Static	Compile Time	Overloading	Same Class
Dynamic	Run Time	Overriding	Multiple Classes

(Types of Polymorphism) MFL

Showing (what is being done) functionality
Hiding (How it is done) implementation

Binding / Wrapping the data & the code into a block (Class) so that the data is not accidentally modified, is called Encapsulation.

We use the "private" access modifier to achieve encapsulation.

To access outside the class, we use two special "public" methods:
① Retrieve → getters ② set → setters

Abstraction & Its Types:

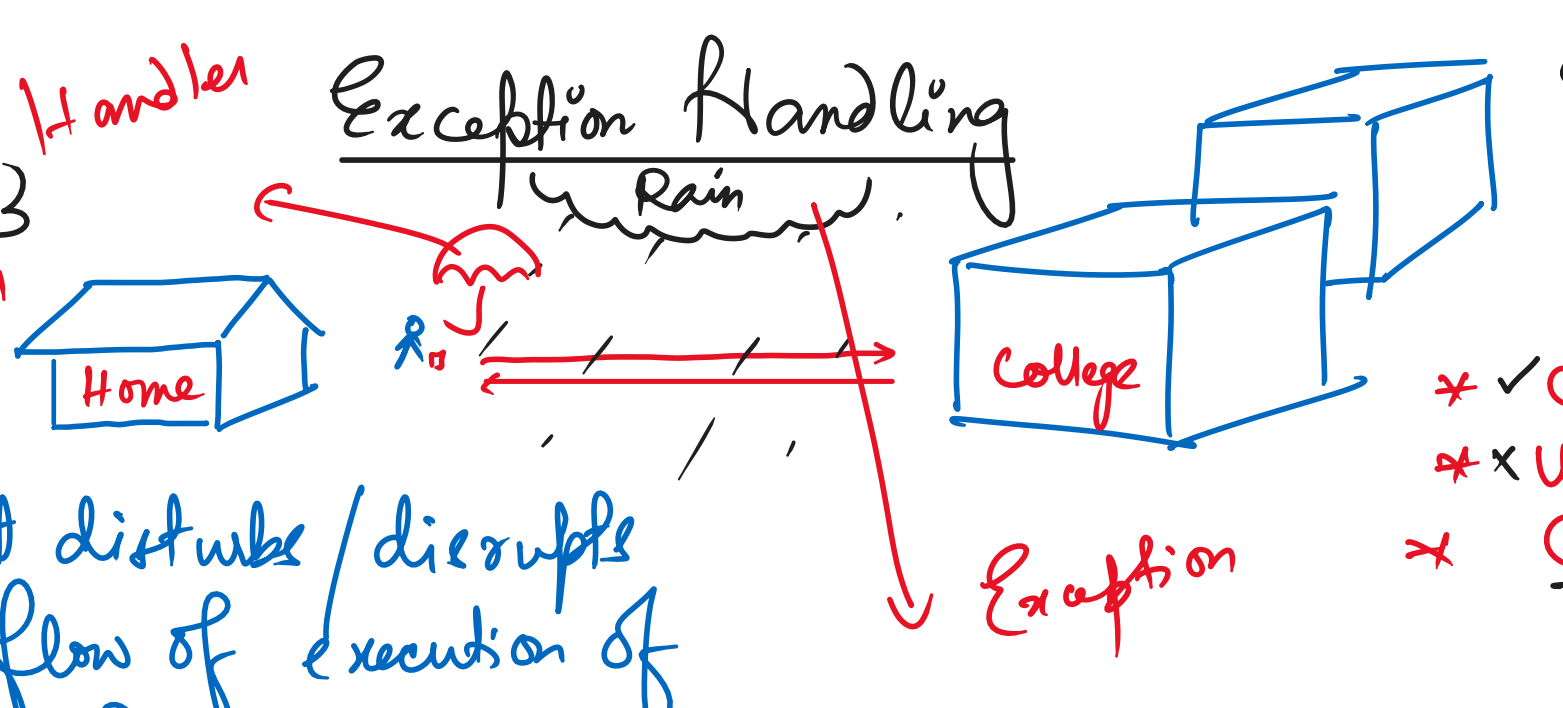
① Abstract Classes & Methods (0 to 100 %)

② Interfaces (100 %)

Exception Handling

Error : CT / RT → Can't be handled — 32GB RGD OOM

Handler



Exception Can be Handled
* ✓ Checked
* × Unchecked
* Custom or UDE

Anything that disturbs / disrupts the normal flow of execution of code, is called exception.

