(iii) Assignment operators
(iv) Relational operators
(v) Logical operators
(vi) Ternery operator
(vii) Bitwise operators
(viii) Shift operators
(ix) Instance of operators

(i) Arithmetic Operators

* / % + -

```
class Arithmetic
{
    public static void main(String[] args)
    {
        int a=10;
        int b=5;
        System.out.println("a+b:"+(a+b));   // 15
        System.out.println("a-b:"+(a-b));   // 5
        System.out.println("a*b:"+(a*b));   // 50
        System.out.println("a/b:"+(a/b));   // 2
        System.out.println("a%b:"+(a%b));   // 0
    }
}
```

Date _/_/_

(ii) Unary operators

(++) Increment operator

++a (pre-increment): Value is incremented first and then the result is completed.

a++ (post-increment): Value is first used for computing the result and then incremented.

(--) Decrement operator

--a (pre-decrement): Value is decremented first and then the result is computed.

a-- (post-decrement): Value is first used for computing the result and then decrement.

```
class Unary
{
    public static void main(String [] args)
    {
        int x = 10;
        int y = 5;
        System.out.println("preincrement: " +(++x));    // 11
        System.out.println("Post increment: " + (x++));  // 11
        System.out.println("predecrement: " + (--y));    // 4
        System.out.println("postdecrement: " + (y--));   // 4
    }
}
```

(iii) Assignment Operator → It is used to assign a data to a variable.

=, +=, -=, *=, /=, %=

x+=10 → x=x+10

(iv) Relational Operators

The return type of relational operator is boolean.

>, <, >=, <=, ==, !=

Date _/_/_

(v) Logical operators

&& , Logical AND → returns true when both conditions are true

|| , Logical OR → returns true if at least one condition is true

! , Logical NOT → returns true when a condition is false and vice-versa.

(vi) Ternary operators

condition ? if true : if false

Q. WAP to find max among three numbers

```
class Max
{ public static void main(String[] args)
  { int a = 20;
    int b = 40;
    int c = 25;
    System.out.println("Max number among 20, 40, 25 is " +
                       a>b ? (a>c ? a : c) :
                       (b>c ? b : c));
  }
}
```

(vii) Bitwise operation

& , Bitwise AND operator

| , Bitwise OR operator

^ , Bitwise XOR operator

~ , Bitwise complement operator → returns one's complement i.e, will all bits inverted

XOR

	1	0
0	1	1
0	0	0
1	0	1

AND

	1	1
0	1	0
1	0	0

OR

	0	1
1	0	1
0	1	1
1	1	1

(viii) Shift operators

<<, left shift : Shift the bits of the number to the left and fills 0 on voids left as a result. Similar effect as multiplying the number with some power of two

>>, right shift : Shift the bits of the number to the right and fills 0 on voids. Similar effect as dividing the number with some power of two.

Date _/_/_

```
Class Shift
{
    public static void main(String [] args)
    {
        int a = 32;
        int b = 16;
        System.out.println("a = " + a>>1);      → a = 16
        System.out.println("b = " + b<<1);      → b = 32
    }
}
```

(ix) instanceof operator

↳ The instance of the operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass or an interface.

(#) Methods :-

```
class Methods → classname
{
    public static void main (String [] args)
    { Access specifier   Modifier   Method name
                  ↑ returntype        ↑ arguments (optional)
    }
}
```

↳ Method is a block of statement which will get executed whenever it is called or invoked.

Syntax of Method

```
<access_modifier> <return_type> <method_name> (list_of_parameters)
{
    // body
}
```

Advantage of Method

- Code reusability
- Code optimization

Date _ / _ / _

Method Calling

↳ The method needs to be called for use its functionality.

Whenever we called any method there can be three situation where the method returns to the code that invoked it

(i) It completes all the statements in the method

(ii) It reaches a return statement

(iii) Throws an exception

```
Class Classname  → Class signature or class declaration
{
    → Static void Sum( )  → method declaration or signature
Method body  { int a = 5;
               int b = 3;
               System.out.println("a+b: " + (a+b));
             }
JVM → public static void main(String []args)  → main method declaration
Main method body  { Sum(); }
}
```

o/p

a+b: 8

Types of Methods in Java

(i) **Predefined** ⇒ The method that is already defined in the Java class libraries is known as predefined methods.

(ii) **User-defined Method**

⇒ The method written by the user or programmer is known as a user-defined method.

(#) **Method with parameter**

⇒ Calling or invoking the method by passing some parameters or values

Date _/_/_

(*) Method with returnType

Whenever we want to perform some operations inside one method body and if we want to return the output of that method body to another method body then we will go for method with returntype.

Note:- In java there are 11 returntype

1. Void
2. byte
3. Short
4. int
5. long
6. float
7. double
8. char
9. boolean
10. String
11. Classtype

→ Among these 11 returntype, void is only one returntype that will not return any value in java.

```
Class Sum
{
    Static int Sum(int a, int b)
    {
        System.out.println("a+b = "+(a+b));
    }
    JVM → Public static void main(String[] args)
    {
        System.out.println("main starts");
        int c = sum(5,6);
        System.out.println("main ends");
    }
}
```

O/p: Main starts
a+b = 11
main ends

Q. WAP to find the area of circle by using method with returntype.

```
⇒ Class Circle
{
    Static float area(int r)
    {
        float aes = (float) 3.14*r*r;
        return aes;
    }
    JVM → public Static void main(String [] args)
    {
        System.out.println("Main starts");
        int ans = area(4);
        System.out.println("Area of the circle of radius 4 is " + ans);
        System.out.println("main ends");
    }
}
```

Q. WAP to demonstrate method or calling method between the classes.

⇒

```
class Demo
{
    static void xyz()
    {
        int a = 10;
        int b = 20;
        int c = b-a;
        return c;
    }
}

class main
{
    public static void main (String[] args)
    {
        System.out.println("b-a= " + demo.xyz());
    }
}
```

→ If we want to access the static method b/w the classes.
we should use the syntax.
classname.methodname();

Static :-

→ Any member of class which is declare with the keyword static is called as static member of class
→ static is always single or one copy.
→ static is always associate with the class.
→ All the static member of the class will get stored inside "SPA" (Static pool Area).
→ If we want to access static members b/w the classes we should use the syntax.

```
classname.methodname()
classname.variable()
```

(or)

```
class Sample
{
    variable → static
             → non-static
    method → static
           → non-static
    constructor
        is always non static
}
```

Date _/_/_

Q. WAP to demonstrate Static members between the classes.

```
class Demo
{
  Static int a=99;
  Static void xyz()
  {
    System.out.println("Hey I'm Static method");
  }
}
class Main
{
Jvm ---> Public Static void main(String[] args)
  {
    Demo.xyz();
    System.out.println(Demo.a);
  }
}
```

(1) Non-Static

Any member of the class which is declared without the keyword "Static" is called as non-static member of the class.

* Non-static is always associated with the "object"
* Non-static is always multiple copies.
* All the non-static member of the class will get stored inside "heap memory".
* If we want to access non-static member of the class inside another method body we should use the syntax:-

  (i) object.methodname();

  (ii) object.variablename;

Syntax for object creation:-

new classname();

(keyword) (constructor)

e.g `new Demo();` → object

Date _/_/_

Q. WAP to demonstrate method or calling method as 'non-static' within the class

→
```
class Demo
{ int x = 50;
  void xyz()
  { System.out.println("Inside Democlass");
  }
public static void main (String[] args)
  { new Demo.xyz();
    System.out.println(" main class and value of x = "+ new Demo().x);
  }
}
```

Q. Why non static is always associated with the object?

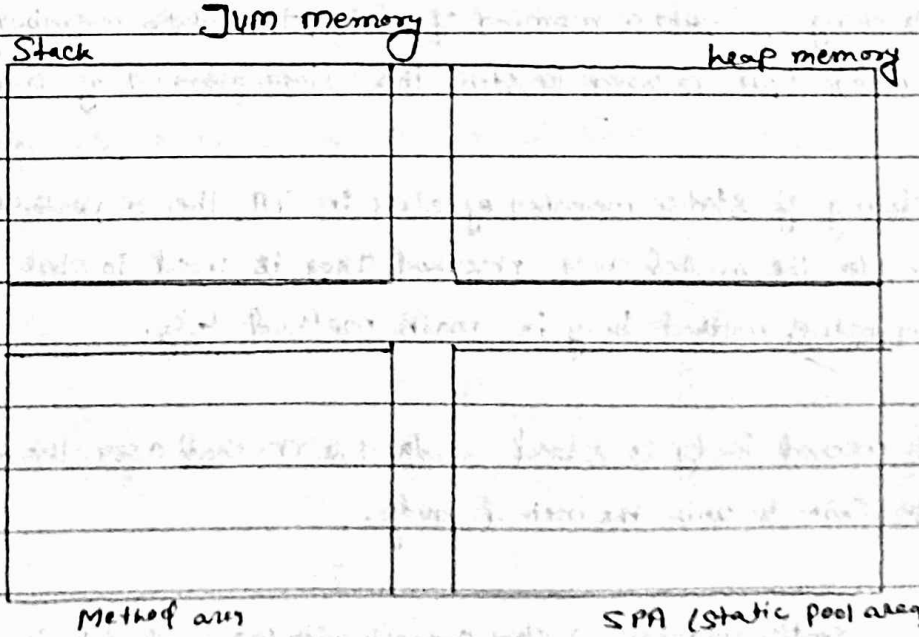
⇒ Here, if I want to access anyone of the non-static member of the class, Object creation is compulsory.

Java Memory Diagram

Object creation Syntax:

```
new classname();
```

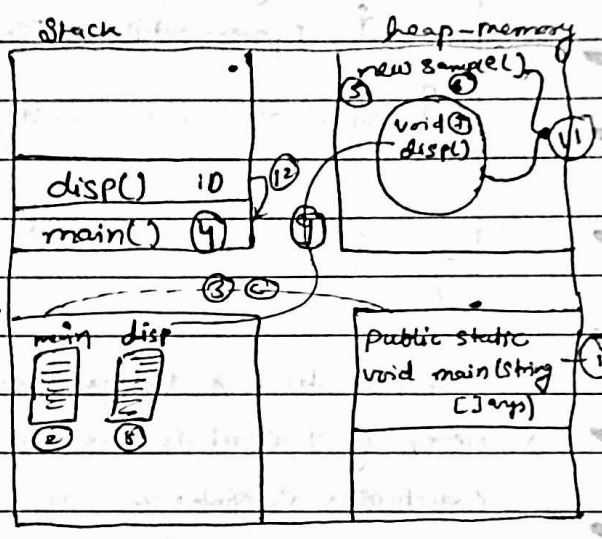
- `new` → Keyword or operator
- `classname()` → constructor
  - ↳ It will initialize or store the non-static members of the class inside heap memory



Date _/_/_

Q. WAP to demonstrate method as calling method as non-static and diagram. Explain it with JVM-memory diagram.

→
```
class Sample
{
    void disp()
    {
        int a = 2;
        int b = 4;
        int c = b-a;
        System.out.println(c);
    }
}
class Jvm
{
    public static void main(String[] args)
    {
        new Sample().disp();
    }
}
```



:→ First class loader works first, he will goes inside the Java program and search for the static member of the class, for above program the main method is only static member of the class.

e.g:→ public static void main(String[] args)

:→ After the searching of static member of class, the static member is stored in the SPA (static Pool Area), SPA is used to store the static member of class.

:→ After the storing of static member of class in SPA, the respective method body will be stored inside the method area, method area is used to store the all method body Here the respective method body is main method body.

:→ once after method body is stored inside the method area then the static member of the class connects with the method body.

:→ And once the static member of class connects with the method body then it stored inside the stack, stack is used for execution process.

Date _ / _ / _

:- once after execution is done then JVM comes inside the java program and it will search for the main method, after the searching of main method, the controller goes inside the main method and executes the program top to bottom, left to right and one by line. So the controller executes the first line of code. And then one method called or invoked inside another method.

:- then controller goes inside the submethod which is non-static

:- But before proceeding for the next line, here, 'new' keyword is used which create random memory space inside the heap-memory and "Sample" class name or constructor is created which initializes or store the non-static value inside heap memory.

:- So, the non-static member of the class is stored inside the heap memory, Heap memory is used to initialize or store the all kind of non-static member of the class.

:- once after non-static member of the class is stored then the respective method body of the class will be stored inside the method area.

:- once after method body is stored inside the method area then the class member connects with the method body.

:- After the connection of the class member and method body, It will stored inside the Stack, Stack is used to execution purpose.

:- Now, Controller goes inside the sub-method and executes each lines of program in proper manner.

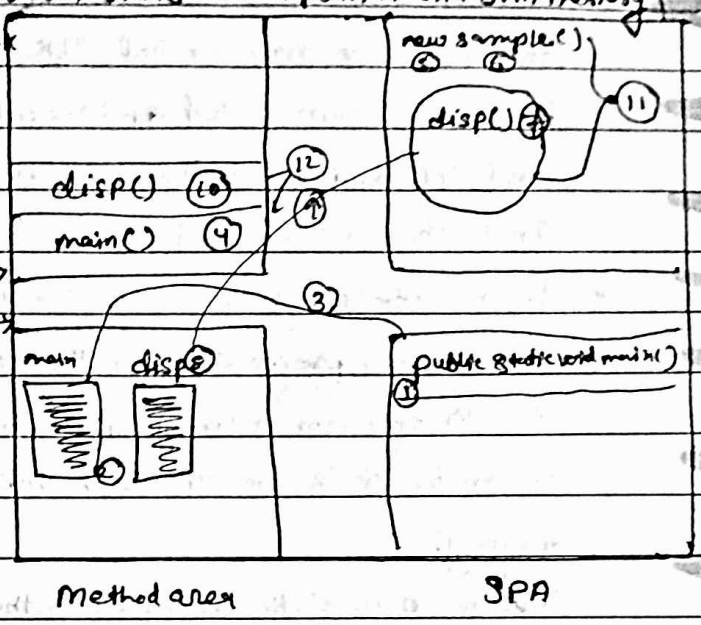
:- After the completion of each and every lines execution then controller goes back to its initial position that means it goes inside main method from sub-method.

:- After that whole, program will be executed then first method body will be terminated and at last the class body will be terminated and after that we will get our output for the program.

Date _ / _ / _

Q. WAP to demonstrate method with parameter as non-static and explain it with JVM memory diagram

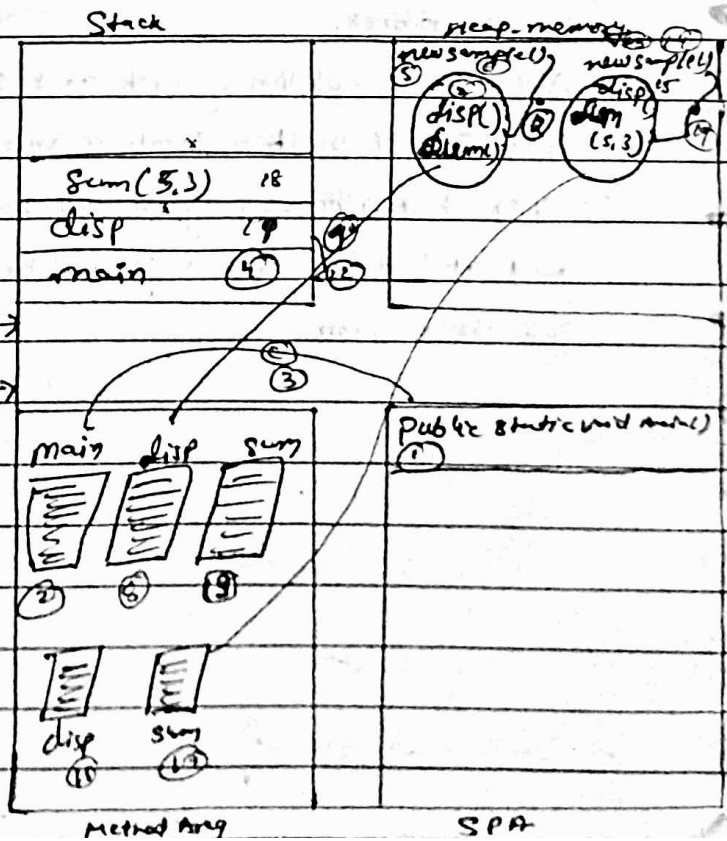
```
class sample
{
    void disp(String s)
    {
        S.o.p("my name is "+s);
    }
    public static void main(String[] args)
    {
        new sample().disp("Rohit");
    }
}
```



Q. Why reference variable?

Ans:- Here, If I want to access multiple non-static members inside another method body, I should create object again and again multiple times by using new keyword. If I use new keyword again and again multiple times it creates random memory space again and again n-no of time inside the heap memory. It leads to more memory consumption. If the memory consumption is more performance of the application will be poor. In order to overcome this drawback we will go for a concept called reference variable.

```
Class Sample
{
    void disp1()
    {
        S.o.p("Hello");
    }
    void sum(int a, int b)
    {
        S.O.p("sum = "+(a+b));
    }
    public static void main(String[] args)
    {
        new sample().disp();
        new sample().sum(5, 3);
    }
}
```



Date _/_/_

(iii) Reference Variable :- It is a special type of variable which is used to store a value, that is object address or null.

