1. Java is a high level language.
→ high level :- If user is able to understand the language easily.
(User understandable).
→ low level :- system understandable language.
→ Compiler & Interpreter converts High level language to low level language.
(Translator - Compiler & Interpreter)

Compiler → Translate entire program.

Interpreter → line by line translate.

Compiler is faster than ~~Trans~~ Interpreter.

2. → Java is an Object oriented language.
OOPS
↳ Class.
↳ Object.
↳ Inheritance.
↳ Polymorphism.
↳ Abstraction.
↳ Encapsulation.

3. → Java is a robus language (rough & Tough).
1. ↳ ~~Go~~ Garbage is collected through Garbage Collector.
↳ No, ~~gro~~ garbage is generated in Java.
2. ↳ Process managment / memory managment.

4. Java is a secure language.
↳ As it does not directly runs on OS (unlike C++). (using pointers).
Java works/runs on JVM.
(Java virtual Machine)

5. Java ~~ade~~ Edition
↳ Java SE Edition (Standerd Edition)
↳ Java EE (Enterprisers) (Advanced)
↳ Java ME (Micro Edition).

SE → Desktop applications are created.
EE → Used to implement Web/ Enterprisers Application
ME → used for developing Mobile Applications.

6. JDK / JRE / JVM★
[name, work]
↳ JDK → Java Development Kit. (name).
, make Program & run (work).
(/Develop & Run)

eg. College. ↳ JRE → Java Runtime Environment (name).
only running (work).
(By clients /users).

eg. Teacher ↳ JVM → Java Virtual Machine. (name).
JVM is responsible to run Java applications on JRE (work).

- JDK → Development tools + JRE.
- JRE → Class libraries + JVM.

★ JVM is also responsible to alocate memory.
5 memory areas:
↳ Class area / Method area.
★ ↳ stack memory.
★ ↳ Heap memory.
↳ PC register (Program counter).
↳ Native method stack.

Class area : Used to store Class structure (no. of classes, constructor, method, etc).
stack memory : Used to store local variable & retrivin values.
Heap memory : (any thing created at runtime). (objects.)
PC register : store the info of the next program.
Native method stack : (Native methods which are made in other language, but used in Java) (These are stored here).

7. Flow of Java program.

Java program (.java)
↓
Compiler
↓
Byte Code (.class)
↓

NOTE: We don't need to import package with are defaulet and predefined:
Java.util class → predefined (Scanner, Arraylist, etc).
Java.lang class → predefined & defaulet (System class, String class, etc)

↓

Linux / Windows / Mac. (platform).

Java is a platform independent language.
Motive (Write once)
↓
(Compile once)
↓
(Execute anywhere)

But JVM is platform dependent

8. A simple Java program.
(Java is a class based language)

Class LNCT
(scope) {
(size) }

Components inside class
- / Variable
- / methods
- / constructor
- / Blocks
- / inner class.

★ # Variables : are used to store data / value.
(Interview) eg. : int a = 100;
↳ variable.

There are 3 types of variables in Java:
1. local variables
2. Instance variables
3. static variables.

Type

→ Who is responsible to call main method to run Java program : JVM.

```
Class LNCT
{
  public static void main (string[] a)
  {

  }
}
```

→ Local variables.

- Local variables are declared inside ~~classes~~, method, constructor, Block,.
- Local variable's memory is alocated in stack.

(alocate → when method is called)
(de-alocate → when call is over).

→ Instance variables.

- Instance variables are declared inside Class, but outside method / constructor or block.

Memory alocated in Heap.

```
class LNCT {  → (Instance variable)
  int b = 100;
  public static void main( string[] a){

  }
}
```

when object is created, then memory is alocated to instance variable.
and dealocate when object is over.

→ static variable : are declared inside Class and outside method / Constructor / Block, with prifix static.

eg. static int a = 100;

Memory alocated in Class loader or Class area.
(/ Non Heap memory)

Java program
↓
Compiler
↓
Bytecode (.class)
↓
Class loader
↓
Memory alocation.

- static variable's memory alocated only once.
- Jitne object banenge utni bar instance variable ko memory alocate hogie.
- local variable's memory alocated, evertime the method is called.
  / Class.
  / Block.
  / Constructor.

→ JIT : (checks syntax of the program) (compiler)
JIT helps to execute byte code into machine code.
Java donot have pointer, but Java uses pointers ~~imp~~ implicit pointers. (internally)

Order of Java code execution } Java source code → Compilation → Class loader → Byte code Verification → Interpretation → execution.
(Intermediate code understood by JVM)

→ Correct statement of main method.

a. public static void main (string [] args)
b. final public static void main (string [] args)
c. public static void main (string... args)
d. public synchronized static void main (string [] args)
e. public strict static void main (string [] args)
f. final synchronized strictfp public static void main (string[] arge).

→ How many class will be created after compiling the program below : 3

```
Class A {
Void m1() {}
}
Class B {
void m2() {}
}
class C {
Void m3() {}
}
```

→ If (string[] args) is not written, then it will be Run time error.

```
class sample {
    public static void main( ) {
        sout ("Welcome");
    }
}
```

→ If variable is called outside the block, then this will be compilation error.

eg:

```
{
  int y = 6;
  }
  sout (y);
  }
}
```

Block

→ <u>Tokens.</u>

It is the smallest unit of any programming language which is understandable by the Java compiler.

Types of Tokens:

1. Identifiers.
2. ~~Ke~~ Keywords.
3. data type.
4. Literals.
5. Operators.

→ Identifiers. (name)

name in Java program is known as Identifier, which is used for identification purpose.

eg. : class name, object name, method name, variable name, etc.)

How many Identifiers.

```
Class Test
{
public istatic void main(String[] args)
{
    System.out.print(" James ");
}
}
```

① ② ③ ④ ⑤ ⑥ ⑦

class

static variable of system class / Object

method.

How / rules to define Identifiers :

1. a-z, A-Z, 0-9, _, $
2. Can't start identifier from digits.
3. No limit of length for Identifier Name.
4. We can't use reserved words for identifiers. (reserved words - Keywords)
5. ~~Key~~ Identifiers are case sensitive.

→ Keywords.

Keywords are the reserved words. (Pre defined)

Totals Keywords : 50.

Catagories of Keywords.

1. Data type:
   Byte, short, int, long, float, double, char, boolean.
2. Control Flow
   if, else, switch, while, do, for, break, continue, default.
3. Exception Handling.
   try, catch, finally, throw, throws.
4. Java five '5'.
   enum, assert
5. Unused
   Goto, const
   ↓ ↓
   (label final) (Now used (replaced)
6. Object level.
   new, this, super, instance of
7. Class level
   Class, Package, import, extends, implements, abstract, interface.
8. Modifiers
   public, private, protected, ~~default~~, final, static, native, synchronized, transient, strictfp.
9. Method level.
   void, return.
10. Value / Not Keywords
    true, false, null.
    (these are values, not Keywords)

★

→ Data type.

Data types are used to define the types of data.

eg. : int a = 100;

Catagories of Data type :

→ Integral : byte, short, int, long.
(only integeral values, w/o decimal values).

→ Floating point : float, double.
(integral values not allowed, only decimal values)

→ Character : Char.
(a, b, c, d, = + - only 1 character).

→ Boolean : True, false. (Boolean).

Size and Range.

Data Type	Size	Range.
Boolean	1 Byte	true, false
Char	2 Byte	0 to 65535
byte	1 Byte	-128 to 127
short	2 Byte	-32768 to 32767
★ int	4 Byte	-2147483648 to 2147483647
long	8 Byte	more than int.
float	4 Byte	more than long.
double	8 Byte	more than float.

If byte b = 148;

error : possible loss of conversion from int to byte.

Ques. An expression involve byte, int and literal number is promoted to which of these.

(a) int ✓ (c) byte
(b) long (d) float

★ → Implicit type casting. (conversion takes place at the time of compilation).
↓ (automatically) ↓ (conversion)

Note: Boolean is only data type which cannot be converted to any other data type. & Vice-Versa.

