Java

→ installation
→ Java is platform independent

Javac ⇒ Java compiler
Jdk ⇒ Java developemd kit

IDE integrated developemnt Environment

code

package is collection of multiple class.

```
impart java.lang.*;
impart java.util.*;

public class SameASFileName{
    public static void main(string args[]){
        System.out.println("Hello world");
    }
}
```

ClassName should be same as filename

Primitive datatypes in java

integers
- Byte 8 bits 1byte
- short 16bits 2byte
- int 32bits 4byte
- long 64bits 8byte

~~datatype~~ keywords

character → char – 2byte
(0 to 128 ascii)

7 digit after point

fraction /float/ double
- Float 32bits 4byte
- double 64bits 8byte

upto 15 digit after zero

Boolean → boolean 1bit
↓ Default value False
↑ true or false

default value for integer ⇒ 0 float = 0.0f double = 0.0d

`float a = 12.45f` → float for float you have to express (f)at end

⇒ variables not start with number or spacial characters
⇒ variables always have underscore if needed spacey
⇒ only $ can start variable name..

Arithmetical operators

+ (add) – (sub) * (mul) / (div)

% ⇒ modulus

with two int we can get integer division & also float division

n++ (increment) n-- decrement
↓ `n := n+1` `n = n-1`

`+=` `-=` `*=` `/=`

logical & bitwise operators

&& ⇒ logical and OR ⇒ || Not (!) (logical)

logical operators are used to sum up multiple conditional statements

(~ ⇒ Bitwise Compliment)

Bitwise operators

& Bitwise and | bitwise or ~~bitwise Not~~

~~10~~ 32 16

do 32 & 16

⇒ ~~48~~ 16

32 | 16 (or)

⇒ ~~48~~

010000 ⇒ 16

& 100000

& 000000 ⇒ 0

| 110000 ⇒ 48

⇒ Bitwise operation performed on Binary value of number

~16 ⇒ 01111 ⇒ ~~48~~ -17

Relational operators in java

~~also known~~ (==) equal to

= assignment operator

> greater than < less than

>= greater than or equal to

<= less than or equal to

!= not equal to

we can use them and then in many codes

If statement

If condition → result in True or false

anything which give output in true or false is condition

```
if (Condition) {
    If true Condition 1
else if (condition 2) {
    If condition 2 true
}
else {
    if Condition 1 &
    condition 2
    false
}
```

⇒ If else if ladder

Nested If statement

~~If else if ladder => that is nested else.~~

=> If else inside else if so that is nested if statement

For loop

for loop really goes to infinit loop

```
for (initialisation; condition ; increment/decrement) {
        code
}
```

we dont have to worry about inc/dec as it already in loop

there is also a for each can used for loops

while loop

we use while loop if we dont know when to stop

```
while (condition) {
    code
}
```

thus how you can write the code and see how can we will able to

→ don't forgot increment/decrement in while loop, heavy chances of infinite loop

nested for

```
for ( ; ; ) {
    for ( ; ; ) {
    }
}
```

nested while

```
while () {
    while () {
    }
}
```

=> loop inside loop is nested loop

Do while loop

```
Do {
    code
} while (condition);
```

runs atleast one time

Ex game, a Player play atleast once & ask retry or not at start

⇓ useful some time,

loop break Used to break loop which is running at a particular time

⇒ continue is used to skip a iteration in loop
⇒ all following code of continue will skipped

⊕ String

String a = "zabc"; String b = "abc"

String c = a+b ; string concatnation

String d = ~~c~~ a.concat(b)

⇒ concat function or ⊕ both can be used to concatnation
→ If any operator is string that will do concatnation

a.length() ⇒ give length of string
index start with 0

Check string length → return ↑ if >0

isEmpty() to check is string empty

isBlank()

↘ check is characters is there in string or not

⇒ trim used remove spaces.
⇒ toUpperCase()

⊛ Compare two string using

case should same ← str1.equals(str2) ⇒ boolean output

str1 equalIgnoreCase(b) ⇒ compare string not by case by character

⇒ CompareTO(first, second)
↑ If string First big give positive, second is big give -ve & 0 if same

⇒ CompareTo IgnoreCase (first, second)
→ compare string without taking case into consider