Reference Variable lifecycle :-

Reference Variable Declaration
⇓
Reference variable initialization
⇓
Utilization

(i) Reference variable declaration :-

```
punter P;
```

(ii) Reference variable Initialization :-

```
reference variable_name = Object;
        P = new Punter();
```

(iii) Reference variable Declaration and Initialization in a single line :-

```
Classname reference variable_name = Object;
        punter p = new punter();
```

Note :- Whenever both the classname and constructor name are same then this kind of object creation it will be called as homogeneous object creation.

(iv) Reference variable Utilization :-

```
System.out.println(reference variable);
```

eg ⇒ S.O.P(P);

o/p ⇒ object address or (Fully qualified path)
⇓
Syntax
packagename.classname@Hexadecimal number.

Date _/_/_

Constructor :→

↳ Constructor is a special type of method or member of the class, which is used to initialize the data member or variables.

Rules of constructor :-

(i) Classname and constructor name should always be same.

(ii) Constructor doesn't support any return type.

(iii) Constructor is always non-static.

(iv) Whenever an object is created, constructor will get invoked.

Syntax

```
Class Classname
{
    Classname ( )
    {
    }
}
```

Q. WAP to demonstrate Constructor.

```
Class Doubt
{
    Doubt( )
    {
        System.out.println("Inside constructor");
    }
}

JVM → public static void main (String [] args)
{
    new Doubt( );
}
}
```

```
Class Apple
{
    String y;
    Apple (String x)
    {
        y = x;
    }
    public static void main (String [] args)
    {
        Apple a = new Apple("Java");
        System.out.println(a.y);
    }
}
```

Date _ / _ / _

Note :- If we want to return value from the constructor, it is not possible because constructor doesn't supports any return type.

Solution :- (i) Declare the global variable

(ii) Copy the value from local variable to global variable.

(iii) Access the global variable wherever we want.

Q. WAP to print employee details using constructor.

⇒
```
class Employee
{ int E_id;
  String name;
  int Sal;
  Employee(int id, string name, int salary)
  { E_id = id;
    name = Name;
    Sal = salary;
  }
  public static void main (String [] args)
  {
    Employee E1 = new Employee(102, "Rohit", 50000);
    System.out.println(E1.id + " " + E1.name + " " + E1.sal);
  }
}
```

o/p

102 Rohit 50000

Note :- When to go for "This" keyword ?

Whenever both the local and global variables are same, then in order to differentiate between them we will go for "This" keyword.

```
class student
{ int roll;
  String Sname;
  Student(int roll, String Sname)
  { this.roll = roll;
    this.Sname = Sname;
  }
  public static void main (String[] args)
  { student S1 = new Student (721, "Rohit");
    S.O.P ("Student roll no: " + S1.roll);
    S.O.P ("student name: " + S1.name);
  }
}
```

o/p

Student roll no: 721

Student name: Rohit

Date _/_/_

Q. Why static is always single copy?

Ans:- Class loader will load all the static member of the class inside static pool area in a single short that's why static is always single copy.

Q. Why non-static is always multiple copy?

Ans:- If I want to access multiple non-static member, I should create object by using new keyword multiple times. Since we are creating object multiple times to access multiple non-static members for that reason non-static is always multiple-copy.

⇒ SIB and IIB :-

* SIB (Static Initialization Block):-

↳ Any Block which is declared with the keyword static is called as Static initialization Block.

↳ SIB will get executed before main method.

↳ We can have n no of SIB's inside one single class.

↳ Order of execution of SIB is sequential.

Syntax:-

SIB ← static { ≡≡≡ } Statements

e.g.

```
class Demo
{
static
{
S.O.P("Hey I'm in SIB");
}
public static void main(String[] args)
{
S.O.P("main starts");
S.O.P("main ends");
}
}
```

O/P

Hey I'm in SIB
main starts
main ends

Q. Can we execute any statement before main method?

Ans Yes, we can execute any statement before main method because of SIB.

Date _ / _ / _

* IIB (Instance Initialization Block):-

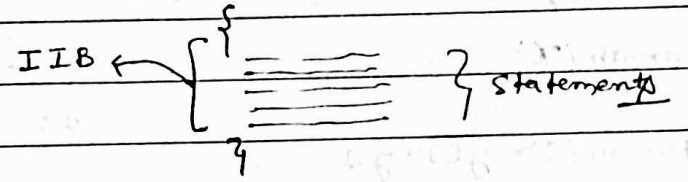
→ Any block which is declared without the keyword static is called as Instance Initialization block.

→ IIB will get executed whenever an object is created.

→ We can have n-no of IIB's inside one single class.

→ order of execution of IIB is also sequential

Syntax :-



e.g:- Class Sample

```
{
    {
        S.O.P ("Inside Instance");
    }
    public static void main (String [] args)
    {
        S.O.P ("main starts");
        new Sample();
        S.O.P ("main ends");
    }
}
```

O/P

main starts
Inside Instance
main ends

e.g

```
class sample {
    static {
        S.O.P ("static Initialization Block");
    }
    {
        S.O.P ("Instance Initialization Block");
    }
    public static void main (String [] args)
    {
        S.O.P ("main starts");
        new sample();
        S.O.P ("main ends");
    }
}
```

O/P

Static Initialization Block
Main starts
Instance Initialization Block
Main ends

Date _/_/_

⇨ Pass by value or call by value :-

Calling or invoking the method by pessing the primitive types of data is called as pass by value or call by value.

eg

```
class demo
{
  static void apple(int x)
  { System.out.println(x);
  }
  public static void main(String[] args)
  {
    apple(100);   // pass by value.
  }
}
```

o/p :-
100

➡ Pass by reference or call by reference :-

Calling or invoking the method by passing the reference variable is called as pass by reference or call by reference.

```
class demo
{ int x = 99;
  static void apple(demo a)
  { S.o.p(a.x);
  }
JVM → public static void main(String[] args)
  { 2
    demo d = new demo();
    apple(d);
  }
}
```

d → Reference variable

```
class QSpiders
{
    void Development_course()
    {
        System.out.println("Joined successfully");
    }
}
```

Date _/_/_

Previously

```
class HR
{
    public static void main(String[] args)
    {
        QSpiders q = new QSpiders();
        Counseller c = new Counseller();
        c.join(q);
    }
}

class Counseller
{
    void join(QSpiders q)
    {
        q.Development_course();
    }
}
```

JVM → HR.main

Q. Why pass by reference?

Ans If I want to access multiple non-static members or same non-static member again and again multiple times then we should create object by using new keyword again and again multiple times, if we use new keywords multiple times it will create random memory space multiple times inside heap memory and it leads to more memory consumption, so In order to overcome this drawback we will go for the concept called pass by reference.

Here, we will create object in a single shot and we will store that object inside one reference variable later we will use that reference variable whenever we want.

➡ Composition :- A class having the object of another class is called as composition.

:→ This composition will be represented by class diagram.

Q. WAP to demonstrate composition.

```
class A
{
    void demo()
    {
        System.out.println("Kuchh bhi");
    }
}
class B
{
    public static void main(String[] args)
    {
        new A().demo();
    }
}
```

JVM → B.main

O/P :

→ Kuchh bhi

Date _ / _ / _

Method Overloading :-

:→ Developing a multiple methods with the same names but variations in the arguments list is called as Method Overloading.

(#) Variations in the arguments list includes :-

(i) Variations in the data type.

(ii) Variations in the length of arguments or datatypes.

(iii) Variations in the order of occurance of the arguments.

(#) Rules of Method overloading :-

(i) Static/Non-static can be overloaded.

(ii) main method can be overloaded.

(iii) Constructor can be overloaded.

(iv) There is no restriction for access specifier, modifier, return type and identifier in method overloading.

Date _/_/_

➡ Method overloading should be happen within the class.

Q. WAP to demonstrate static method overloading.

Ans

```
Class main
{
    static void demo (int i, String s)
    {
        S.O.P( i + " " + s);
    }
    Static void demo (String s, int i)
    {
        S.O.P(s+" "+i);
    }
    Static void demo (int i)
    {
        S.O.P(i);
    }
    Static void demo (String s)
    {
        S.O.P(s);
    }
JVM → Public Static void main (String [] args)
    {
        demo("Rohit");
        demo ("Rohit", 22);
        demo (22, "Rohit");
        demo (22);
    }
}
```

O/P:-

```
Rohit
Rohit 22
22 Rohit
22
```

Note:- In method overloading when we call the method by passing parameters then it will search for the matching datatype in the arguments/parameters and execute the method sequentially.

Q. Why method overloading?

Ans Suppose if we give only one or two options for customers to use and that feature if it fails for that option, chances are these customer usage might go down so that feature add to the method overloading.

Q. WAP to demonstrate non-static method overloading.

```
class sample
{
    void disp(String firstname)
    {
        S.o.p("My name is "+firstname);
    }
    void disp(String first, String last)
    {
        S.o.p("My name is "+first+" "+last);
    }
JVM → public static void main(String[] args)
    {
        new sample().disp("Rohit");
        new sample().disp("Rohit","Gupta");
    }
}
```

Date _/_/_

O/P

My name is Rohit

My name is Rohit Gupta

Q. WAP to demonstrate constructor method overloading.

```
class demo
{
    demo(int i, String s)
    {
        S.o.p("My age is "+i+"and my name is "+s);
    }
    demo(String s, int i)
    {
        S.o.p("My name is "+s+" and my age is "+i);
    }
JVM → public static void main(String[] args)
    {
        demo d = new demo();
        d.demo(22, "Rohit");
        d.demo("Rohit", 22);
    }
}
```

O/P

My age is 22 and my name is Rohit

My name is Rohit and my age is 22

Q. WAP to demonstrate main method overloading?

```
class Sample
{
    static void main(String s);
    {
        System.out.println("My country name is "+s);
    }
    static void main(String s, String t);
    {
        S.o.p(s+" means "+t);
    }
    Public static void main(String[] args)
    {
        main("India");
        main("India");
    }
}
```

O/P

My country name is India

India means Bharat

Date _/_/_

⇨ OOPs:-(Object oriented Programming system)

Q. Why Java is called as oops language?

⇒ Because Java supports 6 oops concepts :-

1. Class
2. Object
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Q. Is Java 100% oops language?

⇒ NO, because Java supports eight primitive data type.

byte
short
int
long
float
double
char
boolean

(#) Features of Java :- (i) Java is oops language.
(ii) It is Strong (robust).
(iii) Java is platform independent
(iv) It is highly secure language.
(v) It is simple and Easy to understand.
(vi) High performance.
(vii) It is multithreaded.

(#) Class :- class is a blueprint or template which is used to create an object.

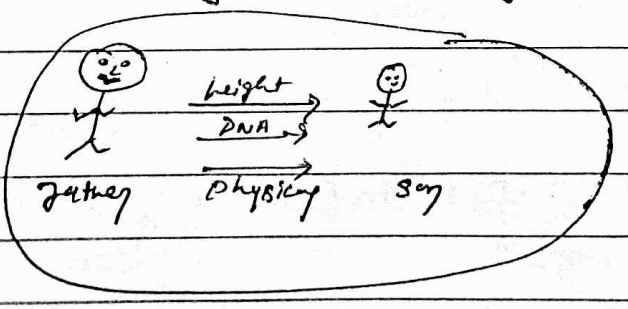
(#) Object :- Object is a realtime Entity (component) which has it's own State and behaviour.

State
State is nothing but what value a nonstatic variable can hold
int z = 100;

behaviour
behaviour is nothing but how the nonstatic method will behave
void demo()
{
}

Inheritance

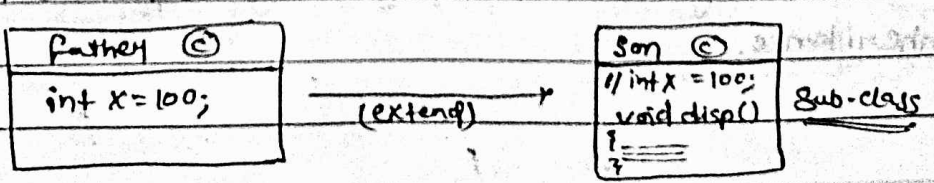
One class is inheriting the properties from another one class is called as inheritance.



IS a relationship

Note:- In Java there will be one rule i.e, Static cannot be inheritance.

Note:- In Java extends is a keyword which used to inherit the properties from one class to another class.

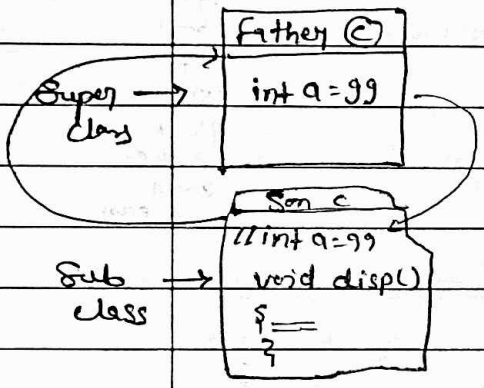


Date _ / _ / _

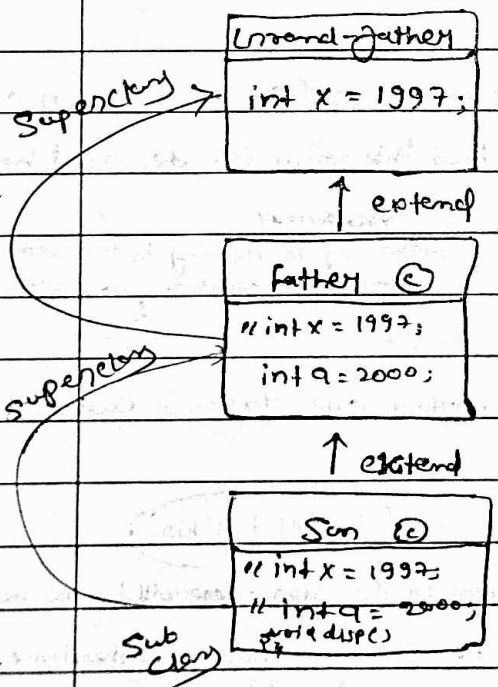
(#) There are five types of Inheritance :-

(a) Single level Inheritance

:→ A subclass inheriting the property from its superclass is called as Single level inheritance.

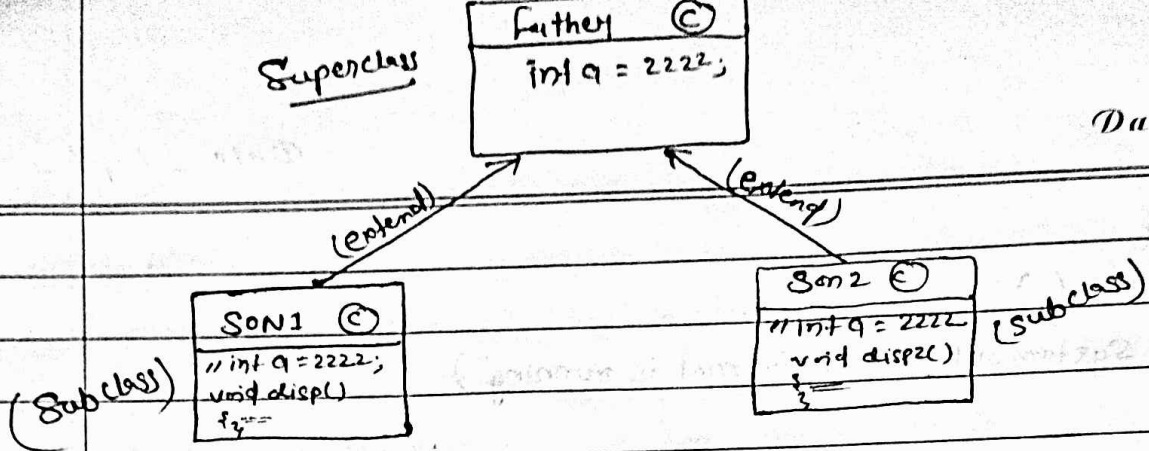


(b) Multi level Inheritance :- A subclass inheriting the properties from its super class which inheriting the properties from its superclass is called as multi-level inheritance.