The process of handling exceptions is called Exception Handling.

Object ← Throwable → package → java.lang.Throwable

Throwable → Errors, Exceptions

NaN

Important Keywords: try, catch, finally, throw, throws, Throwable. (Java)

UDE ↑ throw

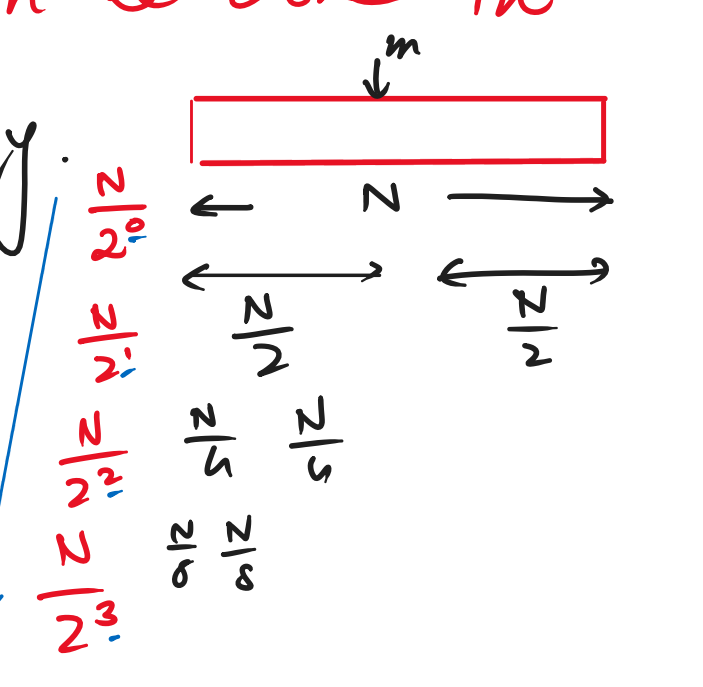
* All your resources are terminated / closed in the finally block.
* This block always gets executed no matter what happens to the code.

⇒ Heap Data Structure ②

* int[] arr = { 8, 2, 6, 9, 7, 4 } ⇒ (2, 4, 6, 7, 8, 9) (indices 0 1 2 3 4 5)

$m = s + \left(\frac{e-s}{2}\right)$ $\quad \frac{5}{2} = 2$

Normal Binary Search for a key can be done in Big O (log N). time complexity.



$\frac{N}{2^0}$, $\frac{N}{2^1}$, $\frac{N}{2^2}$, $\frac{N}{2^3}$

2^K — K is 0, 1, 2, 3, 4, etc

$(K = \log_2 N)$

Big O $K = \log N$

Heap → (Complete Binary Tree) (Array) Recursion

arr | 2 | 8 | 6 | 9 | 7 | 4 |

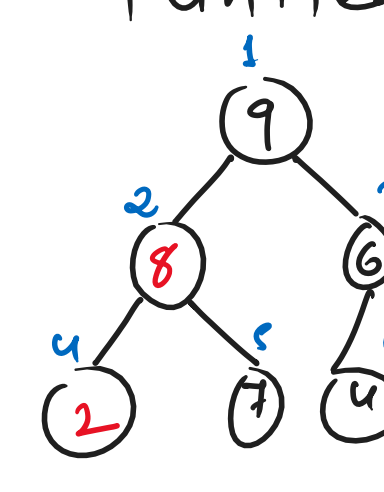
Max Heap → (L < Parent > R)
Min Heap → (L > Parent < R)

parent = i
✓ left child index, lci = 2*i
✓ right child index, rci = 2*i+1
✓ parent index = i/2