⇒ Used to find bigger/smaller string among two

Matches

It is used to check a string with Regular expression.

say str a = "[a-z]" ⇒ so if all character in string are from a-z to z return true

→ matches used to see pattern in string

→ If pattern match ⇒ true else will true

String search

str1.contains(str2) → so if str2 is there in str1 return true else false

↑ case sensitive

to check string is startwith some specific word or word character

str1.startWith(str2) — if str1 startwith str2 return true else false

str.endswith(str2 — if str1 endwith str2 returns true else false

→ return first occurance

⇒ indexof ⇒ used to find index of character of substring

⇒ lastIndexof ⇒ find character or substring from ~~end~~ Search start from end

↑ Search string from reverse order.

String slicing

a.subsequence(start, end)

a.substring(start, end)

Substring used to slicing of string in java

charAt() ⇒ return character at specific index in string → give error outofBoundary!

⇒ split

"hell".substring(1) ⇒ "ell" → only find index

say a = "a_b_c_d_e"

we want to split string on basis of some "_"

a.split("_") return a list

output will be array

String arr will o/p
⇒ you use split to paragraph in words. #Useful

Replace used to replace a character or word
Replace ⇒ use normal string
ReplaceAll → Search for all Regex expression
→ you can do anything replaced in string using these functions
→ with replace you can remove also
- "123gs456".replaceAll("[0-9]", "")
⇒ Then all numbers will removed

string conversion

Integer.toString() convert no → String
Integer.toBinaryString() find binary value of number in String format
Integer.toOctalString() find octal value of number in String form
Integer.parseInt() → find hexadecimal in string
↑
Integer.parseInt("1111", number system)
→ parseInt used to convert ↑ 2, 8, 16.
bi's number String to specific number type

("1111", 2) mean
1111 is binary convert it into dec

parse return value of number string in dec

→ using those functions we can manipulate both number types.

classes & objects

class is blueprint of objects

has → variables (data member)
→ functions member functions

```
class ClassName{
  // code
}
```

→ creating new object for that we need new keyword

obj `ClassName a = new ClassName()`

such like that you can create obj → such like that you can define a class

⊙ Operator (dot) to access datamembers & functions

`a.variableValue = "change"; a.func()`

⇒ you can also create array of objects also

⇒ method overloading ⇒ same name of method but vary in parameter

⇒ method overriding ⇒ say function is declared in class Parent so we will override it using method overriding that is redeclare function in child class,

⇒ abstract function ~~are ba~~ & interfaces create such condition so for inheriting or implementing them we have to do method overriding

---

methods

declaration returntype name (para comma seprated) {

```
{
  Code
}
```

we can call a single method using function call

name() ⇒ function call

void ⇒ function which not return anything

returntype

int, byte, short, double, String, Integer etc

They return some value

multiple function of same name but separated with [type & count of parameter]

⇒ see code / do code

⇒ return keyword used to return a value

Access specifiers → main part to handle

Ⓐ Public
Ⓑ Private
Ⓒ Default / No access specifier
Ⓓ Protected

⇒ Public

↓ ✱

you can access public default outside class and also a file from same package

Private can only accessed inside same class & not outside

→ you can't access in same package also

in

✱ ⇒ you can call public class object ~~outside the~~ file outside the package but you can't access default in file outside package

	Public	Protected	Private	Default
within class	yes	yes	yes	NO
within package	yes	yes	NO	yes
Same package by subclass	yes	yes	NO	yes
outside pack by subclass	yes	yes	NO	NO

Private → only within class

Default ⇒ anywhere in self package

Public anywhere

Protected anywhere with inheritance

Constructor in Java

⇒ this is function which get initialised automatically when object of a class created
⇒ have same name as class
⇒ If you not create constructor default constructor will called
→ constructor is not a member of class (Imp)
→ can have multiple constructor for a same class
⇒ Constructors are overloaded (change in parameter (no or type) & thus how we can create any no. of constructor
↳ constructor overloading

Data Encapsulation

OOPS Basic → Encapsulation, polymorphism, abstraction, Inheritance } Basic principal of OOP