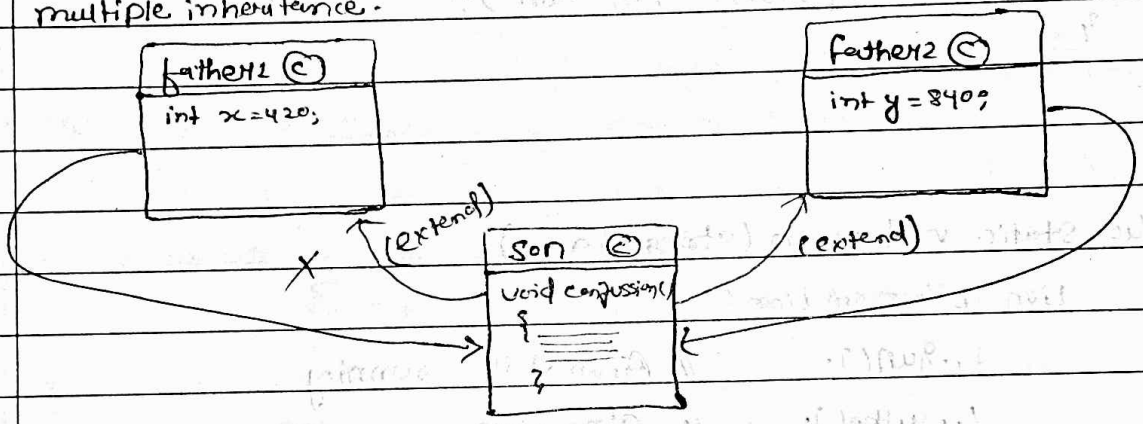
(c) Hierarchical Inheritance :-

→ Multiple subclass inheriting the properties from single super class is called as hierarchical inheritance.



(d) Mupple Inheritance :-

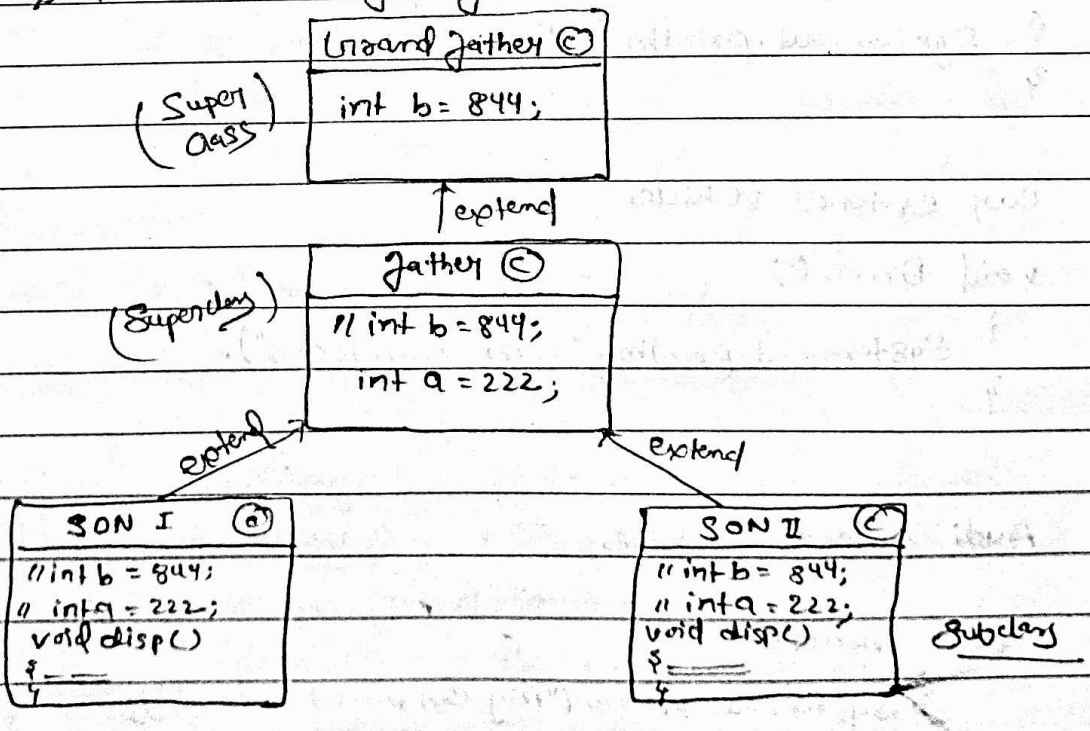
Single subclass inheriting the properties form multiple superclass is called as multiple inheritence.



NOTE :→ Multiple inheritance can't be achieve through class in Java but through multiple interface.

(e) Hybrid Inheritance :-

It is a combination of single level and multilevel and hirarchial inheritance.



Date _/_/_

//Simple Inheritance

```
class Animals{
    void run()
    { System.out.println("Animal is running");
    }
}

class Lion extends Animals
{   void who()
    { System.out.println("I'm Lion");
    }
}

class Main
{ public static void main(String[] args)
    {   Lion L1 = new Lion();
        L1.run();      // Animal is running
        L1.who();      // I'm Lion.
    }
}
```

// Multilevel Inheritance

```
class vehicle
{   void Info()
    { System.out.println("I'm driving a vehicle");
    }
}

Class Car extends vehicle
{   void Drive()
    { System.out.println("I'm driving a car");
    }
}

Class Audi extends vehicle
{
    void model()
    { System.out.println("My Car model is audi");
    }
}
```

Date _ / _ / _

```
Class main
{ public static void main (String[] args)
    { Audi a = new Audi();
      a.Info();      // I'm driving a vehicle
      a. Drive();    // I'm driving a car
      a. model();    // My car model is audi
    }
}
```

// Hierarchical Inheritance

```
Class vehicle
{ void display()
    { System.out.println("I'm a vehicle");
    }
}

Class Bike extends vehicle
{ void bike()
    { System.out.println("I'm a two wheeler");
    }
}

class Car extends vehicles
{ void Drive()
    { System.out.println("I'm a four wheeler");
    }
}

class main
{ public static void main (String[] args)
    { Bike b = new Bike();
      Car c = new Car();
      b.display();   // I'm a vehicle
      b. bike();     // I'm a two wheeley
      c.display();   // I'm a vehicle
      c. Drive();    // I'm a four wheeley
    }
}
```

// Hybrid Inheritance

Date _/_/_

```
class Animal
{   void speak()
    { System.out.println("Animal are speaking");
    }
}

class Dog extends Animal
{   void bark()
    { System.out.println("Dogs are barking");
    }
}

class Cat extends Animal
{   void meow()
    { System.out.println("Cat are meowing");
    }
}

class Streetdog extends Dog
{   void guard()
    { System.out.println("They are guarding our city");
    }
}

class Mainclass
{ public static void main(String[] args)
    {   Cat C = new Cat();
        Dog D = new Dog();
        Streetdog SD = new Streetdog();
        C.speak();      // Animal are speaking
        C.meow();       // Cat are meowing
        D.speak();      // Animal are speaking
        D.bark();       // Dogs are barking
        SD.speak();     // Animal are speaking
        SD.bark();      // Dogs are barking
        SD.guard();     // They are guarding our city
    }
}
```

Date _ / _ / _

(#) Method Overriding

Developing a method inside the subclass with the same name and same signature as in superclass but variation in the subclass implementation.

Notes

(i) Static cannot be overriden.
(ii) There should be is a relationship (inheritance)
(iii) Non-static can be overriden.
(iv) Main method cannot be overriden
(v) Constructor cannot be overriden.

Rules

(i) Method names and signature should be same.
(ii) There should be is a relationship.
(iii) only non-static can be Overriden.

Q. Why Method Overriding ?

⇒ If I want to provide any new implementation or modification for the older features of an application. I will go for a concept called method overriding.

eg Nykaa → Lakme (Product) → Payment → UPI, PhonePay, Cards, Paylater, Paythroughemi

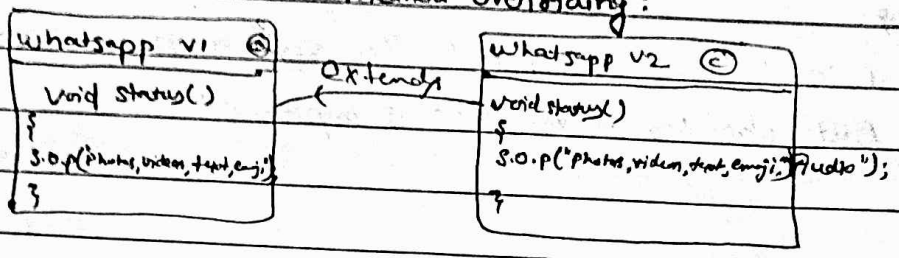
eg Facebook → signup, login, share, comment, reels, marketplace

whatsapp → Status → video, Image, Links, Texts, Audio

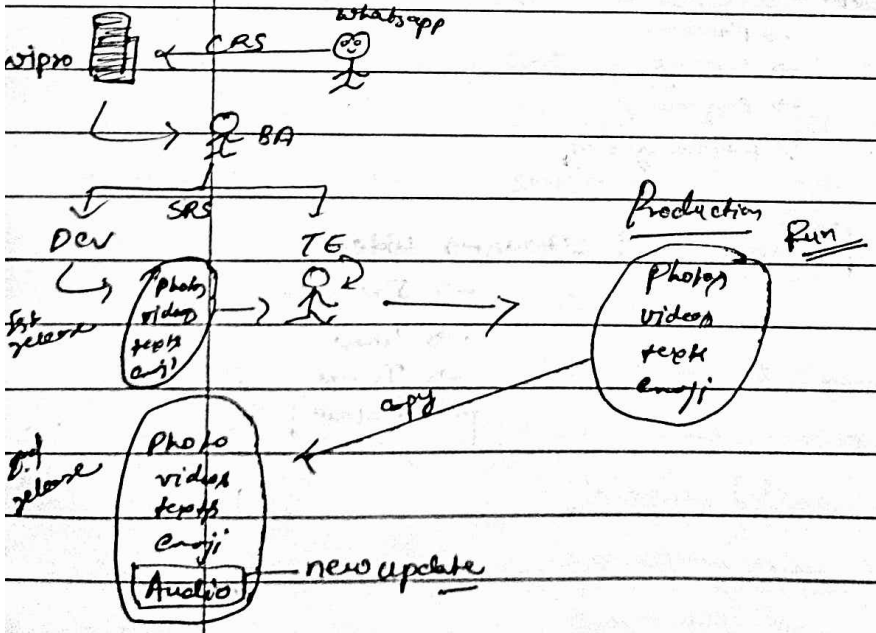
Date _/_/_

Q. WAP to demonstrate method overriding?

A)




```
class whatsapp_v1
{
    void status()
    {
        S.o.p("photos, videos, texts, emoji");
    }
}
class whatsapp_v2 extends whatsapp_v1
{
    void status()
    {
        S.o.p("photos, videos, texts, emoji, videos");
    }
}

class main
{
    public static void main(String[] args)
    {
        whatsapp_v2 w2 = new whatsapp_v2();
        w2.status();
    }
}
```



Date _ / _ / _

Super keyword

Q. When we go for super keyword?

⇒ When we want superclass implementation along with subclass implementation, we will go for super keyword.

```
class Facebook V1
{
    void feature()
    {
        S.O.p("signup, login, comment, post, likes");
    }
}

class facebook V2
{
    void feature()
    {
        S.O.p("signup, login, comment, post, likes, marketplace");
        super.feature();
    }
}

class main
{
    public static void main(String[] args)
    {
        facebook V2 f2 = new facebook V2()
        f2.feature();
    }
}
```

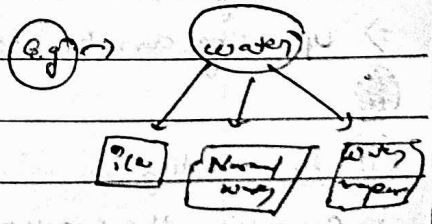
op
=
signup, login, comment, post, likes
marketplace,
signup, login, comment, post, likes

Typecasting

⇒ Converting from one type to another type is called as typecasting.

(i) premitive type casting → widening
→ narrowing

(ii) classtype casting / Derived typecasting → upcasting
→ downcasting



⇒ Converting from one primitive datatype to another premitive datatype is called primitive typecasting.

Date _ / _ / _

(i) Widening :- Converting from Smaller premitive datatype into one of its larger premitive datatype is called as Widening.

Note:- Widening can be done both implicitly and explicitly.

```
double x = 100;  ——> implicitly
    S.O.p(x);  // 100.0
double y = (double)24.3f;  → explicitly
    S.O.p(y);  // 24.3
```

(ii) Narrowing :- Converting from larger premitive datatype into one of it's Smaller premitive datatype is called as narrowing.

i→ Narrowing Should be done explicitly only.

```
int a = (int)69.69;
    S.O.p(a);      // Data loss → o/p → 69
Short x = (Short)266.9f;
    S.O.p(x);      // Data loss → o/p → 266
```

Classtype Casting :- / Derived typecasting

converting from one classtype into another classtype is called Classtype Casting.

(i) Upcasting (ii) Downcasting

(i) Upcasting

⇒ Converting the subclass object into Super class type is called as upcasting.

⇒ Upcasting can be done implicitly and explicitly.

(ii) Downcasting

⇒ Converting the super class object into Subclass type is called as Downcasting.

⇒ Downcasting Should be done explicitly only.

⇒ Before Downcasting we Should always do upcasting.

⇒ Without upcasting there is no Downcasting.

Date _ / _ / _

→ In upcasting we will always get superclass implementation only.

```
Class Bigg
{
    void bada()
    {
        S.o.p("Hello you are bada");
    }
}

Class Small extends Bigg
{
    void Bada()
    {
        S.o.p("Hello you are bada");
    }
    void chota()
    {
        S.o.p("hey you are chota");
    }
}

class main
{
    public static void main(String[]args)
    {
        Bigg S = new Small();
                 ↓
                 new Bigg();
        S.bada();
    }
}
```

Upcasting ←

Downcasting ← `Small z = (Small) S;` → Explicitly

```
z.chota()
z.bada()
```

O/p

Hello you are bada.
hey you are chota.
hey you are bada.

Case Study

Case-1 In upcasting I will get superclass implementation only.

Case-2 In downcasting I will get both subclass and superclass implementation.

Case-3 In method ~~overloading~~ overriding it will get overriden implementation.

Case-4 If use super keyword, I will get superclass implementation along with subclass implementation.

Case-5 Whenever there is a overridden implementation even through I am doing upcasting I will get overriden or subclass implementation only.

Date _/_/_

```
class vidmate_V1
{
    void feature()
    {
        S.o.p("videos, Audio, photos");
    }
}

class vidmate_V2 extends vidmate_V1
{
    void feature()
    {
        S.o.p("videos, Audio, photos, reels");
    }
}

class main
{
    public static void main(String[] args)
    {
        vidmate_V1 V1 = new vidmate_V2(); // upcasting
        V1.feature();
    }
}
```

overridden interpreter

JVM →

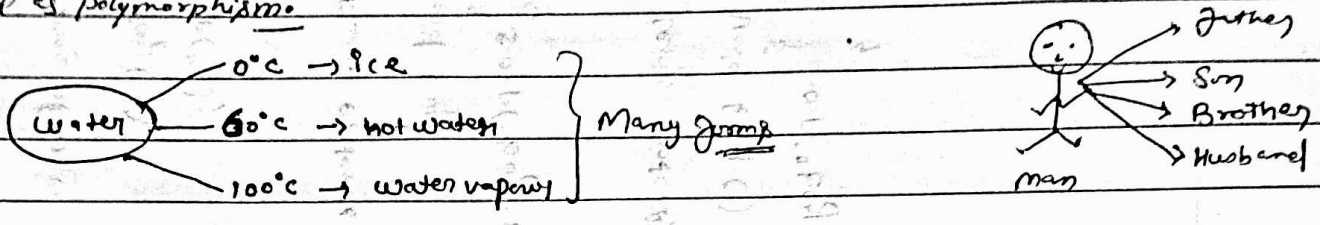
o/p

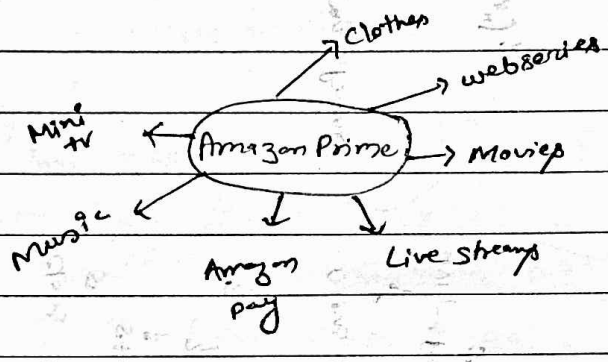
videos, Audio, photos, reels

Polymorphism ← Poly → Many, morphism → form

Date _/_/_

An object showing different behaviour at the different stages of it's lifecycle is called as polymorphism.