If Data types are of same catagory, then they can be converted by checking there size. But if the Data types are of different catagory, then they can be converted by checking there ranges.

same catagory: byte → short → int → long.
↓ ↓ ↓ ↓
size : 1 byte → 2 byte → 4 byte → 8 byte

→ long → float → double.
range: (smaller than float) (smaller than double)

→ char → int.
range: 65535 2147483647

→ int a = 100; (a is variable).

at compile time (CT) 'a'

at runtime (RT) '100'

but when,

final int a = 100; (a is now constant). ★

at CT ⇒ '100',

at RT ⇒ '100',

★ Literals (Value / Data)

1. Integral ~~lit~~ Literal
(byte, short, int, long).

a) ~~Suffix~~ Preffix values.

0X - hexa decimal 0 → zero

0 - Octal

0B - binary

_ - Decimal

↑ space

But Answer will alway come out as Decimal.

```
public static void main(String[] a) {
int a = 100;
int b = 0100;
int c = 0X100;
int d = 0B100;
```

	Output
sout (a);	100
sout (b);	64
sout (c);	256
sout (d);	4
}	

b) • int a = 100; // int ← int.
• ~~int b~~ long b = 100; // long ← int.
(8byte ← 4byte). (Implicit Type Casting).
• We can assign long value, by using suffix 'L' & 'l'.
long c = 100; // long ← long.

c) There is no suffix for byte & short type values. (They store values according to there respective ranges).
byte b = 127; // Valid.
byte c = 128; // byte ← int (Invalid)

2. Floating Point literals (Float, double).
a) double d = 100.5; // Valid.
Every floating point literal is by default double type.

float f = 100.5; // float ← double. Invalid.

b) We can assign float type value by using suffix 'F' or 'f'.

float f = 100.5F; // float ← float, Valid.

3. Character literal (char).

a) It can be represent as single character in a single quote.

```
char ch = 'a';        // Valid
char ch = 'ab';       // Invalid
char ch = ab;         // Invalid
```

b) It can be represented as integral literal.

```
char ch = 127;        // Valid
char ch = 0x127;      // Valid
```

c) It can be represented as unique code.
(syntax of unique code)

'\uxxxx'
↳ 4 digit hexa-decimal value.

eg. :
- '\uabcd'
- '\uab01', etc.

	Min Range	Max Range
• Integral	0	65535
• Unique code	'\u0000'	'\uffff'

```
char ch = '\u000ff'     // Invalid.
```

→ Operators:
Operators are used to perform mathematical operations.

Ques. [* Operator has converted b * 50 into int, which can not be converted to byte without casting]
for → byte b=50;
b = b*50; ← Error.

Ques. double ~~byte~~ a = 295.04; (Explicit type casting)
int ~~byte~~ b = 300;
byte c = (byte) a; c = 295.04 - 256 = 39
byte d = (byte) b; d = 300 - 256 = 44

sout (c + " " + d);

Output : 39 44

Ques. int a = 65597;
short b = (short) a;

32768
32767 + 1
65536

Output = 65597 - 65536 = 61

Ques. int c = 100;
short d = (short) c

Output = 100

Catagories of operators.

1. Arithmetic Operators : + , - , * , / , %
2. Unary Operators : + , - , ++ , --
3. ~~Cont~~ conditional Operators : > , < , >= , <= , == , !=
4. Ternary operator : ? :
5. Logical operator : && , || , !
6. Assignment Operator : = , += , -= , /= , *= , %= , <<= , >>=
   left shift , Right shift.
7. Bitwise operator : & , | , ^
8. shift operator : << , >> .

Ques. int a=10;
int b=(a++) + (++a) : 10 + 12

cout (a + " " + b) : Concat Add.

12 22

30/08/22

→ conditional Operators return Boolean values

eg. : cout (a>=b) a=10
// true b=20.

cout (a > b)
// false.

→ Logical Operators.

1. && - AND operator.

C - Condition

(c_1 && c_2)

c_1	c_2	c_1&&c_2
True	True	True
True	false	false
false.	---	false.

--- means it will not check c_2.

2. || - OR operator.

c_1 || c_2

c_1	c_2	c_1||c_2
false	false	false
false	True	True
True	---	True

eg.: Sout (a>b && b>=50 || 12<=a);

false || false.

|| false.

→ Ternary Operator.

(Condition ? true statement : false statement)

eg: a<b ? Sout ("Hello") : Sout ("Hi");

// error.

~~Cannot~~ Cannot write print statement in place of true & fal

Ques. Find largest no. among 3 no.s.

```
int a = 1
int b = 2
int c = 3.
```

~~(a<b &&~~

```
( a>b && a>c ? a : b>a &&
    b>c ? b : c);
or ( a>b && a>c ? a : b>a && b>c ? b : c);
or (a>b && a>c ? a : (b>c ? b : c));
```

→ Bitwise Operator. (In MCQ questions).

```
int a = 60;
int b = 13;

sout (a&b) ;  // 12
sout (a|b) ;  // 61
sout (a^b)    // 49
```

	2^5	2^4	2^3	2^2	2^1	2^0
60 →	1	1	1	1	0	0
13 →	0	0	1	1	0	1

```
(a)  1 1 1 1 0 0
&
b    0 0 1 1 0 1
=
12   0 0 1 1 0 0
```

(ek bhi 1 hoga to ans = 1)

(a|b)

```
1 1 1 1 0 0
0 0 1 1 0 1
-----------
1 1 1 1 0 1
= 61.
```

XOR[0-0→0, 1-1→0]

(a^b).

1	1	1	1	0	0
0	0	1	1	0	1
1	1	0	0	0	1

= 49

→ Shift Operator.

eg: int a=10;
cout(a<<2);
cout(a>>1);

	32	16	8	4	2	1
a=10			1	0	1	0
a<<2	1	0	1	0	0	0

a<<2 = 40

a = 10.

a>>1 = 5.

[a=10
a<<= 10
a=40]

<<= will store the value in 'a' after shifting.

01/09/22

Ques. int x, y, z; // declaration.
x=0; y=1; // initialisation
x=y=z=8; // Reassigning.

Ques. Class operators

```
{
main (string [] args)
{
int var1 =5;
int var2 =6;
int var3;
var3 = ++ var2 * var1/var2 + var2;
sout (var3);
}
}.
```

Output : 7 * (5/7) + 7
= 17

Ques. main ( )

```
{
int x = 8;
sout (++x * 3 + " " + x);
}
}
```

Output : 27_9

Note: Declaration and initialisation at same time is not valid. (For multiple values.)

int x = y = z = 20 ; // invalid.

Ques. {

```
int a,b,c,d;
a=b=c=d=20.
~~a+=b~~ a+=b-=c*=d/=20.
sout(a+" "+b+" "+c+" "+d);
```

Output:

~~a=21, b=0, c=400, d=1~~

(Right to left)

d=1, c=20, b=0, a=20

Ques. {

```
boolean a= true;
boolean b= !true;
boolean c= a|b;          // true
boolean d= a&b;          // false.
boolean e= d?b:c;

sout(d+" :"+e);
}
```

Output: (false true)

Ques.

```
int x, y=1;
x=10;
if(x!=10 && x/0==0).  (// false)
sout(y)
else.
sout(++y);.
```

Output=2

Ques. boolean a = true;
boolean b = false;
boolean c = a ^ b;
cout (!c);

Output : false.

Ques.

→ Control flow statement :

Control flow statement control the flow of statement.

1. selective statements : (Used for filteration).
2. Iterative statements : (loops)
3. Jump statements.

→ selective statements :
- if else
- switch.

→ Iterative statement :
- for
- dowhile
- while

→ Jump statement
- break.
- continue

→ If else statement.

- if -statement.
- if-else statement.
- if else ladder statement.
- Nested if else statement.

★

Ques. What is the difference between variable and constant expression?

- int a=100;
- int b = 200;
- if(a<b).
- 100<150

Variable	Compile Time	Run Time
a	a	100
b	b	200
a<b	a<b	true
a<150	a<150	true.
100<150	true	true.

Ques. int a=100;
int b;
if (a<150)
b=200;
Sout (b);
Output : error

// error because b is not initialised, if condition a<150 becomes false, b will not be initialised.