→ Encapsulation means hiding data from another class
→ In this we use getters & setters so that one who using our class will not able to access our parameter direct
set ⇒ this is function which set value for data in class without directly accessing the
get ⇒ this function which get value for data in class without direct access to class
⇒ so we will set our data member private

using getters & setters to provide indirect access that is data encapsulation

Static keyword

Ex class hell {
Static int a = 10;
}
Static func() {
Code
}

⇒ the static data member, or function can directly access by using class Name

you can access a & func() by
hell.a hell.func()
ClassName. function()

Static variable are static functions are universal for class and not belong to specific object

```
class hell {
    static int b = 10;
}
```

by this →

```
hell a = new hell()
hell b = new hell()
a.b = 100
println(b.b) => output will 100
```

as static variables are unique to class

⇒ Static data belong to class & unique to all class

Main func

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

- access specifier
- return type / func name
- main func unique to class
- args ↑ Command line

⇒ JVM ⇒ Java virtual machine → search for main func to run java program

Java is platform independent

⇒ In main can any func classes & it is also entry point to enter in Java program

⇒ JVM provide runtime environment to run code in java

⇒ you can run .class file from any system to any other System

⇒ Public is used as JVM is outside packet & want to access fun so it's access specifier is Public

Java abc.class | 1 | 2 | 3 | 4 |

↑ Command line args

⇒ If JVM is unable to find main we get NoSuchMethodError: main & Program will end

Math class In java

This is library which provide large no. of methods to us.
→ All function in math class are static

* function in Math class
  math.min(a,b) → find minimum bet no
  math.max(a,b) → find max among no
* math.PI ⇒ also store constants like PI
  math.E ⇒ value of E
* use pow function to find power of no
  Math.pow(nuber, raisedto)
* math.sqrt() → find square root of a numbr
  mathicbrt() ⇒ find cube root
* math.ceil(n) ⇒ ~~convert & round up number~~
  push to next full into next ~~re~~ integer
  using ceil function

Ex: 10.2 ⇒ 11
10.1 ⇒ 11

math.floor(n) ⇒ convert no, remove decimal value & give raw input

Ex: 10.8 ⇒ 10
11.9 ⇒ 11

do code using ceil, floor & Round fun

math.Round(n) ⇒ round up no

Ex: 10.8 ⇒ 11
110.2 ⇒ 110

→ using round function

⇒ also you can find sin, cos tan values als
math.sin(val), math.cos(val) math.tan(v) ⇒ Such like that
↳ Provide value in radian

⇒ Math.random ⇒ largely used to Create & instantiate random numbers (we also have seprate random class)

Math.random() → this will give ~~integer~~ value bet 0 to 1

StringBuilder In Java strings are immutable

→ the string object created in memory they are immutable

```
String a = "I ";
a = a + "love";
a = a + " Java";
```

→ here java at each stage creating new string & give it to reference

so in memory

I	love	I love	Java	I love Java

⇒ these 5 lies in Java memory

⇒ so it is not good way & memory consuming so to avoid building new string in memory we have StringBuilder function

```
StringBuilder a = new StringBuilder("I");
a.append("love");
a.append("Java");
```

so now output will same but it will not create 5 different object for each class

⊛ methods of StringBuilder

(#) End index is exclusive not inclusive

It has few fun^n like insert, replace, delete reverse, append

try them in Code

→ StringBuilder is a class from java.lang

⇒ you can also use function on string or stringBuilder class

⇒ k.replace (startindex, lastindex, string want to insert)

7 → 10 that mean (789) → will replaced

⇒ k.delete (startindex, lastindex+1) → used to delete string

⇒ k.reverse ; will reverse string.

⇒ k.insert (startindex, string want to insert) → insert substring in string

↑ you can insert string → index at which we insert

Such like that you can use fun^n in StringBuilder class

Scanner object (take user input)

used to take input from user for that we can use Scanner class
It has large no. of function & you use it according to your need
Ex: nextInt(), nextBoolean() & many more

next() ⇒ take input as string (only world as I/o)
nextline() ⇒ take a whole line as I/o

⇒ It is present in java.util so you have to import it specifically

⇒ Java use compiler

Random class in java

a default class in java.utils

Random a = new Random()

a.nextInt() ⇒ to get random integer value
a.nextInt(1000) → will get no betn 0 to 1000