Types:-

(i) RunTime polymorphism (ii) Compile Time Polymorphism

(1) Runtime polymorphism (RTP) → Dynamic Polymorphism

(i) method declaration getting binded to its method definition at the run time by the JVM based on the Object created is called as RunTime Polymorphism.

(ii) Since the method declaration is getting binded to its method definition at the runtime it is called as late binding.

(iii) Method overriding is the best example of Runtime Polymorphism.

(2) Compile Time polymorphism (CPP) → Static Polymorphism

(i) method declaration getting binded to its method definition at the compile Time by the compiler based on the arguments passed is called as Compile Time polymorphism

(ii) Since the method declaration is getting binded to its method definition at the Compile Time is called as Early binding.

(iii) Method overloading is the beast example of Compile Time Polymorphism.

Run Time Polymorphism

```
class AmazonPrime
{
    void buy()
    {
        s.o.p("");
    }
}

class puma extends AmazonPrime
{
    void buy()
    {
        s.o.p("buy: shoes");
    }
}

class levis extends AmazonPrime
{
    void buy()
    {
        s.o.p("buy: shirts")
    }
}

class Simulator(AmazonPrime a)
{
    static void spc(AmazonPrime a)
    {
        a.buy();
    }
}

class main
{
JVM→public static void main(String[]args)
    {
        puma p = new puma();
        levis l = new levis();
        Simulator.spc(p);
        Simulator.spc(l);
    }
}
```

Upcasting

subclass: p = new puma(); l = new levis();

superclass

Rule - / - / -

Override @ implement

single point of contact

o/p

buy shoes

buy shirts

Five Points (Requir

(i) Non static
(ii) hierarchical
(iii) method overriding
(iv) pass by reference
(v) upcasting

Date _ / _ / _

Q. WAP to demonstrate RunTime Polymorphism.

A)

```
Class Insurance                                  ← Super class
{
    int calculate(int price)
    {
        int insurredvalue = 0.9* price;
        return insurredvalue;
    }
}
class Bike extends Insurance                     ← subclass
{
    int calculate (int price)
    {
        int insurred_value = 0.8* price;
        return insured_value;
    }
}
Class Car extends Insurance                      ← subclass
{
    int calculate (int price)
    {
        int insurred_value = 0.7* price;
        return insured_value;
    }
}
Class methoodfinance
{
    void findvalue (Insurance I, int price)
    {
        int Estimated = I. calculate (price);
        S.O.P ("Estimated insurance value: " + estimated);
    }
}
class main
{
    public static void main (String a[] args)
    {
        Insurance B = new Bike();
        methoodfinance (B, 75000);
        Insurance c = new Car();
        methoodfinance (c, 500000);
    }
}
```

Overriding implementation

upcasting: new Bike(), new Car()

Reference variable

O/P

Estimated Insurance value : 60000

Estimated Insurance value : 350,000

Date _ / _ / _

Q. WAP to demonstrate CTP (CompileTime Polymorphism).

→

```
Class Apple
{
    Static void banana (int n)
    {
        S.o.p(n);
    }
    Static void banana (String s)
    {
        S.o.p(s);
    }
    Static void banana (int x, String z)
    {
        S.o.p(x + " " + z);
    }
Jvm → Public static void main(String[] args)
    {
        banana (100, "Java");
        banana ("SQL");
        banana (1000);
    }
}
```

o/p

100 Java

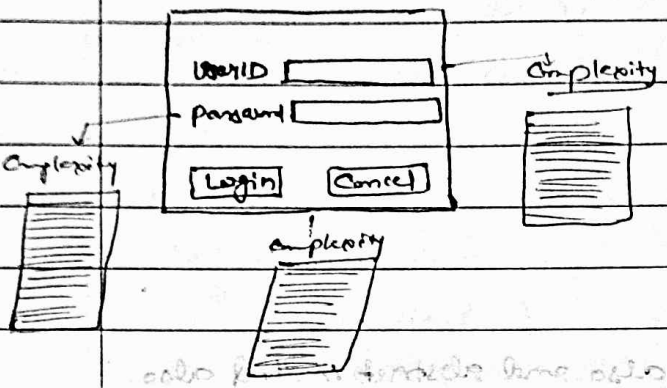
SQL

1000

Abstraction

Date _ / _ / _

=> Hiding the complexity of the system and exposing the essential / required functionality to the end user is called as Abstraction



Abstract class

(1) Complete Method (concrete method) => Any method which is having method declaration and method definition / method body.

```
method declaration  static void disp()
                    {
method body         }
```

(2) Concrete class - Any class which is having only concrete methods is called as Concrete class.

```
class A  ---> concrete class
{
    static void disp()  ---> concrete method
    {
    }
}
```

(3) Abstract method => Any method which is declared with the keyword abstract is called as abstract / incomplete / unimplemented method.

Note:- For abstract method there will be no body.

```
abstract void disp();
```

(4) Abstract class => Any class which is declared with the keyword abstract it will be called as Abstract class.

```
abstract class B
{
}
```

Date _ / _ / _

(5) If any class if it is having atleast one abstract method then that class should be declared as abstract.

```
Abstract class Sample
{
    abstract void operate();
}
```

(6) Abstract class can have both concrete method also and abstract method also.

```
abstract class Z
{
Concrete method ← void sub()
                  {
                  }
Abstract method ← abstract void add();
}
```

(7) For abstract class and Interface we can't create object.

(8) The class which provide implementation for all the abstract method is called as implementation class.

(9) Abstract class cannot be declared as static, final and private.

(10) If any one of the abstract method not provided with the implementation then the sub class should be declared as ~~subclass~~ abstract

(11) Each and every class extends to the ~~abstract~~ object class which is supermost class in java.

(12) Through Abstract class we can achieve upto 100% Abstraction.

(13) For Concrete class I can create object because inside concrete class it holds only concrete ~~one~~ or complete methods.

Date _ / _ / _

Q. WAP to demonstrate Abstract method ?

=>

```
abstract class A
{
    abstract void UN( );
    abstract void pwd( );
}

class B extends A
{
    void UN( )
    {
        S.o.p("xyz");
    }
    void pwd( )
    {
        S.o.p("123");
    }
}

Class Main
{
    public static void main(String[] args)
    {
        B b = new B( );
        b.UN( );
        b.pwd( );
    }
}
```

Date _/_/_

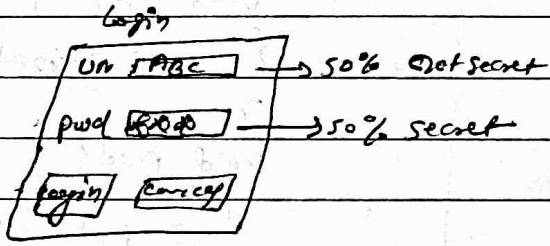
Q. WAP to demonstrate 50% Implementation for abstract method?

⇒

```
abstract class facebook
{
    abstract void UN();
    abstract void pwd();
}

abstract class sample extends facebook
{
    void UN()
    {
        System.out.println("ABCD");
    }
    // abstract void pwd();
}

class Demo extends sample
{
    void pwd()
    {
        System.out.println("123");
    }
}

class main
{
    public static void main(String[] args)
    {
        Demo d = new Demo();
        d.UN();
        d.pwd();
    }
}
```

Abstract method

Concrete Method



O/P

ABC
123

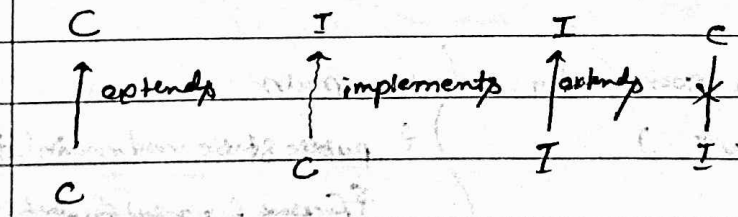
Interface

Date _ / _ / _

=> It is one of the Java type.
=> It is by default abstract
=> Inside Interface we can have only two member that mean Variable and method.
=> All the variable inside interface is static and final.
=> All the method are by default public and abstract.
=> Interface doesn't support constructor.
=> Interface is purely abstract body (it means it only support abstract method).
=> we can't creat object for interface and Abstract class.
=> we can't create object interface but we can create reference variable of Interface type.
=> The class which provide implementation for Abstract method is called as implementation class.
=> If any one of Abstract method is not overidden or is not provide with the body then the class should be declared as abstract.
=> Java provide a keyword called implement to inherit the property from interface to class.
=> Java provide keyword called extends to inherit the property from interface to interface.
=> Each and every class extends object class which the super most class in classtype, but interface doesn't extends any class because Interface itself is the supermost Java type.

Note:- Any variable which was declare with the keyword final is called as final variable. It is used to hold constant or standard value.

=> final variable cannot reinitialize.



Date _/_/_

Q. WAP to demonstrate 100% implementation through interface.

→ abstract interface sample
{ public abstract void UN();
public abstract void pwd();
}

class Demo implements Sample
{ public void UN()
{ S.o.p("hey I am Psycho");
}
public void pwd()
{ S.o.p("hey I am Super Psycho");
}

class main
{ public static void main(String [] args)
{ Demo d = new Demo();
d.UN();
d.pwd();
}
}

Concrete method

o/p
hey I am Psycho
hey I am Super Psycho

Q. WAP to demonstrate 50% implementation through interface.

→ abstract interface Social_media
{ public abstract void UN();
public abstract void pwd();
}

abstract class Meta implements Social_media
{ public void UN()
{ ==========
}
// public abstract void pwd();
}

class facebook extends Meta
{ // public void UN()
// { ==
// }
public void pwd()
{ ==========
}
}

class main
{ public static void main(String[] args)
{ facebook f = new facebook();
f.UN();
f.pwd();
}
}

Date _ / _ / _

Abstract class	interface Class
→ Supports * abstract method * Concrete method * Constructor	→ It supports * Abstract method * Concrete method ✗ * Constructor ✗
→ We can achieve upto 100% abstraction (Data security through Abstract class)	→ We can achieve exact 100% abstraction (Data security through interface).

e.g
abstract class A
{
abstract void key();
void hell()
{ }
A()
{ }
}
(Concrete method)
	e.g
	interface Demo
	{
	public abstract void hey();
	void hell()
	{ }
	void Demo()
	{ }
	}
	→ Concrete method are not possible here

Markey interface ⇒ An empty interface is called as markey interface.

```
interface Sample
{
}
```

Exp— Serializable, Cloneable

Functional interface :- An interface if it is having only one abstract method is called as functional interface.

```
interface Sample
{
public abstract void hell();
}
```

Exp→ Runnable, Comparable, ActionListner

Date _/_/_

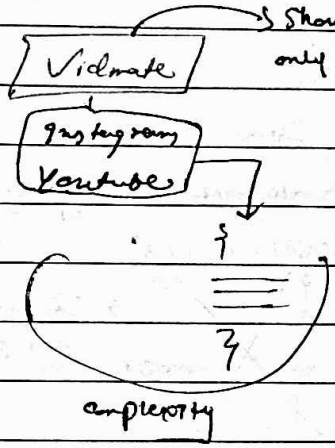
Q. WAP to demonstrate Abstraction.

Q. what do you mean by abstract method overriding?

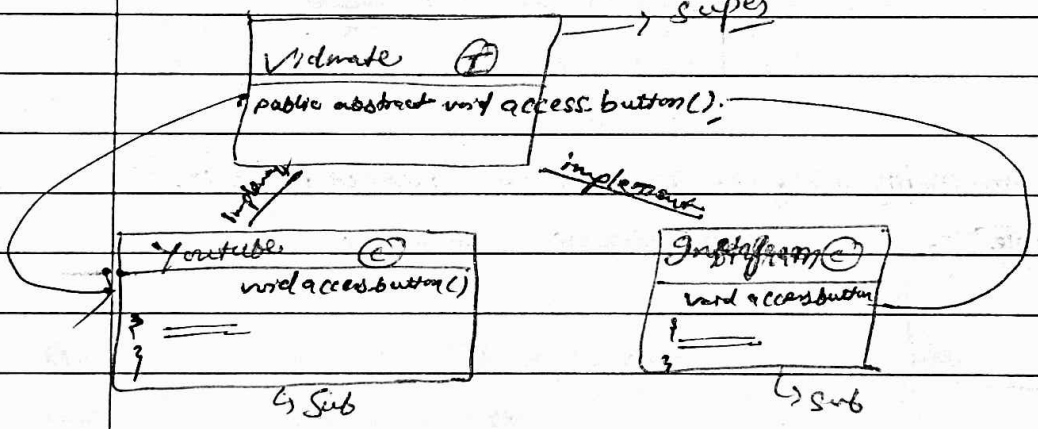
⇒ Converting the abstract methods into concrete method or Complete method is called as abstract method overriding

Hindi

(i) non-static
(ii) hierarchical inheritance
(iii) abstract method overriding
(iv) Pass by reference
(v) upcasting
(vi) interface
(vii) abstract method
(viii) Concrete method.



Aim
→ Single Point of contact
→ Data Security



Date _/_/_

```
abstract interface vidmate
{   public abstract void access_button();
}

Class Youtube implements Vidmate
{   // abstract method overriden
    void access_button()
    {  s.o.p ("hey welcome to youtube");
    }
}

Class Instagram implements Vidmate
{
    // abstract method overriden
    void access_button()
    {  s.o.p ("hey welcome to Instagram");
    }
}

Class Stimulator
{
    Static void spc (vidmate a)
    {
        a.access_button();
    }
}

Class main
{
    public static void main (string[] args)
    {  Youtube Y = new Youtube();
       Instagram I = new Instagram();
           Stimulator.spc (Y);
           Stimulator.spc(I);
    }
}
```

JVM →

spc → single point of contact

upcasting

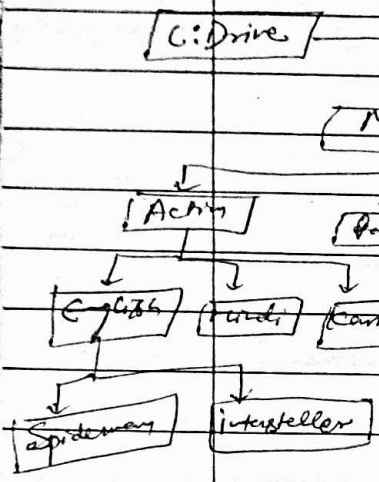
Youtube()

22nd March

Date _/_/_

Packages

⇒ It is a folder like structure which is used to store similar kind of data or files.



Access Specifier ⇒ Def" ⇒ It is used to restrict the access of the member from one class to another class or from one package to another package.

⇒ In java there are 4 access specifier

(i) public
(ii) Private
(iii) protected
(iv) default or package level

(i) **public** ⇒ Any member of the class which is declared with the keyword public is called as public access specifier.

⇒ public access specifier can be accessed

(i) within the class
(ii) It can be access b/w the class
(iii) It can be access within the package.
(iv) It can be access outside the package.

Date _/_/_

(ii) Protected ⇒ Any member of the class which is declared with the keyword protected

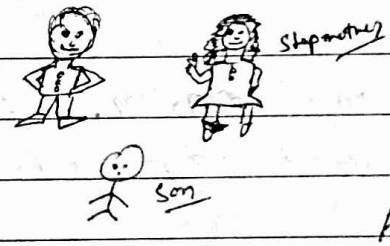
It can be accessed ⇒ within the class
⇒ b/w the classes
⇒ within the package
⇒ outside the package, when there is a relationship. (inheritance)

(iii) default/package level ⇒ Any member of the class which is declared without any keyword (any access specifier) is called as default/package level.

It can be accessed ⇒ within the class
⇒ b/w the class
⇒ It can be accessed only within the package

(iv) private ⇒ Any member of the class which is declared with the keyword private is called as private access specifier.