(Ques will be in infosis)

Ques if (100 < 150)
b = 200
sout (b) ;
Output : 200

because condition is true at compile time.

Ques. final int a = 100;
int b ;
if (a < 150)
b = 200
sout (b);
(a is const now)
Output : 200.

Ques: int a = 100;
int b ;
if (a < 150)
b = 200;
else
b = 300;
sout (b);
Output : 200.

Ques. Write a program to check a number is even or odd.

Ans. {int a = 8;
if (a%2 == 0){
sout("even");
}else
sout ("odd");
}

Ques.

```
int a;
if (a=1)        // Invalid
    Sout ("Hello");
else
    Sout ("Hi");
```

```
boolean a;
if (a = true)
    // valid.
```

Output : error.

We can assign variable inside the condition part of if statement. But only for Boolean value we can assign values inside condition part of if statement.

03/09/22

[Assignment: Doc file → Assignment no. → Ques → O/P (screenshot) (1st line : name).]

Ques.

```
int var1 = 5;
int var2 = 6;
if ((var2=1) == var1)
        sout (var2);
else.
        sout (++Var2)
```

Output : 2.

Ques.

```
int a=3
int b=6;
int c=a|b;
int d=a&b;
sout (c+"  "+d);
```

Output : 7 2

```
  0 1 1
  1 1 0
  -----
  1 1 1  = 7

  0 1 1
  1 1 0
  -----
  0 1 0  = 2
```

Ques. byte x = 64;
int i;
but y;
i = x << 2;
y = (byte) (x << 2)
sout (i + " " + y);

O/P = 250 0

y = 256 - 256
= 0 //

Ques.

int a = 1;
int b = 2;
int c = 3;
a |= 4; // 5 = a
b >>= 1 // 1 = b
c << = 1 // 6 = c
a ^ = c; // a = 3
sout (a + " " + b + " " + c);

O/P: 3 1 6

b 10
01 1

a = a | 4.
1 0 0 1
1 0 0
1 0 1 = 5

c = 0 1 1
<< c = 1 1 0 = 6

a = 1 0 1
^ c = 1 1 0
0 1 1 = 3

Ques. Find largest no. among 3 nos. using // ladder.

int a = 10;
int b = 120;
int c = 15;
if (a > b && a > c){
sout (a);
}

Note : break cannot be used in if else statement. (Single if else).
(but can be used in switch / loop).

```
else if ( ~~b>a~~ ~~&&~~ b>c) {
  sout (b);
} else {
    sout (c);
}
```

```
if(a>b)                          // Nested.
  { if(a>c){
      sout(a);
  } else {
      sout (c);
  }}else {
        if (b>c) {
          sout (b);
        } else {
          sout (C);
        }
  }
```

→ Switch statement.

if-else	switch
• condition (check)	• expression (match.
• boolean result	• not boolean

Syntax.

```
• switch (expression)
  { case case-label :
        //statement.
        break;
  } default : //statement
```

NOTE: We can compile and run empty switch statement.
// Nothing should be written other than case or default statement.

Ques.
```
switch(10)
{
 case 10:
        Sout ("Hello");
 case 20:
        Sout ("Hi");
 default:
        Sout ("Bye");
}
```

Valid expressions
- Byte
- short
- int
- char
- string
- enum

Invalid Expressions
- boolean
- long
- float
- double.

// We can use if else for boolean values.

Valid case-label
- duplicate case-labels are not allowed.
- case label should be constant or constant expression.
- case label should be in range of expression.
- default ek he hona chahiye (but any where inside switch).

```
case 'a':
        Sout("    ");
case 97:
        Sout("    ");
```

Ques.
```
int a=10;
int b=20;
switch(b)
{
  case 'a':  // case 97:        // Valid
        Sout("Hello");          // a will give error but 'a' will not.
  case 20;
        Sout("Hi");
}

case 97+3;                      // Valid
switch(97+3)                    // Valid
```

NOTE: Case, break & default are optional Keywords of switch statements.

Ques.
```
int a=20;
switch(a)
{                               // Break nahi lagaya isliye.
  case 10:
        Sout("Hello");
  case 20:
        Sout("Hi");
  case 30:
        Sout("Good");
  default:
        Sout("Bye");
}
```

O/P : Hi
Good
Bye

Ques. int a=100;

```
switch (a)
{
  default:
      cout ("Bye");
  Case10 :
      cout ("Hello");
  Case20 :
      cout ("Hi");
  Case 30:
      cout ("Good");
```

// Valid.

O/P: Bye
Hello
Hi
Good.

Ques. int a=10;

```
switch (a)
{ default:
      cout ("Bye");
      Break;
  Case 10 :
      cout ("Hello");
  Case 20 :
      cout ("Hi");
  Case 30 :
      cout ("Good");
}
```

O/P: Hello
Hi
Good

08/09/22

Ques. byte a = 100;

```
switch(a)
{
    case 10:
        Sout ("Hello");
        break;
    case 100:
        Sout ("Hi");
        break;
    case 1000;
        Sout ("Bye");
}
```

Output : Error.

Caselabel should be in range of expression.
(1000 not in the range of 'a')

Ques. Same code as above, but change - switch(a+1).
then the code will run completely because '1' is integer which will change expression into integer type.

Ques. Same code,

if 'case 1+a:' is written, it is invalid.

→ Iterative statements.

Ques. Difference between while loop and for loop.

Ans. • If you dont know iteration in advance then we use 'while' loop and Vice versa for 'For' loop.

Syntax :

1. while loop : It is also known as entry control loop.

```
while (condition)
{
//statements.
}
```

condition - Boolean type.

• Unreachable statement : Unreachable statement when conditions are constant.

```
int a=10;
while (a <= 10)
{
Sout ("Hi");
}
Sout ("Hello");
```

// Unreachable statement.

Ques. Prime no. from (1 to 50)

```
n = 1;
while ( n <51)
{
    if ( n % 2 == 0 && n != 2)
        n ++;
else if ( n % 3 == 0 && n != 3)
        n ++;
  else if (n % 5 == 0 && n != 5 )
        n++
  else if (n % 7 == 0 && n != 7).
        n++
   else
        cout (n);
}
```

Ques. Even no. from (1 to 50).

```
n = 1;
while (n<51)
{
  if (n % 2 == 0)
      cout (n);
} n++;
```

Ques. Stop at 24.

```
n = 1;
while (n < 51) {  cout ( a
  if (n == 24)
      break;
} a++;
```

P.T.O.

```
int n = 1
while ( n<51)
{
  if (n % 2 == 0)
    cout (n);
  if (n == 24)
    break;
  a++;
}
```

Ques. Print even no. from 1 to 50 except (12, 38, 48 .).

```
int n = 1;
while (n <51)
{
  if (n % 2 == 0 && (n != 12 || n != 38 ||
  {                                n != 48))
    cout (n)
  }
  n++;
}
```

or.

```
int a = 1;
while (a< 51)
{ if ( a % 2 == 0) {
    if (a == 12 || a == 38 || a == 48 ){
      a++;
      continue ;}
    cout (a);
}}a++;
```

→ do while. (exit control loop)

Syntax

```
do
{
// statement;

} while (condition);
```

Ques.

```
final int a = 1;
do
{
 Sout ("Hello");
}
while (a<=10);
```

O/P : infinte loop.

Ques.

```
final int a = 1;
 do
 {
 Sout ("Hello");
 } while( a> =10);
 Sout ("Hi");
}
}
```

O/P : Hello
Hi

20/09/22

→ For loop.

Syntax : for( )
{
// statement
}

① ②, ⑤ ④, ⑦

for(initialisation ; condition ; inc/dec)
{
//statement ③, ⑥ [steps]
}

// initialisation is done only once.

int i=1;

Ques. for(sout("Hi") ; i<=5 ; i++)
{
sout("Hello");
}.

Output : Hi
Hello
Hello
Hello
Hello
Hello.

We can write print statement in place of initialisation statement.

Ques. int i=1.
for (sout("Hi") ; i<=5 ; sout("Bye")){
sout("Hello");
}

Output : Hi
Hello
Bye
Hello
Bye ——— ∞.