⇒ you also can able to create no betn some range using these fun

---

UUID class → (universally unique identifier)

It is used to create unique identifiers

UUID.randomUUID() → will always return some unique value (128 bit value containing character & alphanumeric

⇒ can used for many purpose Ex gift card, product key, etc

↑ present in java.utils

JDK 10 → recently recent version of Java ⇒ JDK-19

Var keyword

we can declare variable in java using var

var a = 100; so not a has set to be int so you can't change it's type if you a = "hello"; this will raised error

⇒ once initialisation of value Jdk automatically assign type & you can't change it further

```
var a = "hello";
var b = 10.8;
```

you can give int value to float variable

you can do it

```
var a = 10.89;
a = 11
a = 11890
```

⇒ you can also declare object with var

```
var input = new Scanner(System.in);
```

⇒ You can catch output of functions using (var) (noisseee)

⇒ Var can prove usefull

⇒ you can declare array

```
var m = new int[5]
```

→ here you don't have to mention []

also `int a[] = {2,3,4,5};`

But this case you can't do it with var

(X) `var a = {1,2,5}` ⇒ error

⇒ you can't create var keyword variable in class as data member

⇒ you also can't use var as return type of funct

also not pass var as argument in Java fun

`var fn()` ⇒ not allowed

Garbage Collector in Java 10

Garbage collector ⇒ iterate over object which are not in use

⇒ so lot of unused object exist in JVM so garbage collector ~~create~~ clears the the garbage value which not used by user to efficient utilisation of memory

⇒ You can create own garbage collector
⇒ the garbage collector is an interface → interface

you can implement using Class

(*) Imp

⊕ Inheritance & abstraction

inheritance ⇒ used for reuse of code
⇒ can transfer feature from one class to another class
⇒ we use keyword [extends] to create or extend that class

keyword : extends

class One{
}

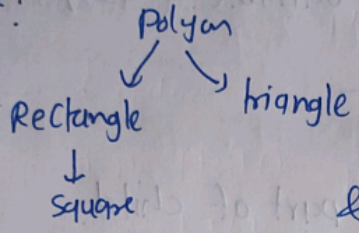
class Two extends One{
}

Two → child, subclass
One → Parent, superclass

~~One~~ → child
subclass
superclass

→ Two get inherited by ~~to~~ function & data from One
→ we can use function (features) in two which implanted in one

② Ex.



extend create "is a" relation
ie Rectangle is a Polygon
& not case that reverse is true

⇒ that is "subclass" is a "Parentclass"

---

① ~~same~~ function declaration of same name in same class multiple times ⇒ method overloading

② same function ~~both~~ declared in parent & also in child so that fun ⇒ overriding functions

```
class Polygon{
        int a;
void displayLength(){
    println(a)
}
}
class Rectangle extends Polygon{
    int c
void displayLength(){
    print(c)
}
}
```

⇒ Polygon b = new Polygon()
⇒ Rectangle g = new Rectangle()
⇒ ~~Rectangle b = new ...~~

b.displayLength() Parent
g.displayLength() child

⇒ If child redeclare function inside parent that is function overriding

~~⇒ in overloading~~

data members
⇒ legs, eyes
⇒ Voice("")
⇒ meal("")

data mapping
~~function~~
funⁿ

Animal
- Carnivore
  - dog
  - Cat
- herbivore
  - deer

- Polymorphism ⇒ multiple forms but same name
  - overloading
  - overriding

⇒ with overloading & overriding we acquire polymorphism

In java we can declare parent variable & assign value of child class

Parent a = new child();

↓ no a will point to parent part of child

⇒ also ~~you can see polymorphism parent variable are alive in holding child's form.~~

⇒ a.overrided function() ⇒ it call function in child as it referenced to child

Imp: for overrided funⁿ even for parent refered child object function called in child body

⇒ reverse is not true

Child a = new Animal(); → error

---

with parent reference you can't access data members from child

Imp

```
Parent {
}
child {
  val : 100
}
```

Parent c = new child()

c.val ⇒ error

Imp

With parent we can only access datamembers in parent & child but we can't access child data member

Super Keyword (do code will lean easily)