⇒ It can be accessed only within the class.



public cycle()
private atm()
protected bike()
car()

Package ~~undefined~~ undefined

```
class Father
{
    public void cycle()
    {
        S.o.p("Tring Tring");
    }
    protected void bike()
    {
        s.o.p("Dug Dug Dug");
    }
    void car()
    {
        s.o.p("B———");
    }
    Private void atm()
    {
        S.o.p("Swipe out");
    }
}
```

```
class Son
{
    Public Static void main(String[]args)
    {
        Father f = new Father();
        f.cycle();
        f.bike();
        f.car();
        f.atm(); X
    }
}
```

```
package auntyfamily;
import unclefamily.father;
class Aunty extends father
{/*
public void cycle()
{ S.o.p("my bicycle");
}
Protected void bike()
{ S.o.p("my Benz");
}
*/
}
```

```
public static void main(String[] args)
{ Aunty a = new Aunty();
  a.cycle();
  a.bike();
}
```

Date _/_/_

Encapsulation

⇒ Wrapping up of Datamember (variable) with the function member (method) in order to make single unit is called as encapsulation.

How to achieve encapsulation.

1. Declare Datamember as private.
2. Restrict Direct access of datamember outside the class.
3. Provide indirect access by using public services called getters and setters.
   getters and setters are not mandatory, just they are industrial convention.
4. getters are used to get the values.
5. Setter are used to set the values.

Note:- If we want to access non-static member inside another one, If of static body, I can access that non-static members without creating object.

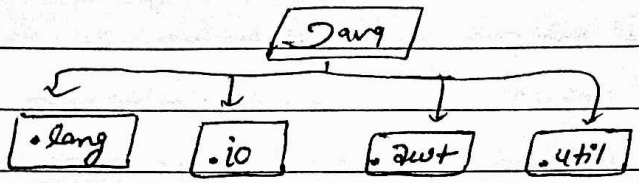
```
class Bank
{ private int atm_pin = 2525;
  public int get_atm()
  { return atm_pin
  }
  public void set_pin(int atm_pin)
  { this.atm_pin = atm_pin;
  }
}
```

```
class Atm_user
{ public static void main(String[] args)
  { Bank b = new Bank();
    // S.o.p(b.atm_pin); → This give error.
    int x = b.get_atm();
    S.o.p(x); //o/p → 2525
    b.set_pin(3636)
  { S.o.p(b.get_atm()); //o/p → 3636
  }
}
```

Date _ / _ / _

Libraries

→ In Java we are having around 4 inbuilt libraries or packages.

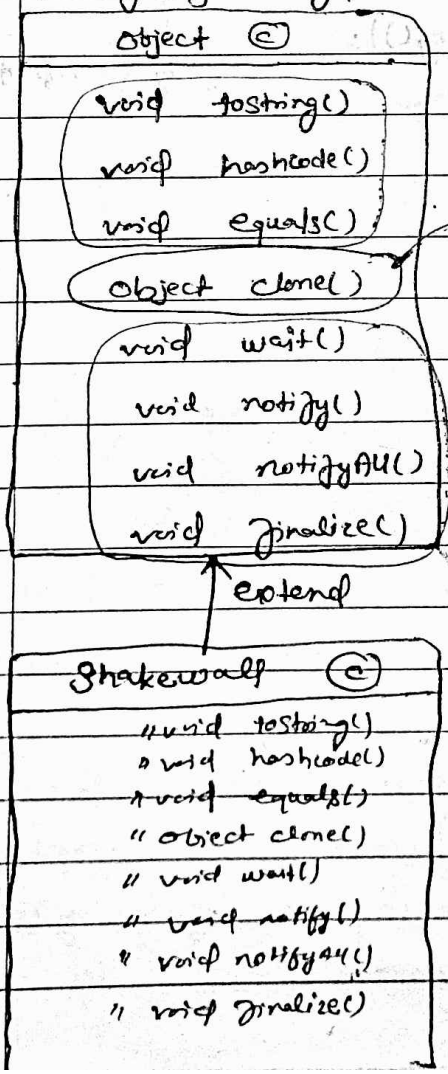


Note:- java.lang is the package it will be imported by default for each and every class.

Object class

1. Object class is the Supermost class in java.
2. Object class presented or belongs to java.lang package.
3. each and every class extends to the Supermost class called object class.

package java.lang;



→ It will not be used (just it is used to clone object)

→ Thread class

tostring() → tostring() method is the non static non final method presented in the object class.

→ It will be inherited to each and every class.

→ It is presented virtually inside each and every class. (Because it is presented inside object class)

→ Whenever we print the reference variable tostring() method get invoked internally implicitly or automatically.

(Package name. classname@ hexadecimal number)

Date _/_/_

→ Signature for tostring method;

public String tostring();

package → java.lang

Class object

```
{
    public void tostring()
    {
    }
}
```

Class Sample extends object

```
{
    public string tostring()
    {
        return "package@HexadecimalFullyQualifiedPath"
    }
    public static void main(String[]args)
    {
        Sample S = new Sample();
        System.out.println(S.tostring());
    }
}
```

O/P
Fully Qualified path

Q. Can we override tostring() method?

⇒ yes

Q. Write a program to override tostring() method;

⇒

```
public class Sample
{
    public string tostring()
    {
        return "I love Java";
    }
    public static void main(String[]args)
    {
        Sample S = new Sample();
        S.o.p(S.tostring());
    }
}
```

Date _/_/_

hashcode()

→ hashcode is a non static non final method of object class, it will be inherited to each and every class.

→ whenever we invoked hashcode method it will generate unique integer number called hashnumber based on the object address.

→ It will generate hashcode based on hashing algorithm, the signature for hashcode is

```
public int hashcode()
```

→ It should be invoked explicitly.

```
Packag→ java.lang;

Class object
{   public void hashcode()
    {
    }
}
Class Tower extends object
{ /* public int hashcode()
    {
        return 127658;
    }
   */
   public static void main(String[]args)
   { Tower t = new Tower();
     int x = t.hashcode();
     S.o.p(x);  ——————→ 127658
   }
}
```

Q. WAP to override the hashcode.

```
class Demo
{
    public int hashcode()
    { return 969669;
    }
    public static void main(String[]args)
    { Demo d = new Demo();
      S.o.p(d.hashcode());
    }
}
```

equals()

→ equals is a non-static non final method of object class. It will be inherited to each and every classes.

→ equals() method is used to compare the object address.

→ equals() method invoked explicitly.

→ Signature for equals() method is

```
public boolean equals()
```

Date _ / _ / _

```
public class Water
{ public static void main(String[]args)
  {   Water w = new Water();
      S.O.P(w);
      Water w1 = w;
      S.O.P(w1);
      boolean x = w.equals(w1);
      S.O.P(x);
  }}
```

Q. Can we override equals() method ?

⇒ Yes

```
public class Water
{
  public boolean equals(Object obj)
  { return true;
  }
  public static void main(String[] args)
  {
    Water w = new Water();          //O/P → True
    S.O.P(w);
    Water w1 = new Water();
    S.O.P(w1);
    boolean x = w.equals(w1);
    S.O.P(x);
  }
}
```

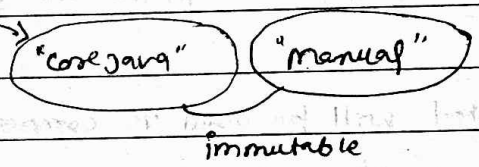
Date _/_/_

String class

(1) String is a final class it is presented inside java.lang package.
(2) Since String is final it can't be inherited
(3) String is immutable.

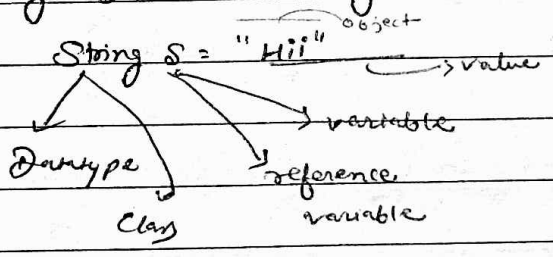
Q. Why String is called immutable?
A) Because it will not change the state and value of an older object for that reason string is immutable.

```
String s = "core java";
s = "manual";
S.o.p(s);
```
o/p → manual

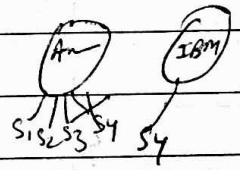


Q) In how many way we can create object for string class?
→ We can create object for String class in two ways:-
(i) with new keyword (ii) without using new keyword.

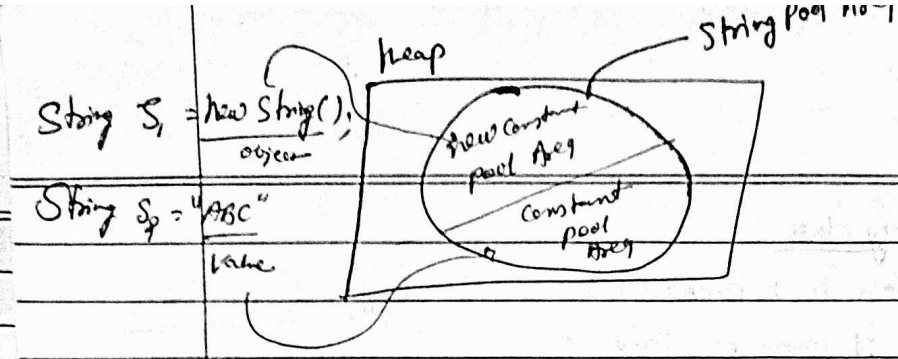
(i) String s = new String()



String s1 = "Amazon"
String s2 = "Amazon"
String s3 = "Amazon"
String s4 = "Amazon"
s = "IBM"



Q. Why String is immutable justify with real time example?
A) When multiple reference variable pointing towards the single object and among those reference variable is anyone reference variable wants dereference it should not affects the other reference variable.
In String class dereferenced referenced variable did not affect the remaining reference variable so that I can say string is immutable.



Date _/_/_

⇒ If I create object for String class without using new keyword, it will get stored inside Constant pool Area inside the String Pool Area.

⇒ If I create object for string class with new keyword, it will get stored inside non constant pool Area inside the String Pool Area.

⇒ Equals() method will be used to compare object Address in object class.
But in String class equals method will be used to compare values.

```
class Demo
{   public static void main(String[] args)
    {   String S1 = "hii";
        String S2 = "hii";
        boolean x = S1.equals(S2);
        System.out.println(x);
    }
}
```

Equals()	Comparision operator
→ It is used to compare the value inside string class.	→ It is used to compare the object address inside String class.

```
String S1 = new String();
String S2 = new String();
System.out.println(S1==S2); // false
```

Date _/_/_

→ final class ⇒ Any class which is declared with the keyword final is called as final class.

Example ⇒ final String class
{ ——— }

→ Non final class ⇒ Any class which is declared without the keyword final is called as non-final class.

Example ⇒ class Object
{ ——— }

→ final class can not be inherited. class sample extends String
{ }

⇒ final methods can be overloaded.

```
class RCB
{
  final static void play(String Y)
  { S.o.p(y); }
  final static void play(int X)
  { S.o.p(x); }
JVM → public static void main(String[] args)
  { play("C sala cup namdy");
    play(143); }
}
```

O/P
C sala cup namdy
143

⇒ final method can not be overridden.

```
class A
{ final void add()
  { }
}
class B extends A
{ void add()
  { }
}
```

⇒ Constructor can not be final.

```
class B
{ final static void B() { }
}
```

Date _/_/_

String inbuilt method are String inbuilt function.

S.length()

Q. WAP to demonstrate or find the length of the given string.

=>

```
class Sample
{   static void operate()
    {   String S = "Java";
        0 1 2 3
        SOP(S.length());
    }

JVM → public static void main(String[]args)
    {   operate();
    }
}
```

S.toLowerCase()

Q. WAP to convert upper case into lower case in given String.

```
class Sample
{   static void operate()
    {String S = "JAVA";
     System.out.println(S.toLowerCase());
    }
    public static void main(String[] args)
    {   operate();
    }
}
```

S.toUpperCase()

```
class Sample
{   static void operate()
    {   String S = "java";
        System.out.println(S.toUpperCase());
    }
    public static void main(String[] args)
    {   operate();
    }
}
```

Date _ / _ / _

S.indexOf('a');

Q. WAP to print the index of a in "Java" ?

```
class sample
{   static void operate ( )
    {   String s="Java";
        System.out.println ( S.indexOf('a'));
    }
    public static void main (String []args)
    {   operate();
    }
}
```

S. lastIndexOf ('a')

Q. WAP to demonstrate last index of in given String ?

⇒

```
class Sample
  {   static void operate ( )
      {   String s = "Java"
          System.out.println ( S. lastIndexOf ('a'));
      }
      public static void main (String []args)
      {   operate();
      }
  }
```

S. charAt (i)

Q. WAP to demonstrate charAt(index) / WAP to fetch each and every character of given Str

⇒

```
class Sample
  {   static void operate ( )
      {   String S = "Java";
          for (int i=0; i<S.length(); i++)
          {   System.out.println (S.CharAt(i));
          }
      }
      public static void main (String [] args)
      {   operate();
      }
  }
```

Date _ / _ / _

S.StartsWith(" ")

Q. WAP to check whether the given string is StartsWith specified sequence.

```
class sample
{
    Static void operate()
    { String S = "Javamava";
      System.out.println(S.StartsWith("Java"));   // true
    }

    public static void main(String[] args)
    { operate();
    }
}
```

S.endsWith(" ")

Q. WAP to check whether the given string is endsWith specified sequence.

```
class Sample
{   Static void operate()
    { String S = "Javamava";
      System.out.println(S.endsWith("ava"));
    }
    Public Static void main(String[] args)
    { operate();
    }
}
```

Q. WAP to concat a String with another String

S.Concat(" ")

```
class Sample
{   static void operate()
    { String S = "Java";
      System.out.println(S.Concat("mava"));
    }
    Public Static void main(String[] args)
    {
      operate();
    }
}
```

Date _/_/_

S.equals(S2)

Q. WAP to check whether the given Strings are equal.

```
class Sample
{   static void operate()
    {   String S1 = "mom";
        String S2 = "mom";
        System.out.println(S1.equals(S2));
    }
    public static void main(String[]args)
    {   operate();
    }
}
```

Q. WAP to demonstrate a method called substring.

S.substring("")

S.substring(indx1, indx2)

```
class Sample
{   static void operate()
    {   String S="methodoverloading"
        System.out.println(S.substring(10,14)); // Load
    }
    public static void main(String[]args)
    {   operate();
    }
}
```

Q. WAP to demonstrate replace method.

or) WAP to replace Specified character with another character in the given String.

S1.replace(old char, new char)

```
class Sample
{   static void operate()
    {   String S1= "javamava";
        System.out.println(S1.replace('a','o'));
    }
    public static void main(String[] args)
    {   operate();
    }
}
```

O/P javomovo

Date _ / _ / _

~~Exception Handling~~

~~Convert string to other~~

toCharArray :-

→ Convert this given String into character array.

```
String S = "Java"
           0 1 2 3

char[] arr = S.toCharArray( );
for (int i=0; i<=arr.length-1 ; i++)
{
    S.o.p(arr[i]);
}
```

O/P
J
a
v
a

// Reverse a String

```
class Sample
{
    public static rev( String S)
    {
        String str = "";
        for (int i = S.length()-1 ; i>=0 ; i--)
        {
            str += S.charAt(i);
        }
        System.out.println (str);
    }
    public static void main (String[] args)
    {
        rev("Java");
    }
}
```

Date _/_/_

Exception Handling

=> It is an event triggered during the execution of program, which stops or interrupts the execution suddenly or abruptly.

Handling this event by using try and catch or throws is called as Exception Handling.

=> All the exception (Abnormal statement) should be developed inside the try block and handle or address them inside catch block.

=> We cannot develop any statement between try and catch block.

=> For one try block there should be minimum one catch block.

=> We cannot develop try block alone, It will throw Compile Time Error.

=> Exception can be handle in two ways.

(i) try and catch

(ii) throws

=> For one try block we can have more than one catch block.

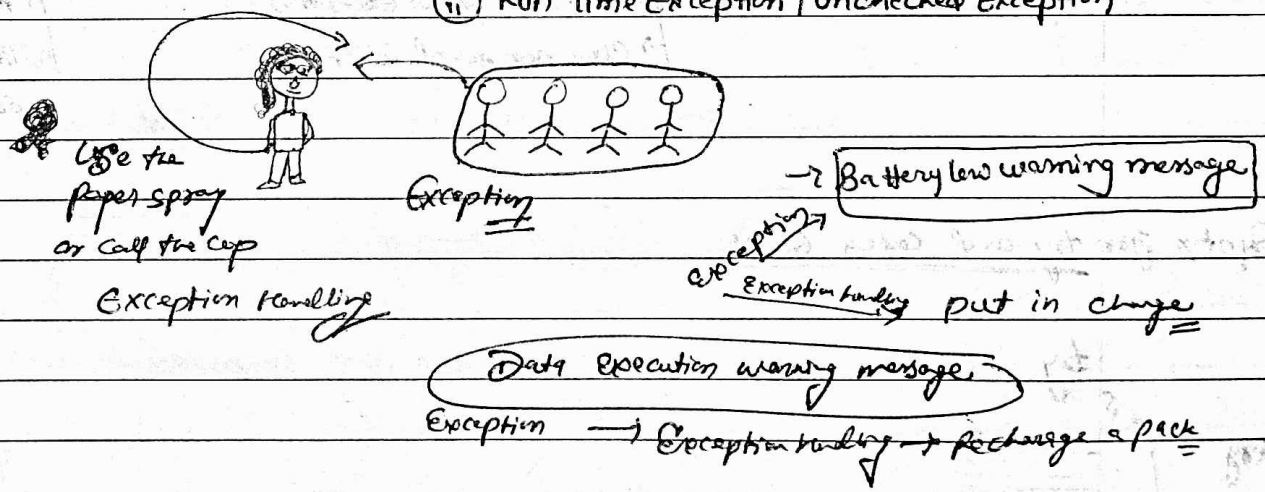
=> All the exception can be handled or addressed by using exception class.