// Infinite loop.

Note: Print statement cannot be written at place of condition part, because condition part must be boolean value. (But in C we can write).

Note: for(;;) → Infinite loop.
By default, the conditions are true, so the loop runs infinitly.

Ques.
```
int i=1;
for(i<5; i==5; i++)
{
  sout("Hello");
}
```
Output: No error & No ~~ot~~ output, but program will run.

22/09/22
→ Scanner Class.

1. It is a pre-defined Class.
2. Scanner stored in java.util package.
   java
   ↳ util (util have so many classes)
   ↳ Scanner.class.
3. It is mandatory to import scanner class from java.util package.
4. Scanner Class is used to take input from user.
5. How to create an object of scanner class.
   (Class) (object) (Constructor)
   Scanner sc = new Scanner(System.in);
   ↳ (to alocate memory in Heap)

6. Methods of Scanner Class.

Data type	Method.
byte	nextByte()
short	nextShort()
int	nextInt()
long	nextLong()
~~double~~ float	nextFloat()
double	nextDouble()
boolean	nextBoolean()
string	1. next()
	2. nextLine()
char	1. next().charAt()
	2. nextline().charAt().

Ques.
```
Scanner sc = new Scanner(System.in);
string a = sc.next();
sout(a);
```
Input: Vatsal Kashyap.
Output: Vatsal.

Ques.
```
string a = sc.nextLine();
sout(a);
```
Input: Vatsal Kashyap
Output: Vatsal Kashyap.

Input.

Ques. string a = sc.next(); // Vatsal Kashyap.
string b = sc.nextLine; // Kuch bhi.
sout(a);
sout(b);
Output: Vatsal
␣Kashyap

sc.next. Vatsal␣
curse on the same line. not in the next line.

Ques. string a = sc.next();
sc.nextLine();
string b = sc.nextLine();
sout(a);
sout(b);
Output: Vatsal
Mai Hoon.

Input
// Vatsal Kashyap.
// Mai Hoon.

Input:

Ques. int a = sc.nextInt(); // 100
string b = sc.nextLine(); // Vatsal.

sout(a);
sout(b);

Output: 100
␣

next() method takes input by removing leading space.
"␣Vatsal"
Output: "Vatsal"

Ques. char a = sc.next().charAt(9);
// Vatsal Kashyap.

sout(a).

Output: string out of bounds exception.

Ques. char a = sc.nextLine().charAt(9);
sout (a); // Praveen sharma.
Output : "h"

Ques. Campus Question.

- inputs 1 → x-axis = 3
  y-axis = 3 { Range }
  string : current position = 2-2-E // (2,2) East.
  string : new-position = R M L M L M
  // path
  (Right, Left, Move).

Explain :

2-2-E → 2-2-S → 2-1-S → 2-1-E → 3-1-E → 3-1-N → 3-2-N.

- Input 2 : x axis = 3
  y axis = 4
  current position = 2-2-E
  path = RMLMLMRM.

2-2-E → 2-2-S → 2-1-S → 2-1-E → 3-1-E → 3-1-N → 3-2-N → 3-2-E → 4-2-E
↳ If limit exceeds,
then Output : 3-2-E-ER
↳ Error.

Sol^n:

```
int xm = sc.nextInt();
int ym = sc.nextInt();
string curr_pos = sc.next();
string path = sc.nextLine();
```

01/10/22

OOP

→ Object oriented programming.

- ↳ Variables
- ↳ Methods
- ↳ Constructor
- ↳ Block
- ↳ Inner class.

} Basic.

Inheritance
Polymorphism
Encapsulation
Abstraction

} Concept.

- ↳ Abstract class
- ↳ Interface.

★ Methods.

1. Methods are used to write logics of the program.
2. There are 2 types of methods:
   a. Instance method.
   b. static method.

example:

```
class LNCT{
        void show()        //Instance.
        {
        }
        static void display()   //static.
        {
        }
}
```

NOTE → Program always start from "main" method.

JVM is responsible to call main method to execute Java program.

```
class LNCT {

    void show{
     sout("Hi");
    }
    static void display() {
      sout("Bye");
    }
    public static void main(string[] args){
     sout("Hello");
    }
```

OUTPUT : Hello.

→ Calling of method.

1. Instance method $\xrightarrow{\text{directly}}$ Instance method / variable directly.
2. Static Method $\xrightarrow[\text{object}]{\text{making}}$ Instance method / variable
3. Static Method $\xrightarrow[\text{class-name}]{}$ static method / variable.
4. Instance Method. $\xrightarrow[\text{class-name}]{}$ static method/variable.

→ Object:

1. object is a physical entity.
2. object is an instance of class.

→ Ways to create object of a class.

1. Named object.
2. Nameless object. / Anonymous object
3. Instance Factory method. } Polymorphism.
4. static Factory method.

1. Named object.

Class name - object name = new constructor-name();

2. Nameless object

new constructor-name();

Constructor is used to initialise the object of a class.

→
```
Class LNCT{
  void show(){
    sout("Bye");
  }
  static void display(){
    sout("Hi");
  }
  public static void main(string[] args){
    LNCT obj1 = new LNCT();
    obj1.show();
    LNCT.display();
  }
```

Output : "Bye"
"Hi"

Note: void show(int a) { → local variable.
sout ("Bye");
}

- parameter of any method is always local.

Note: Class LNCT{ → Parameter.
void show (int a) {
sout ("Bye");
}
public static void main (string[] args){
LNCT obj1 = new LNCT();
obj1. show(100);
↳ argument.

// or
new LNCT(). show (100);
// without using object.

}

☆ Instance method is also called non-static method.

Ques. Class LNCT{
int b = 1000;
void show(int a)
{
sout (b);
}
}

Output: 1000

→ 'this' is keyword.
'this' keyword is represent as current class object.

eg.: class LNCT {
    int a = 1000;
    void show (int a) {
        sout (a);
    }
    main(string[ ] args){
        new LNCT ( ). show (100);
    }

Output : 100.

If we use this keyword.

class LNCT{
    int a = 1000;
    void show (int a){
        sout (this. a);
    }
    main (string[ ] args){
        new LNCT ( ). show (100);
    }

Output : 1000.

Ques. Is it possible to call static method or method variable using this keyword.

→ Method signature.

1. Method signature always check by compiler.
2. eg. int show (int a, int b).

↓

signature = show(int, int).

☆ Constructor.

1. Constructor is a special type of method which is invoke implecitly at the time of object creation.
2. Constructor is used to initialise object of a class.
3. Constructor is also used used to initialise instance variable.

→ Rules to define constructor.

1. Constructor name & class name must be same.
2. Constructor are able to pass parameter.
3. Constructor doesnot have any return type.
   (if we put return type in constructor, then it will behave as normal method & it can be called implicitly).

→ There are 2 types of constructor:

1. Default constructor. (created by compiler (if not created by user))
2. User defined constructor.

	Default constructor	User-defined constructor
created By:	compiler at compile time	user.
	Non-Parameterised or 0-Parameterised ↳ zero.	0-Parameterised / more-Parameterised
eg.:	Class LNCT { public static void main() { LNCT obj1 = new LNCT(); }	eg.: Class LNCT { LNCT() { } public — — main() { LNCT obj1 = new LNC }

Ques.

```
Class LNCT{
int a= 100;
LNCT(){

}
public static void main(string[] args)
{
LNCT obj = new LNCT();
sout(obj1.a);
}
```

OUTPUT : 100

```
Class LNCT
{
 int a ;        // a=0
 LNCT() {
      // a=0,                    // If a=100.
 }
 public static void main( String[] args).
 {
   LNCT obj1 = new LNCT();
   sout(obj1.a);                 // 100.
 }
```

Output = 0 //

NOTE: → Constructor is also used to initialise instance variable.

Constructor initialise with default value.

Data type : →	Default Value.
byte, int, short, long	0
float, double.	0.0
String	null
Boolean	false
char.	' ' (space)

01/10/22 → Constructor can be kept private. only If the constructor is private, then the Class will be able to use that constructor. so the Object will also be created in the same class.