It is used by subclass to access property & function of superclass

⇒ with super keyword we can fetch overrided function from parent

⇒ also if you have same name of variable in child & parent

this.var1 ⇒ child this.~~var2~~ super.var2 ⇒ Parent

⇒ do code it is very easy to use

---

with super you can pass value from child constructor to parent constructor

for that you have super() ⇒ function

child.e() {

super(val1, val2)

}

this ⇒ this class obj

super ⇒ parent

⇒ you can only use super function in child constructor

⇒ you can't use super keyword outside the class

⇒ else you can use super keyword around class

Protected access ⇒ If we want pass value from parent to child we use protected access specifier (usefull)

Abstraction ⇒ this is process of hiding implementation details from user and provide only function

↳ user will directly use function without knowing about the how they are implented.

↳ data structure are also abstract

abstract class

Provide abstraction

For abstraction we use abstract ~~not~~ class

these classe can acts as parent but can't able to create any object

⇒ you can't access abstract class

⇒ abstract class refernce can hold child object
⇒ abstract class contains abstract functions which should be overrided by child class
⇒ you can write constructor for abstract class.
⇒ It is easy to use abstract class ☺
⇒ with abstract classes the can efficiently hide data using protected keywords
⇒ the abstract function don't have fun body

abstract class are like contract so if you inherit abstract class you have to implement abstract functions

→ abstract class can also have non abstract method which need to be implemented in child class

Interface

→ It is collection of abstract methods
→ we can implement class over interface using keyword implements
⇒ interface can implement another interface
⇒ non abstract class which implementing interface need to implement unimplemented methods
⇒ we can't create object for interface
⇒ Interface dont have constructor abstract class do
⇒ methods in interface are **Static & default**
⇒ All variables are by default final & static → cant change value further
⇒ interface don't have data members
⇒ you have to assign value to interface variable at time of declaration & cant be changed further

⇒ final function / Keyword variable cant be reinitialized or overrided

⇒ All functions are by default abstract in interface so you don't have to implement it:

⇒ to create interface from another interface we extend

interface extends interface

you also have to maintain access hierarchy in interface

class → class → extends

~~class~~ interface → interface → extends

interface → class ⇒ not possible

class implement → interface → implements

Imp

you can use interface as reference for it's child

So say

interface one { void f1() }

interface two extends one { void f2() }

Implements

class zz ~~extends~~ implements two { }

just this thing in mind & implementation is easy

Imp

So here you have implement abstract fun from interface one or interface 2

② you also have to maintain weaker Restriction

private
default
protected
public

less Restriction

So you have a fun in interface so default

→ So while implementing it try to use public / protected

③ you can't override final fun

you can't extend create a class having multiple parent but a class can have multiple interface as its parent

Arrays in Java — Collection of similar elements (datatype)

String [] dayofweek = {"mon","Tue","Thu"}

⇒ index start from 0

dayofweek[index] ⇒ used to fetch value from that index in dayofweek array

⇒ dayofweek.length ⇒ this will give length of array

⇒ also can update value also.

⇒ Array length is fixed & given at time of initialisation

⇒ If you give large index > length that give error

⇒ you can use loop to iterate through loop.

⇒ you can create array of any datatype

⇒ Java array contains all element of same datatype

you can create array of classes also

Array object

we can use new keyword to create array object

int arr[] = new int (size) ← this will be size of array

↑ name of array

⇒ all value will be 0

---

2) Iterate loop over array

int j[] = {10, 20, 30, 40, 50, 60, 70}

for loop

```
for (int i=0 ; i<j.length; j++){
    println ( j[i]
}
```

for each loop

```
for (int x : j ]{
    Println(x)
}
```

⇒ Both will print all value in array in output

2D array in Java — array of arrays

```
String arr[][] = { {'a','b','c'}, {'x','y','z'}, {'k','l','p','d'} }
```
This is how you can make 2D array

(easy) a[1][2] ⇒ two index need to access a block in 2D array

* In java you can create ~~object of~~ class array and that can handle multiple data type & it is easy to use

```
class hell{
}
```

```
hell a[] = new hell[10]
```
a is hell class array of size 10

Regular Expression → we can define search pattern

```
.stringmatches("pattern")
```
it can be anything