=> All the exception class belong to java.lang package.

=> Exception are of two types (i) Compile Time Exception / Checked Exception

(ii) User defined Exception / Customized exception.

(ii) Run Time Exception / Unchecked Exception



Compile Time Exception

=> Any exception which is occur at compile Time is called as Compile Time Exception.

=> Compile Time Exception is handled by try and catch or throws.

=> Compile Time Exception should be addressed at compile time.

Run Time Exception :-

Date _ / _ / _

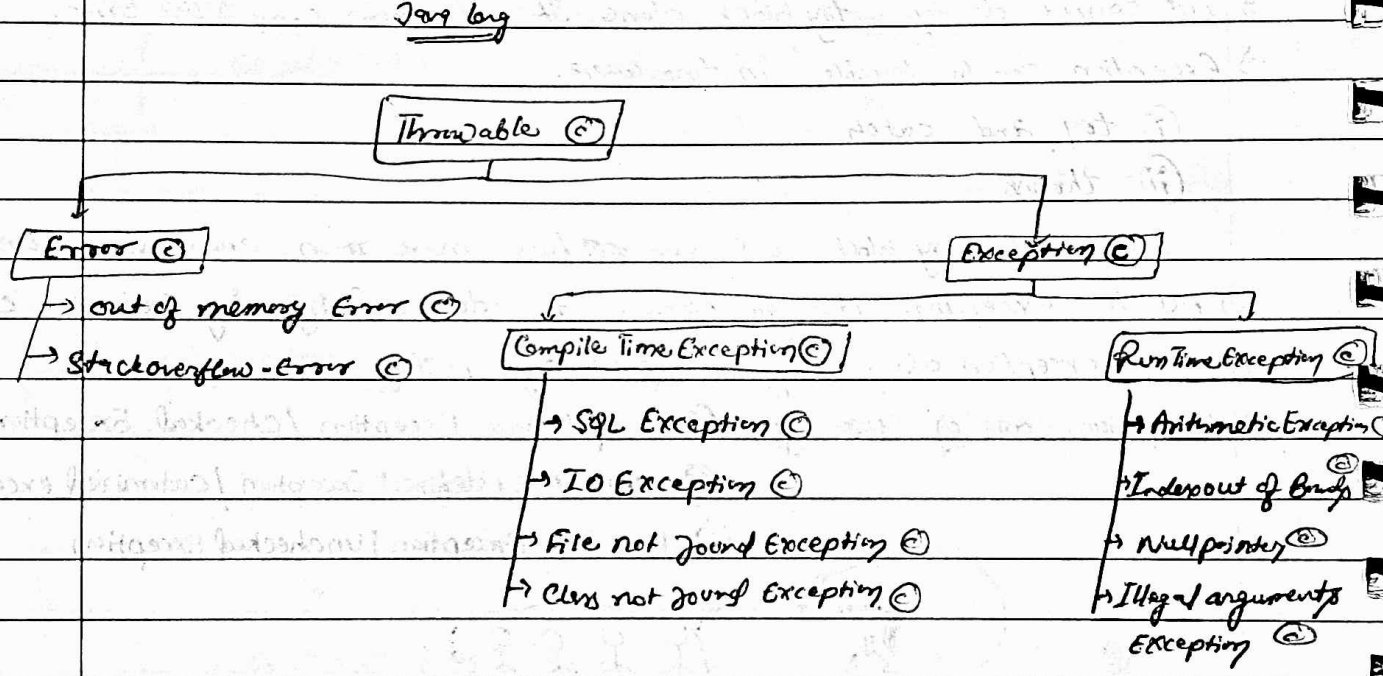
=> Any Exception which is occurred at runtime is called as runtime Exception.

=> Run Time Exception can be handled try and catch or throws.

=> Run Time Exception should be addressed at runtime itself.

=> once the exception is addressed or handled it can't be rehandled or readdressable.

Exception Hierarchy



Syntax for try and catch block

```
try
{
    ==========
}
Catch (Exception classname referencevariable)
{
    ==========
}
```

try block

Date _ / _ / _

Syntax:-

try-catch-catch

```
try
{
}
catch(Exceptionclassname reference variable)
{
}
catch (Exception classname reference variable)
{
}
```

Nested-try-catch

```
try
{
   try
   {
   }
   catch (Exception classname reference variable)
   {
   }
}
catch(Exceptionclassname reference variable)
{
}
```

Date _ / _ / _

Syntax

try - finally

```
try
{
    ____
    ====
}
finally
{
    ____
    ====
}
```

try - catch - finally

```
try
{
    ____
    ====
}
catch(Exceptionclassname reference vari
{
    ____
    ====
}
finally
{
    ____
}
```

Q. WAP to demonstrate Arithmetic Exception.

→

```
class Sample
{
    public static void main (String [] args)
    {
        S.O.P("Main starts");

        try
        {
            int x = 1/0 ;  → Abnormal statement or Arithmetic Exception
        }
        catch (Arithmetic Exception e)   ← new Arithmetic Exception();
        {
            S.O.P ("Exception has been handled");
        }
        S.O.P("Main ends");
    }
}
```

Date _ / _ / _

StringIndexOutOfBoundsException

Q. WAP to demonstrate ~~ArithmeticException~~ Exception.

```
class Sample
{
    public static void main(String[] args)
    {   S.o.p("main starts");
        String s = "Java";
        try{
            S.o.p(s.charAt(33));
        }
        catch(StringIndexOutOfBoundsException e)
        {   S.o.p("Exception handled Sussessfully");
        }
        S.o.p("~~Exception~~ main ends");
    }
}
```

Abnormal Statement or StringIndexOutOfBoundException → Internally thrown or create → new StringIndexOutOfBoundsException

Q. WAP to demonstrate ArrayIndexOutOfBoundsException.

⇒

```
class Sample
{   static void operate()
    {   int[] arr = {9, 99, 999};
        try{
            System.out.println(arr[10]);
        }
        catch(ArrayIndexOutOfBoundException e)
        {   System.out.println("Exception handled");
        }
    }
    public static void main(String[] args)
    {   operate();
    }
}
```

new ArrayIndexOutOfBoundsException()

JVM

Date _/_/_

Q WAP to demonstrate NullPointerException.

```
class Sample
{
static void operate()
{
   String S = Null;
System.out.println("Main Start");
try{
    System.out.println(S.toString());
   }
Catch(NullPointerException e){
      System.out.println("Exception handled");
  }
}
public static void main(String[]args)
{
    operate();
}
}
```

Abnormal or NullPointerException

(#) once exception is handled or addressed it cannot be rehandled or readdressed.

```
class Sample
{ public static void operate
  { String S = null;
    try
    { System.out.println(S.toString());
    }
    catch(NullPointerException e)
    { System.out.println("Exception Handled");
    }
    Catch(NullpointerException e)
    { System.out.println("Exception Handled");
    }
}
Public & static void main(String[]args)
{operate();
}
```

=> It gives CTE

Date _/_/_

(#) we can't develop any statement between try & catch:

```
class Sample
{ public static void operate()
  { String s=null;
    S.o.p("main starts");
    try{
       System.out.println(S.toString());
    }
    int x=100;  ——————> gt cause CTE
    Catch(NullPointerException e){
       System.out.println("Exception handled");
    }
  }
  public static void main(String[]args)
  { operate();
  }
}
```

(#) try and finally Block

↳ finally block is a block in exception handling, it will get executed, even the exception is handled or not.

→ WAP to demonstrate try, finally

```
class Sample
{ static void operate()
  { String s=null;
    S.o.p("main starts")
    try{
       S.o.p(s.toString());
    }
    finally{
       S.o.p("data base error message");
    }
  }
  public static void main(String[] args)
  {
     operate();
  }
}
```

o/p

```
main starts
database Error
Exceptin ——————
```

Date _/_/_

WAP to demonstrate try-catch-finally

=>

```
try{
    System.out.println(s.toString());
}
catch(NullPointerException e)
{
    System.out.println("Exception Handled");
}
finally{
    System.out.println("data base closed Handled");
}
```

4/4/2024

```
class Shadidotcom
{
    static void submit() throws MarriageException
    { int a=20;
      if (a>=21) //false
      { System.out.println("Happy married life");
      }
      else{
          throw new MarriageException("Invalid age");
      }
    }
    public static void main(String[] args)
    { try{
          submit();
      }
      catch(MarriageException e)
      {
          S.O.P(e.getMsg())
      }
    }
```

JVM

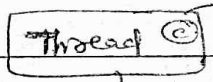
Date _/_/_

```
class MessageException extends Exception
{   String msg;
    MessageException(String msg)
    {
      this.msg=msg;
    }
    public String getmsg()
      { return msg;
    }   }
```

Q. What is the difference between throw and throws.

=> throw is used to create an exception	throws is used to propogate the exception.
=> throw should be develop within method body.	throws should be developed with method declaration
=> throw can throw only one object at a time.	throws can propogate more than one exception.

Interrupted Exception



sleep(); → Sleep is a method belong to Thread class.

```
public class Sample
  {
    public static void main(String[] args)
      { for(int i=1 ; i<=10; i++){
          System.out.println(i);
        try {
          Thread.sleep(2000);
        }
        catch(InterruptedException e){
          System.out.println("Hello");
        }
      }
  }
```

Date _/_/_

Stack unwinding

=> If any Exception in any of the method it will get propogated to main method and stack will get destroyed, hence it is called Stackunwinding.

```
public class Sample{
    static void disp1(){
        int i=1/0;
    }
    static void disp2(){
        disp1();
    }
    static void disp3(){
        disp2();
    }
    static void disp4(){
        disp3();
    }
    public static void main(String[]args)
    {
        try{
            disp4();
        }
        Catch(ArithmeticException e)
        {
            e.printStackTrace();
        }
    }
```

printStackTrace() => It is a non static method of throwable class, which is inherited each and every class. It will print the complete backtrace of the exception which is storing in the system.

Error	Exception
=> Error is unpredictable.	=> Exception is predictable
=> Error is occured due to System configuration	=> Exception is occured due to mistake done by Programmer
=> Error can be handled	=> exception can also be handled.

Date _ / _ / _

Constructor Overloading

→ Developing a multiple constructor within the class but variations in the argument list is called as constructor overloading

Variations

(i) variations in no of argument
(ii) variations in data type of argument
(iii) variation in order of occurance

this() Statement ⇒ this() calling statement is used in case of constructor overloading.

⇒ this() calling statement is used to call from one constructor to another constructor within the class.

Rules

⇒ this() calling statement should be the first statement in the constructor.

⇒ we can use multiple this() keyword.

```
Class Sample
{   Sample (String a, int b)
    {
        S.O.P(a+" "+b);
    }
    Sample (int c, String a)
    {
        this("hello", 55);
        S.O.P(c+" "+a);
    }
    Sample (int a, int b)
    {
        this(50, "java");
        S.O.P(a+" "+b);
    }
}
Class MainClass
{
    psvm ( )
    {
        new Sample(10,20);
    }
}
```

JVM

o/p

hello 55
50 java
10 20

⇒ A subclass constructor calling its immediate superclass constructor and that superclass constructor calling its immediate superclass constructor is called constructor chaining.

Super() Statement

A super calling statement is used in the case of inheritance to call from subclass constructors into its immediate superclass constructor.

Rules :- (i) It should be the first statement.

(ii) We cannot develop multiple super() calling statement inside a constructor.

(iii) It is used in the case of inheritance.

```
class choti{
    choti(int a, int c){
        System.out.println(a+" "+c);
    }
}
class moti extends choti{
    moti(string a, int b)
    {
        super(10,20);
        System.out.println(a+" "+b);
    }
}
class sample extends moti
{
    sample(int a, double b)
    {
        super("Rohit",10);
        System.out.println(a+" "+b);
    }
}
class mainclass
{
    public static void main(String[] args)
    {
        new sample(2, 2.5);
    }
}
```

O/p 10 20
Rohit 10
2 2.5

Date _ / _ / _

Multiple Inheritance

→ (i) Diamond problem

(ii) Constructor chaining

- only one supercalling
- only one this calling

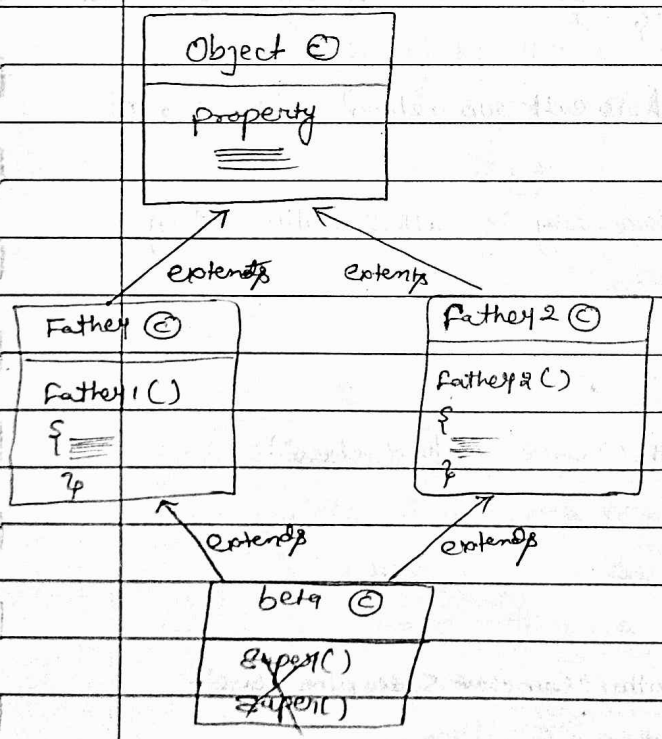
Diamond problem ⇒ The diamond problem occurs when a class inherits from two classes that both inherit from a common superclass. This creates ambiguity about which superclass's properties or method should be inherited by the class.

Constructor chaining ⇒ Constructor chaining refers to the process where one constructor calls another constructor in the same class or superclass.

Ex- (1) Class A has a constructor

(2) Class B and class C each have constructors that call class A's Constructor

(3) Class D would need to call constructors of both class B and class C, which both try to call class A's constructor.



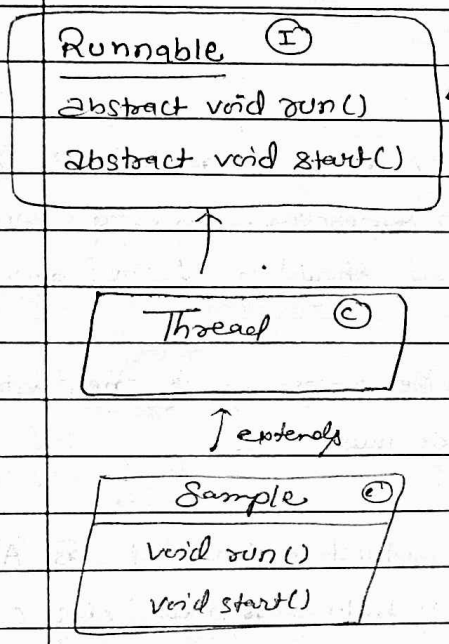

```
interface johny_bhai
{
    void doctor();
}

interface Rocky_bhai
{
    void golddigger()
}

Public class chutki implements johny_bhai, Rocky_bhai
{
    void heroine()
    {
        System.out.println("I'm your heroine");
    }
    void doctor()
    {
        System.out.println("hey I'm your doctor");
    }
    void golddigger()
    {
        system.out.println("hey I'm your lover");
    }
}

Public class main()
{
    public static void main(String[] args)
    {
        chutki c= new chutki();
        c.heroine(); c.doctor(); c.golddigger();
    }
}
```

Thread

Date _ / _ / _

=> Thread is a class which belong to java.lang package which will be imported each & every package.