Example: Class LNCT

```
{
 private LNCT()
 {
   sout("constructor - 1");
 }
 main(          ) {
   LNCT obj1 = new LNCT();
 }
}
```

Output : Constructor - 1.

Ques. Class Test100

```
{
 private Test100()
 {
   sout("constructor-1");
 }
}
```

Class LNCT

```
{
 public static void main(          )
 {
   Test100 obj1 = new Test100();
 }
}
```

Output: error.

private constructor.

→ Class Test100
{
private Test100()
{
sout ("constructor-1");
}
//Test 100 get Instance () // Don't Write.
static Test 100 get Instance ()
{
return new Test100 ();
}
}
Class LNCT
{
public static void main (string [] args).
{
Test 100 obj1 = Test100. get Instance ();
}
}

☆ this Keyword in Constructor.

```
Class LNCT {
    LNCT () {
        Sout ("constructor-1");
    }
    LNCT (int a) {
        sout ("constructor-2");
    }
    LNCT (string a, int b) {
        sout ("constructor-3");
    }
```

```
public static void main(string[] args){
  LNCT obj 1 = new LNCT();
  LNCT obj 2 = new LNCT(10);
  LNCT obj 3 = new LNCT("abc", 10);
```

Ques. How to call all 3 constructor by creating only 1 ~~constructor~~ object

```
Class LNCT{
  LNCT(){ this("abc", 10);
    sout("constructor-1");
  }
  LNCT(int a){
    sout("constructor-2");
  }
  LNCT(String a, int b){
    this(10);
    sout("constructor-3");
  }
public static void main(string[] args){
  LNCT obj1 = new LNCT();
}
```

→ How to use 'this' in method.

```
Class LNCT{
    string a= "James";
    int b= 200;
    LNCT ( string a, int b){
        cout ("Constructor-3");
    }
    main ( ) {
      LNCT obj1 = new LNCT(); //Error.
      LNCT obj1 = new ("Petter", 200); // Constructor -3
    }
}
```

→

```
Class LNCT {
    string a;
    int b;
    LNCT ( string a, int b){
        this.a = a;
        this.b = b;
    }
    void show(){
        cout (a+" " +b);
    }
    main (     ){
    LNCT obj1 = new LNCT("Petter", 200);
    obj1. show();
    }
}
```

Output:

→ Blocks (Used to initialize variables).

- Instance block → Used to initialise Instance variable.
- static block → Used to initialise static variable

```
{
  //statement
}
```

↑ Instance block.

```
static {

}
```

↑ static block.

Example:

```
Class LNCT{
    LNCT(){
        sout("Constructor-1");
    }
    {
        sout("Instance block-1);
    }
    main(  ){
        LNCT obj1 = new LNCT();
    }
}
```

Instance block execute at the time of object creation. Instance block invoke implicitly at the time of object creation.

Output : Instance block - 1
constructor - 1.

Ques. Class LNCT{

```
    LNCT(){
        sout("constructor 1");
    }
    LNCT(int a){
        sout("constructor 2");
    }
    {
    sout("IB - 1");
    }
    main(){
        LNCT obj1 = new LNCT();
        LNCT obj2 = new LNCT(10);
    }
}
```

Output: IB1
constructor 1
IB-1
constructor -2

NOTE: Instance block logic is same / common for all object but constructor logics are same for same object.

Ques. Class LNCT{

```
    LNCT(){
        sout("C1")
    }
    LNCT(int a){
        sout("C2");
    }
```

```
{
 sout ("IB-1");
}
static {
   sout ("SB-1");
}
main ( ){
   LNCT obj1 = new LNCT();
   LNCT obj2 = new LNCT (10);
}
```

```
{ sout ("IB-1");
}
{ sout ("IB-2");
}
static {
 sout ("SB-1");
}
static {
 sout ("SB-2");
}
```

Output: SB-1
IB-1
C-1
IB-1
C-2.

If there are 2 SB & 2 IB then O/P will be:
SB-1
SB-2
IB-1
IB-2
C-1
IB-1
IB-2
C-2

Note: • static block invoke before main method
• static block execute only once at the start of program.

11/10/22

☆ Inheritance:

One class can aquire all the properties & behaviours of another class.

(Its properties are accessed) (Base class) [A] Parent class (/super) class

(extends)

(Its access the property of other class) (derived class) [B] child class (/sub) class

NOTE: Parent of every class is object class. (Written by defaulet)

→ Class A {	class A extends Object
}	{
	}
Class B extends A{	Class B extends A
	{
}	}

NOTE: Inheritance is also known as IS-A relationship.

→ final Class A
{
}
Class B extends A
{
}

// Invalid.

→ Class ke aage final laga hai to uska child create nahi kar sakte.

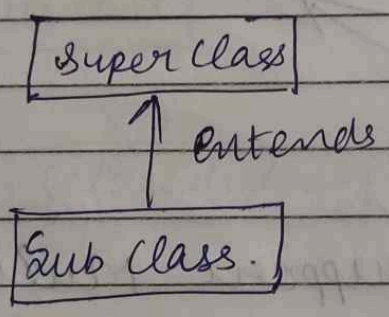
→ Class A. // Valid.
{
}
final class B extends A
{
}

✱ Types of Inheritance.

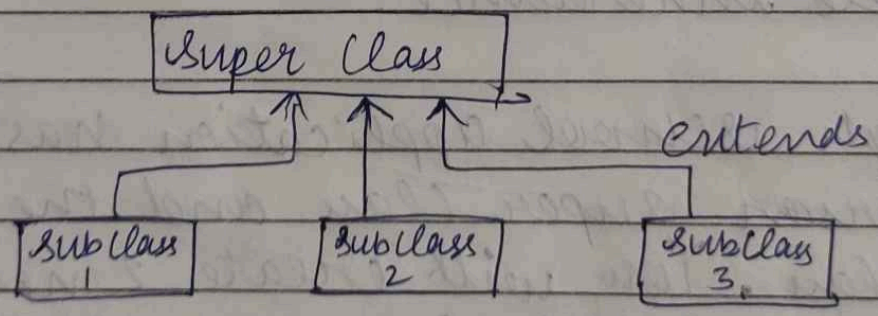
There are 5 types of inheritance:

1. Single inheritance

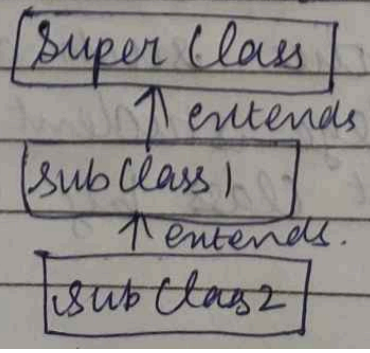
Ex



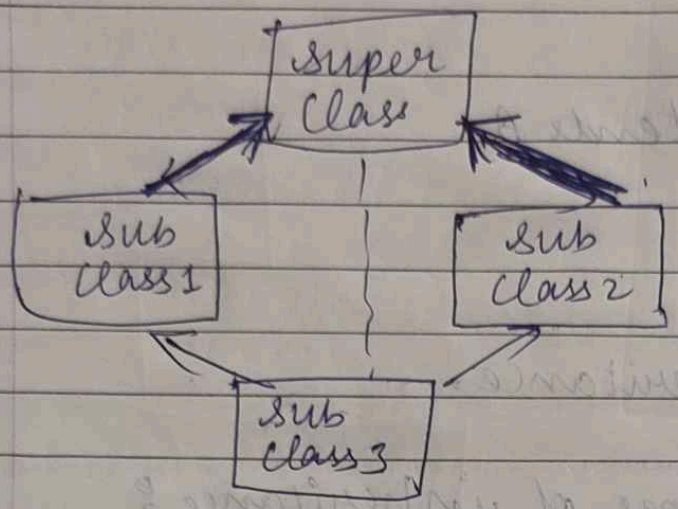
2. Hierarchial inheritance.