⇒ Regex can create pattern & make our life easy

⇒ you can use Regex in many operations related to string

Quantifier in Regex (See code in folder)

* →	match 0 or more than instance of preceding char
+ →	match 1 or more than instance of preceding chara
? →	occur once or not at all (0 or 1)
X{n} ⇒	occurs n time (x occu
X(n,) ⇒	occur n time or greater than it
X(n,m) ⇒	occurs at least n time & not more than m

① abc* ⇒ ab, abc, abbc, abccc, etc

② abc+ ⇒ abc, abc, abcc, abccc, etc

abc? ⇒ abc only

⇒ you can block character in (bracket

"ab(xyz)*) ⇒ ab abxyz, abxyzxyz etc

Pattern

↑ all element should three

( ) ⇒ block of characters

→ xyz should repeat 3 time

{n}

"a b (xyz) {3}"

so ↑

true

abc xyxyzxyz

+ xy should be happened only 3 time

* ⇒ 0 or more + ⇒ 1 or more

{ } ⇒ 0 or 1

character classes

\d ⇒ matches any digit → not any alphabet

\w ⇒ matches any single word character

\s ⇒ match for spaces

\D ⇒ non digit character

\W ⇒ non letter character

\d for \ escape sequence

use \\d

↓ in code

complement

\d complement \D

\w complement \W

W ⇒ can used to represent special characters

so phone in ind can be

[978]\\d{9} → any indian no

Regex is very helpfull tool for pattern matching stuff

[ ]

⇒ Bracket expression

"[abcd]" → a or b or c or d

also range ⇒ [a-zA-Z] this is also valid

a to z or A to z → pattern

⇒ [hrstbm]at

so hat, rat, sat, tat, bat all are legal

so for all alphanumeric characters → [a-zA-Z0-9] any alphanumeric

⇒ All you can provide custom range

(afg) → as block

^ it is used in bracket that mean not [afg] → a or F or g

[^a-zA-Z0-9] ⇒ so any non alphanumeric character can be act as true and

^ ⇒ exclude

[-^Az-I] → anything but capital letter

⊛ OR operator

| ⇒ pipe sign

a(b|c) → ab or ac

ab(c|d|e|f) → abc, abd, abe, abf

↑ it act like or

(b|c) ⇒ [bc] similar to

⊙ dot operator

. ⇒ any character

Say you want to find cat

[c.t] & cat will be matched

⇒ placeholder ~~character~~ char

ABC ⇒ at position of dot there can be any character possible

.* ⇒ mean anything in further

abc.* ⇒ so acbc abct abt

(.*) ⇒ greedy match

(.*) → we adding . & * both property any any character can be match

⊛ greedy match & lazy match

** → greedy match

*+ greedy

*{} ⇒ greedy

+? —
*? — ⇒ non greedy matches
(){}

⇒ have stoppage

---

greedy mean search undefinitely will any possibility so it can acquire any char (no stopage)

⇒ the pattern which has finite set of possibilities
⇒ greedy non-greedy

⇒ the pattern which go to infinite set of possibilities
non-greedy

⇒ \. ⇒ . ⇒ for (.) dot you have to use a escape sequence

⇒ we can use matcher class to retrieve patter from string
in array format
matcher & pattern class used for it

---

⑧ Except handling

Exception ⇒ is the event which disrupt normal execution of Programme Ex: StringoutofBound, Divide by zero.

⇒ So we can avoid and handle such exception using exception handler.

⇒ JVM try to see where that exception happened & is there any exception handler is there or not

⇒ When exception is happen it is thrown & some exception handler catchy it. & do work.

Error vs Exception — we cant handle error but we can handle the exception in the programme

---

Checked exception ⇒ these exception which checked & thrown at compile time is checked exception
Ex: a file location

⇒ Compiler never able to caught exception which occurs at runtime

⇒ compile can only goab exceptions at compiletime exception

checked Exception — we must have to handle this

⇒ unchecked exception: Ex: a division & you pass I/O 0
at denominator obviously Programme

will throw error as divison by zero is forbidden

Checked exceptions ⇒ Checked at compile time

unchecked exception ⇒ checked at run time

throw declaration

we can throw exception or we can use try-catch block.