=> Thread is instance execution which owns its own cpu time & memory.



Q. WAP to implements run method?

```
class Chamby extends Thread{
        public void run(){
            System.out.println("Come to class");
        }
}
Public class Sample
{   public static void main (String[] args)
    {   Chamby C = new Chamby();
        Thread th1 = new Thread(c);
        th1.run();      // come to class.
    }
}
```

start() => start() is a non-static method used to call run method or invoke run method.

multithreading => processing of multithread simultaneously is called multithreading.

Example

```
class Chamby extends Thread {
        public void run {
            System.out.println("come to chambyclass");
        }
}
class Sudeeptha extends Thread$
        public void run{
            System.out.println("Come to Sudeeptha class");
        }
class mainclass
{   Public static void main (String[] args)
    {   Chamby C = new Chamby();
        Thread th1 = new Thread(c);
        th1.run();
```

```
        Sudeeptha s = new Sudeeptha();
        Thread th2 = new Thread(s);
        th2.start();
    }
}
```

Date _ / _ / _

Q. In how many ways we can override run method :-

(i) By extending Thread class

(ii) By implements runnable interface

Multithreading (Asynchronous function)

```
class chomby extends Thread
{
    public void run()
    {
        for(int i=0; i<=10; i++)
        {
            System.out.println(i);
            try{
                Thread.sleep(2000);
            }
            catch (InterruptedException e)
            {
            }
        }
    }
}

class Sudeeptha extends Thread
{
    public void run()
    {
        for(int i=100; i<=110; i++)
        {
            System.out.println(i);
            try {
                Thread.sleep(2000);
            }
            catch (InterruptedException e)
            {
            }
        }
    }
}

class Mainclass
{
    public static void main (String [] args)
    {
        chomby c = new chomby();
        Thread th1 = new Thread(c);
        th1.start();
        Sudeeptha s = new Sudeeptha();
        Thread th2 = new Thread(s);
        th2.start();
    }
}
```

Arrays in Java

Date _ / _ / _

Array is linear Data structure which is used to store homogenous types of data.

Why Array?

→ To overcome the drawback of variable.

eg→ | 1 | 20 | 21 | 50 |

Drawbacks of Array :-

(i) Array Size is fixed.

(ii) we can store only homogenous type of data.

(1) Array Declaration :-

Datatype [] arrayname;

e.g. int [] arr; | 10 | 22 | 33 | 66 | 99 |

(2) Array size initialization :-

arrayname = new int[size];

eg. arr = new int[5];

(3) Store the elements inside array :-

arrayname[index] = value;

arr[0] = 10;

arr[1] = 15;

(4) Array Utilization :-

System.out.println(arrayname[index]);

e.g- S.O.P(arr[0]); // 10

S.O.P(arr[1]); // 15

(5) Array length calculation :-

System.out.println(arr.length);

(6) Array declaration and initialization in a single line

Datatype [] arrayname = {v1, v2, v3, v4};

int [] arr = {1, 2, 3, 4};

Date _ / _ / _

(7) Array re-initialization :-

arrayname[index] = new value;

e.g arr[3] = 89;

(8) Copying the value from one array to another array :

```
int[] arr = {99, 69, 63, 21};
int[] abb = arr;
```

(9) Looping through an array

```
// Using a for loop
for(int i=0; i< arr.length; i++)
{
  System.out.println(arr[i]);
}

// Using a for-each loop
for(int element : arr)
{
  System.out.println(element);
}
```

(10) Sorting an Arrays;

```
import java.util.Arrays;
int[]arr = {5, 3, 8, 1, 2};
Arrays.sort(arr);
```

(11) Binary Search;

```
int index = Arrays.binarySearch(arr, 3);
```

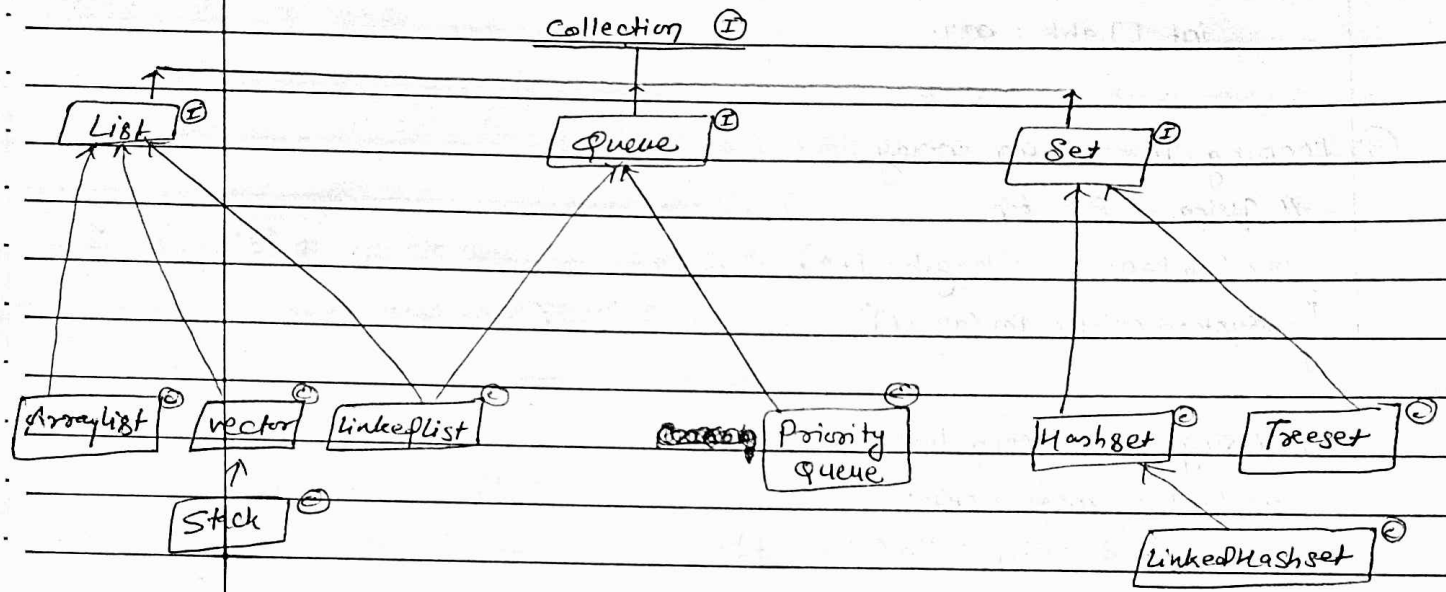
(12) Filling an Array :

```
int[] arr = new int[5];
Arrays.fill(arr, 10);
```

Collection → Java.util package

Date _ / _ / _

→ Collection is a unified architecture.
→ Collection consists of both interface and class.
→ Collection will store both homogeneous and heterogeneous type of data.
→ In order to overcome the drawback of array we go for collections.
→ collection belong to java.util package



* In java, when it comes to string, array, collection index always starts from zero.

	Array	Collection
→	Size is fixed	→ Size is dynamic
→	Stored only homogenous	→ Stored both homogenous and heterogenous
		→ The default capacity is 10.
		→ It increases its size by 50%.

→ In java object is the only class type which is having capability to store heterogenous type of data.

In collection we have 3 sub interfaces :-

(#) List
(#) Queue
(#) Set

Date _ / _ / _

List

List is an interface whenever we want to store the element upon index and allow duplicates then we should go for List type of collection.

Features of List →

(i) Size is dynamic
(ii) we can store heterogenous type of data.
(iii) It is indexed type of collection
(iv) It allows duplicates
(v) It follows order of insertion
(vi) It allows null
(vii) Since, it is indexed type of collection we can fetch the elements upon index randomly.
(viii) It is not auto sorted.

List has three subclasses.

(i) ArrayList
(ii) Vector
(iii) LinkedList

(i) ArrayList

(i) Size is dynamic
(ii) we can store heterogenous type of data
(iii) It is indexed type of collection.
(iv) It allows duplicates
(v) It follows order of insertion
(vi) It allows null.
(vii) Since, it is indexed type of collection we can fetch the elements upon index randomly.
(viii) It is not auto sorted
(ix) It increases its size by 50%
(x) It is not synchronized.
(xi) Since its not synchronized the performance is fast.

Date _/_/_

add() :> add is non static method of collection interface, it is used to add the element into collection or list. here elements will get appended at the end of the list.

Q. WAP to demonstrate add() method ?

" Inside reference variable we can store object, null and user defined data also "

```
import java.util.Arraylist;
public class Sample
{
    public static void main (string[] args)
    {
        Arraylist l1 = new Arraylist();
        l1.add ("Java");
        l1.add ("SQL");
        l1.add (99);
        l1.add (4.3);
        l1.add (true);
        l1.add (null);
    }
}
```

O/P

[Java, SQL, 99, 4.3, true, null]

Q. WAP to demonstrate add (int index, object obj) method ?

=> If we want to add the element to the specified index in collection/list we will use a method called add(int index, object obj)

```
import java.util.Arraylist;
public class Sample
{
    public static void main (string[] args)
    {
        Arraylist l1 = new Arraylist();
        l1.add ("Java");
        l1.add ("SQL");
        l1.add (1, "J-spider");
        System.out.println(l1);
    }
}
```

O/P

[Java, J-spider, SQL]

Date _ / _ / _

Q. WAP to display addAll() method?

⇒ addAll() is a method which is used to add one collection in another collection.

```
import java.util.Arraylist;
public class sample
{ public static void main (String[] args)
  { Arraylist l1 = new Arraylist();
    l1.add ("java");
    l1.add ("SQL");
    l1.add (99);
  Arraylist l2 = new Arraylist();
    l2.add (100);
    l2.add (water);
    l2.add ("old_monk")
    l1.addAll (l2);
  System.out.println (l1);
  System.out.println (l2);
} }
```

O/P

[Java, SQL, 99, 100, water, old_monk]

[100, water, old_monk]

Q. WAP to demonstrate addAll (int index, collection c);

⇒

```
import java.util.Arraylist;
public class sample
{ Arraylist l = new Arraylist();
    l.add ("Java");
    l.add ("SQL");
    l.add (99);
  Arraylist l2 = new Arraylist();
    l2.add (100);
    l2.add ("water");
    l2.add ("old_monk");
    l2.addAll (1, l2);
  System.out.println (l2);
}
```

O/P

[Java, 100, water, old_monk, SQL, 99]

⇒ If we want to add one collection with another collection based on specified index. we will go for addAll (int index, collection c);

Date _ / _ / _

Q. WAP to demonstrate remove() method inside collection.

⇒ remove() is a method which is used to remove the element from the particular Collection / List.

⇒ remove (object obj)

⇒ remove (int index)

// Demonstrate remove (object obj)

```
import java.util.ArrayList;
public class sample
{ public static void main (String [] args)
  { ArrayList l1 = new ArrayList ();
    l1.add (100);
    l1.add ('water');
    l1. add ("old.monk");
    l1. remove ("water");
    System.out.println (l1);
  }
}
```

o/p

[100, old_monk]

// Demonstrate remove (int index)

```
import java.util.ArrayList;
public class sample
{ public static void main (String [] args)
  { ArrayList l1 = new ArrayList ();
    l1.add (100);
    l1.add ('water');
    l1.add ("old_monk");
    l1.remove (1);
    System.out.println (l1);
  }
}
```

o/p

[100, old_monk]

Q. WAP to demonstrate the size() method in collection or List

⇒ size() is a method which is used to get the size of the collection or list

```
import java.util.ArrayList;
public class sample
{ public static void main (String [] args)
  { ArrayList l1 = new ArrayList ();
    l1.add (100);
    l1.add ('water');
    l1. add ("old.monk");
    System.out.println (l1.size());
  }
}
```

o/p ⇒ 3

Q. WAP to demonstrate the get() method.

⇒ get() is the method which is used to get the element based on the index.

```
import java.util.ArrayList;
public class sample
{ public static void main (String [] args)
  { ArrayList l1 = new ArrayList ();
    l1.add (100);
    l1.add ("water");
    l1.add ("old_monk");
    for (int i=0 ; i< l1.size(); i++)
    { System.out.println (l1.get(i));
    }
  }
}
```

Q. WAP to demonstrate removeAll() ?

⇒ removeAll() is a method used to remove the duplicate of first collection after comparing with the next/specified collection

```
import java.util.ArrayList;
public class Sample
{ public static void main(String[] args)
  { ArrayList l1 = new ArrayList();
    l1.add("Vishwanath_Sir");
    l1.add("Niharika_mam");
    l1.add("SQL");
    ArrayList l2 = new ArrayList();
    l2.add("vishwanath_sir");
    l2.add("Niharika_mam");
    l2.add("Java");
    l1.removeAll(l2);
    System.out.println(l1);
    System.out.println(l2);
  }
}
```

o/p

[SQL]

[vishwanath_sir, Niharika_mam, Java]

Q. WAP to demonstrate retainAll() ?

⇒ retainAll() is a method used to retain the duplicate element of first collection after comparing with the next collection.

```
import java.util.ArrayList;
public class Sample
{ public static void main(String[] args)
  { ArrayList l1 = new ArrayList();
    l1.add("vishwanath_sir");
    l1.add("Skr_sir");
    l1.add("sachin_sir");
    ArrayList l2 = new ArrayList();
    l2.add("Skr_sir");
    l2.add("sachin_sir");
    l2.add("Govind_sir");
    l1.retainAll(l2);
    System.out.println(l1);
    System.out.println(l2);
  }
}
```

o/p [Skr_sir, sachin_sir]

[Skr_sir, sachin_sir, Govind_sir]

Contains() ⇒ contains() is the method which is used to checked the given element is presented inside collection or not.

```
import java.util.ArrayList;
class Sample
{ public static void main(String[] args)
  { ArrayList l1 = new ArrayList();
    l1.add("vishwanath_sir");
    l1.add("Niharika_mam");
    l1.add("SQL");
    if(l1.contains("SQL"))
      S.O.P("It is present");
    else
      S.O.P("It is not present");
  }
}
```

o/p

It is present

Date _/_/_

2. Vector :-

(i) Size is dynamic.
(ii) We can store heterogenous type of data
(iii) It is indexed type of collection
(iv) It allows duplicates
(v) It follows order of insertion
(vi) It allows null
(vii) Since it is indexed type of collection we can fetch the elements upon randomly.
(viii) It is not auto sorted.
(ix) It increses its size by 100%
(x) It is synchronized.
(xi) Since its synchronized the performance is slow.

Q. WAP to demonstrate vector

```
import java.util.vector;
public class sample
{
    Public Static void main (String[] args)
    {
        vector v1 = new vector();
        v1.add ("coke");
        v1.add (1.23f);
        v1.add (100);
        for(int i=1 ; i< v1.size(); i++)
        { System.out.println(v1.get(i));
        }
    }
}
```

Date _ / _ / _

3. LinkedList

⇒ It is a class which has dual property.

⇒ LinkedList inherits the properties from List and Queue interfaces.

⇒ So, we can use get(int index).

Along with that we can use 2 other methods of Queue interfaces

i.e (i) peek() → It will retrieve/fetch the topmost element of the list but doesnot remove it

(ii) pull() → It will retrieve/fetch the topmost element of the list removes it thereby the size gets decreased.

Whenever we need proper order of insertion then we should go for LinkedList, it has dual properties.

i.e, it inherits from both List and Queue.

Features of LinkedList :→

(i) Size is dynamic
(ii) We can store heterogenous type of data
(iii) It is indexed type of collection.
(iv) It allows duplicates
(v) It follows order of insertion
(vi) It allows null
(vii) Since, it is indexed type of collection we can fetch the elements upon index randomly.
(viii) It is not auto sorted.
(ix) It increases its size by 50%

Q. In collection which property will show dual property.

⇒ LinkedList (Reason → It inherits the properties from List interface and Queue interface).

Here, we can see multiple inheritance.

Date _/_/_

```
import java.util
public class Sample {
    public Static void main(String[] args)
    {   LinkedList l1 = new LinkedList();
        l1.add('2');
        l1.add('Java');
        l1.add(null);
        l1.add(1);
        l1.add('2');
        System.out.println("Before peek method:" + l1);
        System.out.println("peek element:" + l1.peek());
        System.out.println("After peek element:" + l1);
        System.out.println("poll element:" + l1.poll());
        System.out.println("After poll method:" + l1);
    }
}
```

Before peek method : [2, Java, null, 1, 2]
peek element : 2
After peek method : [2, Java, null, 1, 2]
poll element : 2
After poll method : [Java, null, 1, 2]

Q. WAP to demonstrate Priority Queue?

```
=> import java.util;
public class Sample {
    public static void main(String[] args)
    {   priorityqueue p= new priorityqueue();
        p.add(7);
        p.add(3);
        p.add(10);
        p.add(9);
        p.add(9);
        System.out.println(p);
    } System.out.println(p.peek());
}
```

o/p [3, 7, 10, 9, 9]

[7, 10, 9, 9]

Date _ / _ / _

Queue :- Queue is an interface whenever we want follow general Queue then we should go for queue type of collection.