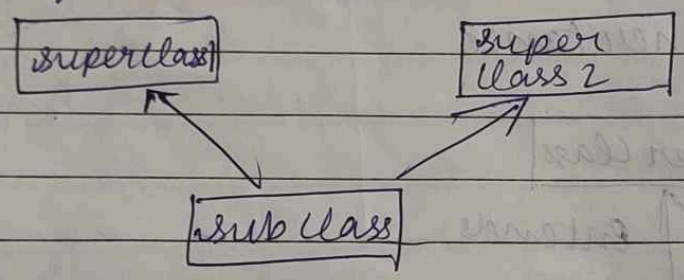
3. Multilevel Inheritance.



4. Hybrid Inheritance. (If 2 or more type of Inheritance are used)

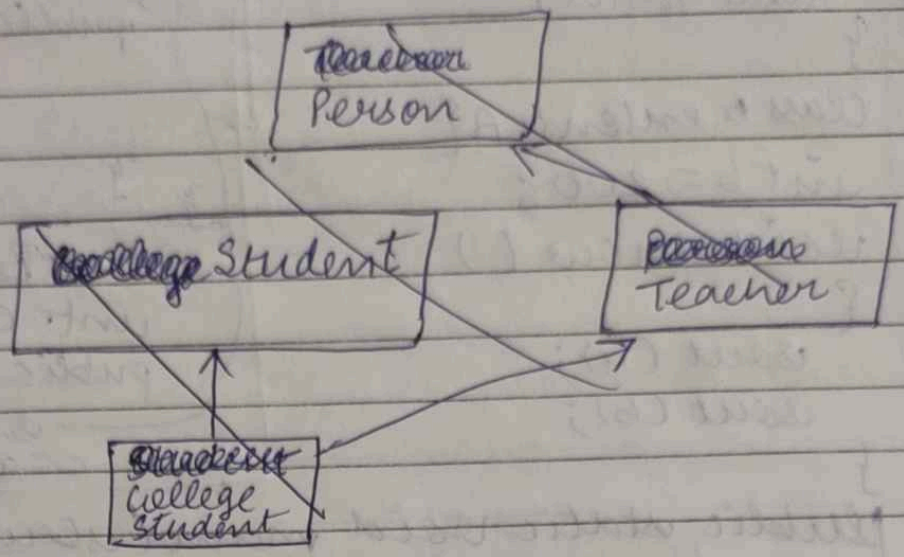


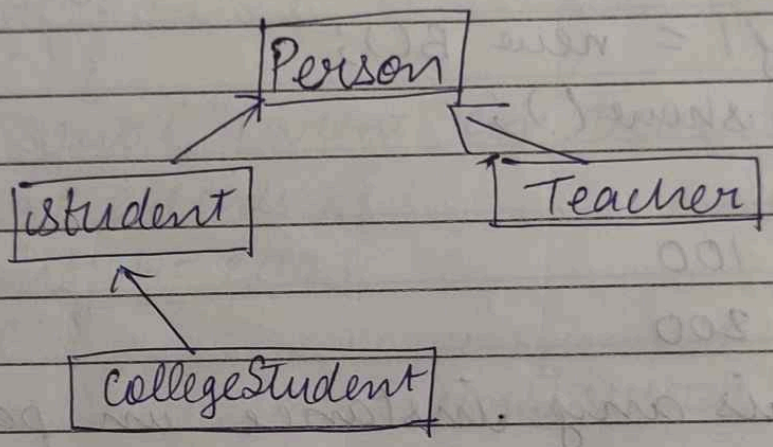
5. Multiple Inheritance.



Note: Java does not support Multiple Inheritance through Class.
We can use Interface concept instead of multiple inheritance.

Ques.→ A high school application has 2 classes, the person super class and the student sub class. You will create 2 new classes. Teacher and CollegeStudent. A Teacher will be like person but will have additional properties such as salary & subject, the college student class will extend the student class by adding year & major





Ques. Class A

```
{ int a = 100;

}
Class B extends A
{
  int b = 200;
  public static void main (String[] args)
  {
    B obj1 = new B();
    sout (obj1.b);
    sout (obj1.a);
}
```

Output : 200
100

always 1st statement.

```
class A {
    int a;
    int b;
    public A( int a1, int b1){
        a = a1;
        b = b1;
    }
}
class B extends A{
    int c;
    public B (int a2, int b2, int c2){
        super(b2, c2);
        c = a2.
    }
}
```

Ques.
```
Class A{
    int a = 100;
}
Class B extends A{
int b = 200;
void show ()
{
    sout (a);
    sout (b);
}
public static void main (string[] args) {
    B obj1 = new B();
    obj1. show ();
}
```

Output : 100
200

Note: If there is any instance in parent, then child can call it easily.

Ques.
```
Class A{
int a = 100;
}
Class B extends A{
int a = 200;
void show (int a) {
    sout (a);   // 300
    sout (a);   // 300
    sout (a);   // 300
}
main (  ){
    B obj1 = new B();
    obj1. show();
}
```

Output:
300
300
300

To get all the values, so.

```
sout (super.a);    // 100
sout (this.a);     // 200
sout (a);          // 300
```

☆ Constructor calling in Inheritance.

```
Class A{
  int a=100;
  A(){
    sout("Parent constructor");
}Class B extends A{
  int b=200;
  B(){ // super() by default by compiler
    sout("Child constructor");
  }
}main(  ){
  B obj1 = new B();
}
}
```

Output: Parent cons.
Child cons.

Note: After calling 'B()' constructor, By default super(); is used at compile time by the compiler, so before sout("Child constructor"); 'A()' is called.

access.

```
}
class child extends Parent{
  obj1.a ; //invalid.
  obj.b ; //valid
```

Imp →

```
Class A{
int a=100;
A(int a){
    system.out.("Parent constructor");
  }
}
Class B extends A{
  int b=200;
  B(){ //To correct : super(100);
      systout("Child constructor");
  }
  main(){
    B obj1 =new B();
  }
}
```

Output : error.
Because compiler doesnot pass any parameter in super() by defaulet.
(Read To correct).

NOTE: Advantage of inheritance is, we can achive overriding using inheritance.

Imp → Role of object creation in Inheritance.

P - Parent
C - Child.

Object - Scope.

1. P p = new P(); p - parent //Valid
2. C c = new C(); c - child, Parent //Valid.

3. P c = new C(); c - Child, Parent // Valid
4. C p = new P(); p - Parent // Invalid

$\underbrace{\text{Class-name}}_{\text{use}}$ $\underbrace{\text{object-name = new Class-name();}}_{\text{object.}}$

13/10/22

☆ Scope Resolution operator.

Access specifier = Access modifier.

→ Access specifier : Access specifier is used to determine scope of class and class elements.

There are 4 types of Access specifier in Java:

1. Public → Class & class element.
2. Private → ~~class &~~ Class elements
3. ~~Private~~ Protected → Class elements.
4. Default. → Class and Class element

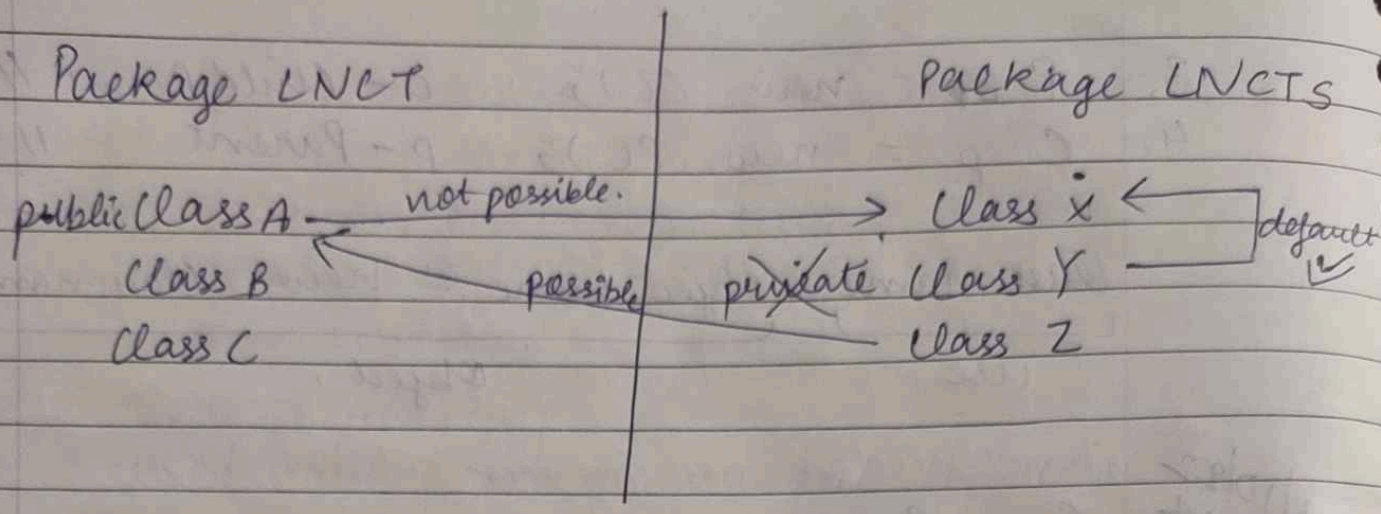
Example:

Class A {
}

→ If nothing is written, then default is considered by default.