→ firs learn about throws.

the function throws exception (throws keyword)

```
public void abc() throws ExceptionWhichFunctionThrows {
    code → this code will happen to have this exception
}
```

⇒ Also there exist a family which can handle an exception

Ex: FileNotFound Exception

So

Throwable → Exception → IOException → FileNotFoundException

so any one of above can handle that fileNot found exception

→ Its on us which to use more specific or more generic

⇒ A function should either throw except away or handle it

- for throwing exception ⇒ throws & to be handled by someone else
- to handle exception ⇒ try-catch- finally block used

```
try {
    code which throw exception;
}
catch (typeofException a) {
    code to execute if you tackle some exception
}
```

you can have multiple exception So you can have multiple catch

```
try {
} catch(Exception a) {
    ||Code||
} Catch(Exception b) {
    ||Code||
}
```

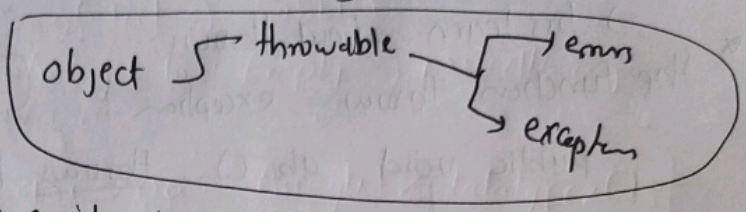
⊙ You can also used nested try catch

```
try {
    code
    try {
        code
    }
    catch( ) {}
}
catch {
    code
}
```

→ in catch we can provide exception

⇒ nested try catch block

⇒ nested try catch is more useful than multiple catch blocks



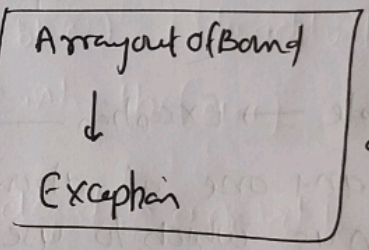
→ There is build in class exist in Java called Exception

→ you can create own exception by extending Exception class

⇒ In try block line after which exception occurs will never run if exception arised

⇒ while putting catch block order should be

more specific ↓ more General



Such like that

Generic

⇒ finally block

⇒ Finally block runs ~~even~~ though Exception arises or not

```
try {
}catch(){}
finally{}
```

Case

try → ~~catch~~ → finally

If exception arises →

try → finally ⇒ If exception not arises

⇒ finally block is optional

⇒ You can also use try-finally block

Date & time operations in Java (Java 8)

for that we can use localeDate, LocalTime

function :- LocalDate, LocalDateTime, LocalTime

now() function are there in all three which gives current details related to time

for Date : year-month-date ⇒ format

for time : hours:m:seconds ⇒ format

for date time : year-month-date T hour:min:sec

T is separator for date & time

this will remain same

① Create date

LocalDate ab = localDate.of(year, month, day)

month ⇒ is enum (month.January) you can do here

Both will give same o/p than Ex 2022-11-13 E

② LocalTime : set time (by each of function in these function is overloaded so you can use anything

LocalTime b = LocalTime.of(hour, min, sec, nanosec)

& for

localDateTime a = LocalDateTime.of(ab, b)

So you can put date time directly

Date we can find future dates

Println(LocalDate.now().plusMonth() → output

⇒ minusmonth(3) ⇒ plusYear() ⇒ plusDay()

⇒ useful

for time Difference ⇒ Duration

for date difference ⇒ Period

Arraylist → it is list (<> can't use primitive datatype use class)

→ in array we can't change size of array but in Arraylist we can

→ Arraylist supports dynamic array creation

methods:
- Accessing element ⇒ Get, Add, update, remove, clear
- Searching ⇒ indexof, contains
- viewing portion of list ⇒ SubList
- ⓐ SIze, toArray, isEmpty ⇒ miscellaneous

`Arraylist<datatype> arr = new Arraylist<datatype>();`

↑ It is present in java.utils pkg → list is abstract class for Arraylist

or you can use list

~~`list<datatype> arr = new list<datatype>();`~~

`arr.add("red"); arr.add("blue");` → to add value

`println(arr);` → directly prints the arr

→ `arr.get(index);` → get the element in arr at that index.