Sentinel value

-1	9	8	6	2	7	4
0	1	2	3	4	5	6

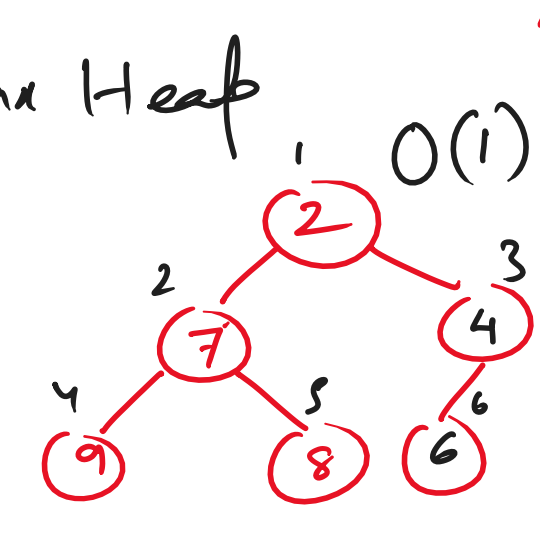
Max Heap



-1	2	7	4	9	8	6
0	1	2	3	4	5	6

Minheap

O(1)

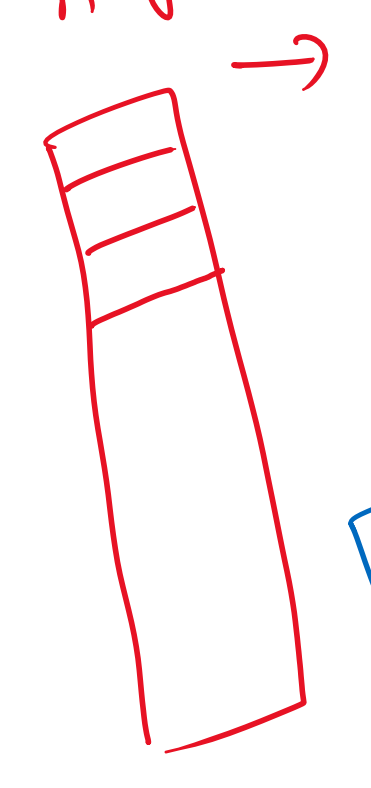


Find the (K Largest) Elements in an Array:

⇒ { 8, 9, 2, 7, 6, 4, 1 } k = ③

nlogn → sort (1, 2, 4, 6, 7, 8, 9)
n log n + O(n)

(Min Heap:)



100 elements (log k)
min Heap : size == 3

7 6 5 4 3

~~1~~ ~~2~~ ~~4~~ ~~6~~ 7 8 9 → O(1) O(n)

Find the kth (smallest) in an array: k = 3

{ 3, 4, 7, 9, 8, 10 }
1 2 3

(max heap)
10 ×
9 ×
8 ×
7 → top
4
3

max heap . peek ( )
poll ( )

* Greedy Algorithms : →

(LeetCode, Coding Ninjas, GFG)

* Minimum Number of Coins * (Sliding Window Problem)
* Chocolate Distribution Problem / (Minimum Absolute Distance)
* Minimum Cost of Connecting Ropes *
* Gas Station Problem
* Minimum Number of Platforms *
* Fractional Knapsack
* 0 - 1 Knapsack
* Activity Selection Problem *
* Job Scheduling Problem
* Huffman Encoding
* Lemonade Change Problem *
* Minimum Number of Arrows to Burst All Balloons
* Nikunj & Donuts

* Minimum number of coins to reach a certain value:

coins[ ] = { 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 };

V = 91	50, 20, 20, 1	= 4 coins
V = 31	20, 10, 1	= 3 coins
V = 10	10	= 1 coin

91 >= 50
91 - 50
41 >= 50
41 - 20
21 - 20
1 - 1
0

```
for (int i = coins.length -1; i>=0, i--) {
    if ( V >= coins[i]) {
        V -= coins[i];
}   }
```

[50, 20, 20, 1] (.size)

Lemonade Change Problem : bills 5, 10, 20 Price ₹ 5

Input 1: 5, 5, 5, 10, 20 — True
Input 2: 5, 5, 10, 10, 20 false

Greedy:
₹20 → 5, 5, 5
10 - 5 → 10, 5 greedy
5 —

bill	₹ 5	₹ 10
5	1	0
5	2	0
5	3	0
10	2	1
20	1	0
5	1	0
5	2	0
10	1	1
10	0	2
20		

(Polymorphism + Abstraction + Encapsulation) Pending topics
this & super keyword
Exception Handling
Heaps → Max + Min + Problems + Priority Queue
Greedy Algos

Pending → Trie + DP + BackTracking + Graphs