Note: Default specifier's scope is within a package.

Scope of public specifier is within a package & outside the package.



NOTE: We cannot keep outer class private, we can only keep inner classes private.

Private access modifier is accessible only within a class.
A Class cannot be private or protected except nested class.

Example: private variable & method.

```
Class A {
private int data = 40;
private void msg()
 {
 sout ("Hello java");
 }
Class B
```

Javatpoint

★ protected access Modifier.

- The protected access modifier is accessible within package and outside the package but through inheritance only.
- The protected access specifier can be applied on the data member method & constructor.
- It can't be applied on the class.

★ Public access Modifier.

- Public access modifier is accessible everywhere.
- It has the widest scope among all other modifiers.

15/10/22

★ Polymorphism

Poly + morphism

Poly → More than one

morphism → behaviour

There are 2 types of polymorphism:

1. Compile time polymorphism and
2. Run time polymorphism.

Compile time →	checked by compiler	To achive
Run time →	JVM	Method over loading
		Method over riding

* Method Overloading is achive for the purpose to increase Readability of program.
* Method overriding is achived for the purpose to provide specific implementation.

☆ Method overloading:
- Method having same name but different parameter is known as method overloading.
- Method overloading is performed in the same class. (No need to make more than 1 class).

eg.:
```
Class B {
    void show () {
        sout("Method-1");
    }
    Void show (int a) {
        sout("Method-2");
    }
    public static void main (String [] args) {

    }
}
```

- In overloading it is not nessacary to notice the return type, only method name & parameter are considered.
- Compiler checks method signature for overloading

(Method signature(Method-name (parameter-list)))

eg.:

```
Class B{
    void show(byte a){
        sout ("Method-1");
    }
    void show (int a){
        sout ("Method-2");
    }
    public static void main(String[] args){
        B b1 = new B();
        b1.show(10);          // show int
    }
}
```

Output: "Method-2"

To call method 1,

```
byte b=10;
b1.show(b);
```

eg.:

```
Class B{
    void show(char a){
        sout ("Method-1");
    }
    ~~void show~~
    public static void main( ){
        B b1 = new B();
        b1.show('a');          // valid
        b1.show(97);           // Invalid, error.
        b1.show((char)97);     // Valid
    }
}
```

eg. :
```
Class B{
    void show( int a, string b){
        sout("M-1");
    }
    void show( string b, int a){
        sout("M-2");
    }
    main( ){
        B b1 = new B();
        b1.show(101, "Method");
    }
}
```

Note: By changing the position of parameters, then also method overloading occurs.

eg. :
```
Class B{
    void show(int... a){
        sout("M-1");
    main( ){
        B b1 = new B();
        b1.show(10);
    }
}
```

Note: (int... a)
↑ variable argument.
(vararg)

varargs will be called in following calls.

1. b1. show (10);
2. b1. show ( );
3. b1. show (10, 20, 30, 40, 50);

Note: If any method is present with the specific parameter then the varaerg will not be called.

eg.:
```
void a(int... a){
}
void a(int a){    // This will be called.
}
main(){
B b1 = new B();
b1.a(10);
}
}
```

eg.:
```
Class A{
    void show(int... a, string b){
    sout("M-1");
    }
main( ){
    A b1 = new A();
    b1. show (10, 20, 30, 40, "abc");
}
}
```

Output: error.

Note: Vararge should always be the last parameter

of any method.
eg.: int show (string b, int... a);

NOTE: There shouldn't be more than 1 vararg in one method. (It will generate error)
eg.: void show (int... a, string... b);
// error.

```
eg.: class B{
        void show (int... a){
          sout("M1");
        }
        main (   ){
        B b1 = new B();
        b1.show ('a', 'b', 'c', 'd');
    }
}.
```

Implicit Type Casting
(char → int)

Output: "M1"

★ working of vararg.

void show (int... a) // int []
//works same as array.

```
eg.: class B{
        void show (int... a){
          sout (a[0]);
          sout (a[1]);
      main (string [] args){
        B b1 = new B();
        b1.show ('a', 'b', 'c', 'd', 'e');
    }
}
```

Output : 97 //
98 //

★ Method Overriding.

- Method having same name, with same parameter but in different class & class should be in inheritance (or is-a relationship).

eg. :
```
Class A {
    void show ( int a){      // common for all implementation
    }
}
Class B extends A{
    void show(int a){        // specific implentation (Child)
    }
    main (    ){

    }
}
```

eg. :
```
Class A{
    void show ( int a){
        sout ("M-1");
    }
}
Class B extends A{
    void show(int a){
        sout("M-2");
    }
```

```
        main(  ){
        A b1 = new B();
        b1.show(10);
    }
}
```

Output : "M2".

eg. :

```
    class A {
        static void show(int a){
        sout("M 1");
        }
    }
    class B extends A{
        static void show(int a){
        sout("M 2");
        }
    public main( ){
        A b1= new B();
        b1.show(10);
    }
}
```

Output : "M1".

NOTE: If we use static, then overriding doesnot take place.

NOTE: If we use finile for a method, then overriding doesnot take place.

```
class A{
eg. : final void show (int a) {
      }
    }
    Class B extends A{
      void show (int a){        //error on this line.
      }
    main ( ) {
    }
  }.
```

18/10/22

☆ Abstraction.

Abstraction is a process in which hiding the implimentation details and showing only functionality to the user.

There are two ways to achieve abstraction:

1. By abstract class
2. By interface.

We can't instantiate (Object creation) abstract class & interface.

☆ Abstract Class.

A class which is declared as abstract is abstract class.

```
eg: abstract Class B {
```

2. Abstract class consists both abstract method and non abstract method.

eg.:
```
abstract class B
{
    abstract void show();
                        // abstract method.
    void display(){
                        // Non-abstract method.
    }
```

A method which is declared as abstract and without implementation is called abstract method.

3. Abstract class is needs to be extended and its method implemented class.

eg.:
```
abstract class A{
    abstract void show();
                        // abstract method.
    void display(){
        sout("Non-abstract method");
    }
}
class B extends A{
    void show()
    {
    sout("Abstract method implementation");
    }
}
```

- abstract method
  ↳ Parent class
- implementation method.
  ↳ Child class.

4. Abstract class cannot be instantiated (object creation).

eg:
```
abstract class A{
        abstract void show();
        void display(){
            sout("Non - AM");
        }
}
class B extends A{
    void show(){
        sout("AM implementation");
    }
main(){
A b1 = new B();        // Valid
b1.display();
b1.show();
A b1 = new A();        // Invalid.
}
}
```

Note: We can create constructor of abstract class.

eg:
```
abstract class A{
        A(){
            sout("AC - constructor");
        }
// rest all same as above.
```

O/P :

current obj = new

eg.: 1. abstract class A // Valid.

```
{
    void display()
    {
    }
}
```

2. abstract class A // Invalid.

```
{
    abstract void display();
    {
    }
}
```

(cannot implement)

3. abstract class A // Valid.

```
{
    abstract void display();
}
```

4. Class A // Invalid

```
{
    abstract void display();
}
```

(class must be abstract).