Basic list operations

⇒ `arr.set(index,value)` ⇒ to change value at a position in list

⇒ list can be used to have duplicate value

-) `arr.clear()` ⇒ remove all elements in list

⇒ `arr.remove(index)` ⇒ remove element at specific index

⇒ `arr.remove("blue")` ⇒ remove element

You can use remove using index or value

these are ~~array addition + array~~ Arraylist related topic functions

Eg: `list<datatype> arr = new Arraylist<String>();`

In java one value can ~~mor~~ reference by multiple refr

```
int a = 10;
int b = a;
```

a → (10) ← b

Such case happen not like c where a & b are differ

⇒ a & b are pointer itself in java.

Search operation indexOf() → return index of a value
contains (val) if that val present in list return true else false

return -1 ⇒ if not

You can use foreach, for loop & other loops easily on arraylist

size() ⇒ return length of arraylist

subList (start, end) → end is not inclusive

It is used to get sublist from a list

isEmpty() → true / false — Check that list is filled or empty
Boolean o/p

toArray()

⇒ used to convert a list /arraylist to array

---

for arraylist you can't give primitive datatype inside

```
Arraylist<int> arr = new Arraylist<int>();
```

↑ this will give error

use object class like Integer, String, Character, Float & that will work

---

listIterator is the which object to hold list & next() used to increment count

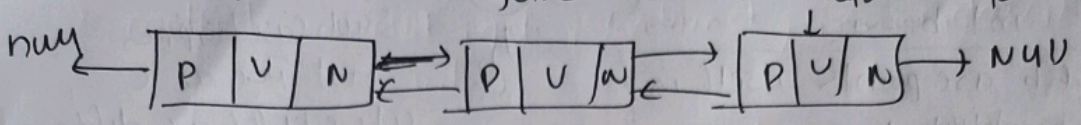
this type should match with Arraylist

```
listIterator<String> a = ~~Name~~ ArraylistName.listIterator()
```

```
while( ~~name~~ a.hasNext() → return bool
println( ~~xxx~~ a.next() );
```

linkedlist

→ linked list elements stored at random location & joined together using pointers



Each block has value
→ a forward pointer ⇒ next
→ a backward pointer ⇒ previous

linked list in java / internally linked list it implement doublylinkedlist

linkedlist → add(3,2) add(index,val)

All functions in list can be applied to linked list

specific fun^n used only to linked list

peek() ⇒ give value of head

In linked list add will add value at

iterator two main fun hasNext() next()

& ~~Iterator~~

```
ListIterator<datatype> a = yourArraylist.iterator();
```

⊙ Linkedlist vs arraylist

⇒ addfunctions is different among those two

⇒ linkedlist has addn fun

Poll functions ⇒ pollhead, pollfirst, pollLast → remove them & return val

peek give value of them → only give val

peekhead, peekfirst, peekLast

pollFirst()
pollLast()
peekFirst()
peekLast() ⇒ main fun^n

Hash set (sets in java) → (we can do that in list)

→ set do not have duplicates

→ Do not maintain insertion order (no index access)

method: add, remove, clear, contains size

IMP: union, intersection

⇒ can able to convert list into sets

→ set is abstract class for hashset so you can use it as reference

Set1.addAll(set2) → this used to find union of two sets

Set1.retainAll(set2) ⇒ this will find intersection among two

you can use for each loop to iterate through set, for loop is not that useful here

to convert set into list

```
var list = new ArrayList<Integer>();
list.addAll(set1);
```
⇒ So set1 will converted to list

* to creat linked hash set it use same funcn as hashset

```
LinkedHashSet<Character> n = new LinkedHashSet<Character>();
```

↑ easy way to find creat linked hashset

↑↑ all fun are same

↓ only difference is linkedhashset maintain insertion order

Tree set

It is sorted set

the elemt which will fill get sorted in ascending order

all methods are same as of Hashset

Iterating over set

use also you can use iterator to access set ⇒ use **Iterator class**

(#) Hashmap It store data in key value pair

→ we can access element using corresponding keys

→ methods → put, get, containsKey, remove, clear, keySet, values

---

Treemap ⇒ values are sorted