In queue we have

(i) Linked list

(ii) priority queue ⇒ It is a class which implements Queue interface which belongs to java.util package.

Features of priority Queue :-

(i) Size is dynamic

(ii) we cannot store heterogenous type of data.

(iii) It is not indexed type of collection.

(iv) It allows duplicates.

(v) It is partially Auto sorted {Either max priority or min priority}

(vi) It doesnot allow null.

(vii) Since it is not indexed type of collection we can't fetch the elements upon index randomly.

(viii) It doesnot follow order of insertion.

Date _/_/_

Set :- Set is an interface which helps to store the data not upon index and doesn't allow duplicates.

In set we have

(i) Hashset
(ii) LinkedHashset
(iii) Treeset

Features of set

(1) Size is dynamic
(2) we can store heterogenous type of data
(3) It is not indexed type of collection
(4) It doesnot allow duplicates
(5) It allows null
(6) since it is not indexed type of collection, we can't fetch the elements upon index randomly
(7) It increases its size by 50%.

Hashset

(i) Size is dynamic
(ii) we can store heterogenous type of data
(iii) it is not indexed type of collection
(iv) It allows null
(v) Since it is not indexed type of collection we can't fetch the elements upon index randomly
(vi) It increases its size by 50%
(vii) It doesnot allow duplicates
(viii) It doesn't follow order of insertion.

Date _/_/_

```
class Demo
{
    public static void main(String[] args)
    {
        Hashset h = new Hashset();
        h.add("pen");
        h.add(10);
        h.add(null);
        h.get(0); X
        System.out.println(h)
    }
}
```

```
for (object element : h)
{
    System.out.println(element);
}
```

o/p [null, 10, pen]

(iv) Treeset :-

→ size is dynamic
→ we can't store heterogeneous type of data.
→ It doesn't allow duplicates.
→ It doesn't allow null.
→ Since It is indexed type of collection we can't get data based upon index.
→ It doesn't follow order of insertion.
→ It is completely auto sorted.

Example :-

```
class Sample
{
    public static void main(String[] args)
    {
        Treeset h = new Treeset();
        h.add(1);
        h.add(21);
        h.add(7);
        h.add(3);
        System.out.println(h);
    }
}
```

o/p

[1, 3, 7, 21]

Date _/_/_

LinkedHashSet:- It is a class which belongs to java.util package.

Features of LinkedHashSet:

(i) Size is dynamic
(ii) we can store heterogeneous type of data.
(iii) It doesn't allow duplicates
(iv) It allows null.
(v) Since it is not indexed type of collection we can't fetch the data based upon index.
(vi) It increases it's size by 50%.
(vii) It follows order of insertion.
(viii) It is not auto sorted.

```
class Sample
{
    public static void main(String[] args)
    {
        LinkedHashSet l1 = new LinkHashSet();
        l1.add("Java");
        l1.add(100);
        l1.add(null);
        System.out.println(l1);
    }
}
```

o/p
[Java, 100, null]

For each loop

⇒ If I want to print the elements of array/collection/map without checking into index, we will go for "for-each-loop".

```
for (arraytype/datatype variable name : variablename)
{

}
```

Example

```
for (int element : arr)        // Traversing in an array.
{
    s.o.p(element);
}
```

Collections

Date _ / _ / _

⇒ It is a class which belong to java.util package, it consists some of the static method like sort, reverse, shuffle etc.

// Sorting a List

```
import java.util;
public class Sample{
    public static void main (String[] args)
    {   ArrayList l1 = new ArrayList();
        l1.add(100);
        l1.add(99);
        l1.add(3);
        l1.add(2);
        l1.add(1);
        System.out.println("Before sorting:" + l1);      // [100, 99, 3, 2, 1]
        Collections.sort(l1);
        System.out.println("After sorting:" + l1);       // [1, 2, 3, 99, 100]
    }
}
```

// Reverse a List

```
import java.util;
Public class sample {
public static void main (String [] args)
    {   ArrayList l1 = new ArrayList();
        l1.add(100);
        l1.add(99);
        l1.add(3);
        l1.add(2);
        l1.add(1);
        Collections.reverse(l1);
        System.out.println("Reversed list:" + l1);  // [1, 2, 3, 99, 100]
    }
}
```

Date _/_/_

// Shuffling a list
Collections.shuffle(l1);
// finding the minimum and maximum
int miny = Collections.min(l1);
int maxy = Collections.max(l1);
// Binary Search
int index = Collections.binarySearch(l1, element);

Q. WAP to copy the element from one Arraylist to another Arraylist.

```
import java.util;
public class Sample {
    public static void main(String[]args)
    {   Arraylist l1 = new Arraylist();
        l1.add(100);
        l1.add(99);
        l1.add(3);
        l1.add(2);
        System.out.println(l1);
        Arraylist l2 = new Arraylist(l1); // copying element.
        System.out.println(l2);
    }
}
```

Wrapper class

→ In Java each and every primitive data type has it over classes called wrapper class.

Primitive data Type	wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Date _/_/_

Q. How the data will get stored in the form of object in Java.

int a = 10;

→ (i) Boxing / autoboxing → converting the primitive datatype into corresponding wrapper class object is called as Boxing.

→ (ii) Unboxing ↳ converting the wrapper class object again back to its corresponding primitive type of data is called as unboxing.

s.o.p(a)

Integer a = new Integer(10); internally

```
public class Sample
{
    public static void main(String[] args)
    {
        int a = 10;
        Integer b = new Integer(a); // boxing
        int c = (int) b; // unboxing
        System.out.println(c);
    }
}
```

Q. Difference between int and Integer

int	Integer
→ int is a primitive datatype.	→ Integer is a wrapper class.
→ int is the datatype which is having a capacity of 32 bits to stored the data	→ Integer is a wrapper class which wraps up the primitive data type into an object.

Generics

(i) When to go for generics?

⇒ If we want to restricts the collection / map to store homogenous type of data then we will go for generics.

```
public class
{
    public static void main(String[] args)
    {
        ArrayList<Integer> l1 = new
        ArrayList<Integer>();
        l1.add(5);
        l1.add(10);
        l1.add(50);
        System.out.println(l1);
    }
}
```

Note:- We can form the generics by using wrapper classes and angular braces.

Syntax : <wrapper class>

Map (I)

Date _ / _ / _

Map is the interface which belong to java.util package.

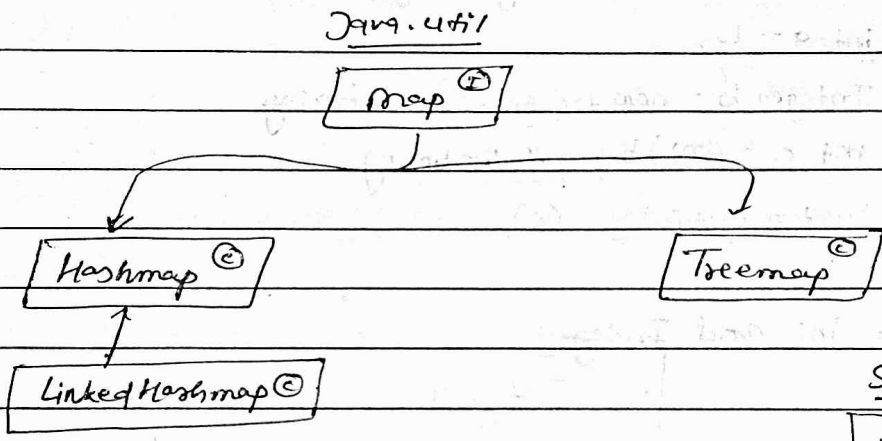
→ Whenever we need to stored the data based upon key and value pair we will go for the map.

→ Map will not allow duplicate keys, But allow duplicate value.

→ Sorting happens in map only for keys based upon ASCII value.

→ To stored the data into map, we will use a non static method called put.

→ Map is by default it is generic.



get (k)
containskey (k)
entrySet()

Store Table

Key	value
"ABD"	10000
"Kohli"	10001
"Dhoni"	9999
"Yuvraj"	2000
"Hitman"	9998

HashMap ⇒ Hashmap is a class which inherits the property from map interface.

features of Hashmap :-

→ Size is dynamic

→ we can store hetrogenous type of data.

→ It can store element based upon key and value pair.

→ Hashmap is not auto sorted.

Date _/_/_

```
import java.util.*;
public class Sample
{  public static void main (String[] args)
   {  HashMap <String, Integer> h = new HashMap <String, Integer> ();
      h.put ("akash", 1000);
      h.put ("rohit", 1015);
      h.put ("chapri", 2);
      System.out.println (h);          // {rohit=1002, akash=1000,
                                          chapri = 2}
   }
}
```

TreeMap

→ Size is dynamic.
→ we can store hetrogeneous type of data
→ we can store data based upon key and value pair.
→ we can't have duplicate keys but we can have duplicate values.
→ It is completely auto sorted.

```
import java.util.*;
public class Sample
{  public static void map (String[] args)
   {  Treemap <String, Integer> h = new Treemap <String, Integer> ();
      h.put ("akshata", 1015);
      h.put ("rakshita", 1015);
      h.put ("varshita", 2000);
      h.put ("Sushmita", 2);
      System.out.println (h);
   }
}
```

O/P
{akshata = 1015, rakshita = 1015, Sushmita = 2, varshita = 2000}

LinkedHashMap

Date _/_/_

→ Size is dynamic
→ We can store data in key and value.
→ Here we can't have duplicate keys but we can have duplicate value.
→ It follow order of insertion.

```
import java.util.*;
public class Sample {
    public static void main (String[]args)
    { LinkHashMap<String,Integer>h=new
                                LinkHashMap<String,Integer>();
      h.put("akshita", 1);
      h.put ("rakshita", 2);
      h.put ("varshita", 3);
      h.put ("sushmita", 4);
      System.out.println(h);
    } }
```

// {akshita=1, rakshita=2, varshita=3, sushmita=4}

Q. WAP to find the order of occurance or no of occurance for the given array by using map or collections.

```
class Sample
{ public static void main (String[] args)
    { int arr[] = {16, 17, 18, 16, 18, 19};
      HashMap<Integer, Integer> h = new HashMap<Integer, Integer>();
      for(int i=0; i<arr.length; i++)
      { if (!h.containsKey(arr[i]))
          { h.put(arr[i], 1);
          }
        else { int cnt = h.get(arr[i])
               h.put(arr[i], cnt+1);
             }
      }
      for (Entry<Integer, Integer> X : h.entrySet())
      {
          System.out.println (X.getKey() + "  " + X.getValue());
      }
    }
}
```

Date _ / _ / _

(#) ContainsKey() → It is a method which comes in map concepts. It will used to check whether the element is present inside map or not.

(#) Entryset() → Entryset() is also method, it will act like vehicle (supporter) to bring the map element to print.

(#) Entry → It is the only interface in java which is having capability to store key and value at a time.

final	finally()	finalize()
→ final is a keyword and this keyword is used to made the variable as final or unchangeable	→ finally() is a block which is used to develop the exception statement where we use this block to store database warning messages. → finally() will get executed even the exception is not handled.	→ finalize() is a me used in garbage colle where we going to throw the unused object outside the program.

Garbage Collection

Date _/_/_

→ When the reference variable get dereferanced from the current object and it is pointing to some other object or unwanted objects (Garbage) should be collected or should be removed. This process is called as garbage collection

→ It will be done through System.gc();

```
public class Avantika
{
    @Override
    void finalize()
    { System.out.println("Garbage collected successfully");
    }
    public static void main(String[] args)
    {
        Avantika a1 = new Avantika();
        Avantika a2 = new Avantika();
        Avantika a3 = new Avantika();
        a1 = null;
        a2 = null;
        System.gc();
    }
}
```

O/P => Garbage collected successfully
Garbage collected successfully

StringBuilder and StringBuffer

Date _ / _ / _

StringBuilder and StringBuffer are classes used for creating and manipulating dynamic strings.

StringBuilder

StringBuilder is designed for use in a single-threaded environment. It is not synchronized, which makes it faster but not thread-safe.

- append (s);
- insert (index, "s");
- replace (6, 10, "s");
- delete (index1, index2);
- reverse ()

```
Public class Aravtika
{
  public static void main (String[] args)
  {
    String S = "Java";
    StringBuilder S1 = new StringBuilder();
    S1.append(S);
    System.out.println(S1.reverse());
  }
}
//avaJ
```

File - Handling

Date _ / _ / _

(#) File is nothing but collection of similar kinds of data.

↳ In java file is inbuilt class which belong to java.io package.

↳ performing operations like creating the file, deleting the file, updating on file is called as file handling.

↳ file class consists of non-static method such as

```
package java.io.*;
class file
{   void mkdir()
    {
    }
    void exits()
    {
    }
    void createnewfile()
    {
    }
    void delete()
    {
    }
}
```

(#) mkdir / make directory (folder)

→ It is used to create a folder,

(#) Createnewfile()

→ It is used to create a new file inside a folder in the requized format.

(#) Exit()

→ It is used to check whether the file or folder is exit or not

(#) Delete()

→ Delete() is a non-static method of file class which is used to delete a file.

Date _ / _ / _

Q. WAP to create a file, to delete a file, to check whether the created file is exist or not?

=>
```
package practice;
import java.io.File;

public class Wafey
{ public static void main (String[] args)
  { File f = new File("C:\\color_juice");
    if(f.mkdir())
    { System.out.println("Folder has been created successfully");
    }
    else { System.out.println("Folder has not been created");
    }
//  if(f.delete())
//    { System.out.println("file has been deleted successfully");
//  }

  if (f.exist())
    { System.out.println("file is existed");
    }
  else
    { System.out.println("file is not existed");
    }
  }
}
```

Date _/_/_

Q. WAP to create a new file inside the existed folder.

```
package practice;
import java.io.File;
import java.io.IOException;

public class Sample
{ public static void main(String[] args) Throws IO Exception
    {   File f = new File("C:\\color-juice\\collcat.txt");
        if (f.createNewFile())
          {
            System.out.println(" file has been created successfully ");
          }
}  }
```

Q. WAP to write the data inside a created file.

File Writer

- File writer is a class which is used to write the data inside the created file.
- File writer is belong to java.io package.
- File writer class has write() method which is non-static in nature used to write the data into the file.
- once after writing the data in the program, that data should be pushed inside the file which we can use flush method.

Date _ / _ / _

```
package practice;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class Samples
{  public static void main (String [] args) throws IOException
   {     File f = new File ("c:\\ color-juice\\Kolkata.txt");
         FileWriter fw = new FileWriter (f);
         fw.write("Java is an emotion");
         fw.flush();
         System.out.println("data has been written successfully");
   }
}
```

Q. WAP to read the data which is already written inside the file.

(#) FileReader → filereader is a in-built class which is presented inside java.io package. It consists a non-static method called read() to read the data from the given file.

→ Here it is having one drawback i.e, it will read the data character by character. So it is to much slow.

```
import java.io.FileReader;

Public class Sample
{  public static void main (String[] args) throws IOException
   {     File f = new File ("c:\\color-juice\\Kolkata.txt");
         FileReader fr = new FileReader(f);
         int length = (int) f.length();
         char [] arr = new char [length];
         fr.read(arr);
         String s = new String(arr);
   }     System.out.println(s);
}
```

Date _/_/_

Buffered writer and Buffered reader

↳ Both are inbuilt - classes present inside java.io package.

↳ By using Bufferedwriter, I can write the data in multiple line in single shot. So my time consumption is very less.

↳ In Bufferedreader instead of read the data char by char we will read the data in the format of sentence. (take string).

```
public static void main (String [] args) Throws IOException
{   File f = new File("C:\\Color_juice\\kolkata.txt");
    FileWriter fw = new FileWriter(f);
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write("water is clear");
    bw.newLine();
    bw.write("air is fresh");
    bw.flush();
}
```

```
class BufferedReader
{
    void readline()
    {
    }
}
```

```
class Sample
{   public static void main (String[] args)
    {
        File f = new File("C:\\my_file\\kolkata.txt");
        FileReader fr = new FileReader(f);
        BufferedReader br = new BufferedReader(fr);
        String z = br.readline();
        while(z!=null)
        {   System.out.println(z);
            z = br.readline();
        }
    }
}
```

kolkata.txt

water is clear
Air is fresh