5. Class A // Invalid

```
{
    Void display();
}
```

(always Implementation of non-abstract method)

6. Class A // Valid.

```
{ void display() { }
}
```

☆ Interface.

There are mainly 3 reason to use interface :

1. We can achieve abstraction using ~~inherit~~ interface
2. By interface, we can support the functionality of multiple inheritance.
3. Interface can be used to achieve loss coupling (loss coupling ⇒ less dependency).

Coupling

eg: Class A
{
}
Class B
{
~~A[10]=new~~
A a1 = new A();
}

Tight coupling.

(B is dependent upon A)

→ Syntax of Interface.

```
interface Test
{
}
```

eg: interface Test
{
int temp = 100;
void show();
}

at compile time compiler by defaulf puts these preffix

```
interface Test
{
    public static final int temp= 100;
    public abstract void show();
}
```

☆ Relation b/w classes & interface.

Class —Extend→ Class ✓

Interface —Extend→ Interface ✓

Class —implement→ Interface ✓

interface —Invalid→ Class ✗

Note: We cannot put ~~non~~ implemented method inside interface.

eg.:

```
interface A {                    // Invalid.
    void display (){
    }
}
```

eg. : interface A{
void display();
}
class B implements A{
void display(){
sout("Implimentation");
}
main( ) {
A b1 = new B();
b1.display();
}
}

Output : error.

correction

when ever implent method of interface use public suffix (Weaker access privilage).

eg. : interface A{
void display();
}
Class B implements A{
public void display(){
// same as above.

Implemented

void show() {}
void show() {}

Non-implemented

void show();
(implementation in child)
Void show();

implemented in parent

Class A
Class B
Extends
Class c
C c1 = new C();
C1.show();
error: imbiguity

interface A
interface B
Class c
void show() {}
child
C c1 = new C();
C1.show
No imbiguity

eg.: interface A{
void show();
}
interface B{
void show();
}
Class C implements A,B
{
public void show(){
sout("Implemented Method");
}
public static void main(string[] args){
C cl = new C();
cl.show();
}
}

We can put A or B also.

Output : "Implemented method"

20/10/22

Note: We can also provide implemented method to the interface. (This was introduced from java 8).

eg.: interface A{
void show();
default void display(){
}
}
}
class B implements A{
public void show(){}
}

NOTE: Any default method can be overrided.

eg. :
```
interface A{
    void show();
    default void display(){
        sout("Method-1");
    }
}
Class B implements A{
    public void show(){
    }
    public void display(){
        sout("Method-2");
    }
    public static void main(String[] args){
        A b1 = new B();
        b1.display();
    }
}
```

Output : Method -2

Note : Interface doesnot have constructor.

15/09/22

→ string
sequence of character which is represented in "——"
String is a predefined Class in Java.
String is considered as an object in java, which is stored in heap memory.
Not a data type
String is of 2 types : → Immutable (Unchangable).
(Class) → mutable. (Changable).

→ string (Immutable)
→ string Buffer } Mutable
→ string Builder }

{ strings are unchangable, but we can reassign }

→ Ways to create string.
1. Using string literal.
String S1 = "Hello";
2. Using new Keyword.
String S2 = new String ("Hello");

in string, duplicates objects are not allowed.
in Heap, duplicates are allowed.

20/09/22.

Literals → always make reference object.

string a = "Hello";
↳ reference object/ variable which is using 101 memory.

string b = "Hello";
↳ b is also reference variable which will use 101 memory.

SCP
101: Hello.
↓
Memory is assigned to Hello not a ob.

SCP – String Constant Pool

string c = new string ("Hello");
string d = new string ("Hello");

Heap
a[Hello]
b[Hello]

Heap: seperate memory allocation.

compareTo → If value of s1 is greater than return +ve value.
→ If value of s2 is greater than return -ve value.

s1.compareTo(s2);

Different ways to compare strings

1. == (reference variable will be compare).
2. equals( ). (string values will be compare).
3. equalsIgnoreCase( )
4. compareTo. (character by character of the string lexicographically).
   compareTo (return type integer)

eg. :

```
string s1 = "Hello";
string s2 = new string ("Hello");
string s3 = "Hello";
string s4 = new string ("Hello");

sout (s1 == s3);
sout (s2 == s4);
```

Output : ~~False~~ True
False

```
Sout (s1 == s2)        // False.
s1.equals(s2);         // True.
s3.equals(s4);         // True.
```

eg. : "sachin" (compareTo)
"Sehwag"

	s	9
	18	97
s	18	101
	(0)	(-5)
		(e)

eg. : S_1 = "Sachin";
S_2 = "Sachin";

S_1 . compare To (S2)
Output = 0.

String method :

1. charAt();
2. substring()
3. index of()
4. to Lower Case()
5. to Upper Case()
6. replace ()
7. Value of()
8. concat()
9. +
10. equals() , equals Ignore Case().
11. Compare To().
12. length().
13. starts with()
14. trim()
→ split()
→ last Index of()
→ end with()

S_1 = "Sachin tendulkar".

eg. : sout (charAt (0)) // S
sout (substring (3,7)) // hin t
sout (index of ("c")) // 2
sout (last index of ("a") // 14
sout (

Ques. Write a program to copy One string to another string.

Ans.

```
class a{
    main (String[] args){
        string S1 = "Vatsal";
        string S2 = "";
        for (i=0; i< S1.length(); i++)
        {
            ~~string(S2.+S1.charAt(i))~~
            S2 = S2 + S1.charAt(i);
        }
        sout (S2);
    }
```

Ques. Write a program to convert lower case to Upper case.

Ans.

```
S1 = "india".
for (i=0; i< S1.length(); i++)
{
    S1 = (char) ((int) S1.charAt(i) - 32);
} sout (S1);
```

or (char) (S1.charAt(i) - 32);

20/09/

return type	method.
int	length()
boolean	equals ()
boolean	equalsIgnoreCase()
~~int~~ String	concat()
string	toUpperCase ()
string	toLowerCase()
string	trim ()
string [ ]	split ()
string	substring ()
~~string~~ int	compareTo( )
int	indexOf ( )
int	~~[illegible]~~ lastIndexOf () .
char	charAt()
string	replace()
string	valueOf()

Ques. string S = " Hello";
sout (S.length()); //5
sout (S.charAt (S.length() - 1)); //O

Output : 5
"O"

Ques. string S1 = "Hello";
string S2 = "Hello";
string S3 = "HELLO";
sout (S1. equals (S2)); // False
sout (S1 . equals (S3)); // False
sout (S1. equalsIgnoreCase (S3)); // True

```
sout (S1. compareTo (s2));        // -32.
        // (ASCII value)
```

Ques.
```
string S1 = "Hello";
S1=S1. concat("Hello");
sout (S1);                        // HelloHello.
```

Ques.
```
string S1 = "Hello";
S1. concat("Hi");
sout (S1);                        // Hello.
```

Ques.
```
string S1 = "    Hello    ";
```

leading space

trailing space.

Remove these using trim

```
sout (S1);
sout (S1. trim());
```

Output : "   Hello   "
"Hello".

Ques.
```
string s = "this is a string to test";
string[] r = S. split(" ");
sout (S. split(" "));  or  sout (r);
sout (r[0]
```

Output : ["this", "is", "a", "string", "to", "test"].

· this

Ques. string S1 = "this is a string test for testing";
string[] b = S1.split("e");
for (string a: r) {
sout (a);
}

Output : "this is a string "
"st for t "
"Sting "

Ques. string S1 = "this is a string test for testing

sout (S1. substring (5));
sout (S1. substring (5, 11));

• Output : "is a string test for testing"
"is a s"

sout (S1. Index of ("is")); // 2
sout (S1. Index of ("test")); // 17
sout (S1. last Index of ("test")); // 26

Ques. int a = 10;
int b = 20;

sout (a + b); // int + int = int.
string x = string. value of (a); // string
string y = string. value of (b); // x = "10"
// y = "20"

sout (x+y) // "10" + "20" = "1020"

Output : "1020"