# Lambda Expression

→ One-line or inline functions → code will be a one line anonymous function
  ↳ no name, no modifier, no return type, no datatypes of parameter

→ Benefits :→ (1) reduce code lines, makes code readable/maintainable/concise
(2) Sequential & parallel execution support → passing functions as parameters
(3) calls API very efficiently

→ Inbuilt Functions using Lambda Expression ⇒ forEach method

→ Custom Lamba Expression
  → Priority Queue → Comparable
                   → Comparator
  → Arrays.sort or Collections.sort → Reverse Order
                                    → Custom order (Pairs)

```
public String concatenate(String a, String b) {
    return a+b;
}
```

⇓ Type Infer at Compile time

```
(a, b) → {
    return a+b;
}
```

```
Run | Debug
public static void main(String[] args) {
    int[] arr = { 10, 20, 30, 40, 50 };

    // FOR EACH LOOP (Iterable)
    for (int val : arr) {
        System.out.print(val + " ");
    }
    System.out.println();

    // FOR EACH METHOD
    ArrayList<Integer> al = new ArrayList<>();
    al.add(e: 10);
    al.add(e: 20);
    al.add(e: 30);
    al.add(e: 40);

    al.forEach((val) -> System.out.print(val + " "));
    System.out.println();
}
```

anonymous function or arrow func = lambda Expression

Syntax ②

Curly braces can be omitted only if one line instruction is there

Syntax ①

```
al.forEach((val) -> {
    System.out.print(val + " ");
});
System.out.println();
```

```
al.forEach((val) -> {
    if (val % 2 == 0)
        System.out.print(s: "Even ");
    else
        System.out.print(s: "Odd ");
});
```

## Comparator (many)

```
public int compare(Object a, Object b){
    (1) return a.val - b.val;
    (2) return b.val - a.val
}
```

(1) → a - b = -ve → smaller will be placed first → ⇒ Increasing order or min Heap

(2) → b - a = +ve → larger will be placed first → ⇒ Decreasing order or max Heap

## Comparable (Unique)

→ Same class whose objects to be compared

→ By default sorting

```
public int compareTo(Object other){
    (1) return this.val - other.val;
    (2) return other.val - this.val;
}
```

(1) → Increasing order

(2) → Decreasing order

# Arrays.sort

```java
public static void customLambdaExpression() {
    int[] arr = { 50, 30, 80, 90, 10, 20, 70, 40, 100, 60 };
    Arrays.sort(arr);
    // Arrays.sort(arr, comparator); INVALID FOR PRIMITIVES

    // Increasing Order : Default
    // System.out.println(arr); // HashCode

    for (int val : arr)
        System.out.print(val + " ");
    System.out.println();

    Integer[] copy = new Integer[arr.length];
    for (int idx = 0; idx < arr.length; idx++)
        copy[idx] = arr[idx];

    Arrays.sort(copy, (a, b) -> a - b); // Increasing Order
    for (int val : copy)
        System.out.print(val + " ");
    System.out.println();

    Arrays.sort(copy, (a, b) -> b - a); // Decreasing Order
    for (int val : copy)
        System.out.print(val + " ");
    System.out.println();
}
```

Integer

Integer

# Approach (1) Custom Lambda Expression

Marks of PCM (GFG)

```java
class Student{
    int phy, chem, maths;
    Student(int phy, int chem, int maths){
        this.phy = phy;
        this.chem = chem;
        this.maths = maths;
    }
}
public void customSort (int phy[], int chem[], int math[], int N)
{
    Student[] stud = new Student[N];
    for(int idx = 0; idx < N; idx++){
        stud[idx] = new Student(phy[idx], chem[idx], math[idx]);
    }

    Arrays.sort(stud, (a, b) -> {
        if(a.phy != b.phy) return a.phy - b.phy;
        // Increasing order of physics
        if(a.chem != b.chem) return b.chem - a.chem;
        // Same in Physics, Decreasing Order of Chemisty
        return a.maths - b.maths;
        // Same in Phy & Chem, Increasing order in maths
    });

    for(int idx = 0; idx < N; idx++){
        phy[idx] = stud[idx].phy;
        chem[idx] = stud[idx].chem;
        math[idx] = stud[idx].maths;
    }
}
```

Students Type

```java
String[] names = { "Guneet", "Vrushabh", "Chinmay", "Raghav", "Hardik", "Archit" };
Arrays.sort(names);
for (String val : names)
    System.out.print(val + " ");
System.out.println();
```

Archit < Chinmay < Guneet < Hardik < Raghav < Vrushabh → Lexicographically increasing order

Lexicographical:

- Chinmay < Chirag
- Chin < Chinmay

"g21211111" < "g2129"

"g212" < "g21200000"

# largest no from Array

52 90 6 92 5 97 50 9 59 9927

"9 97 92 9927 90 6 59 5 52 50"

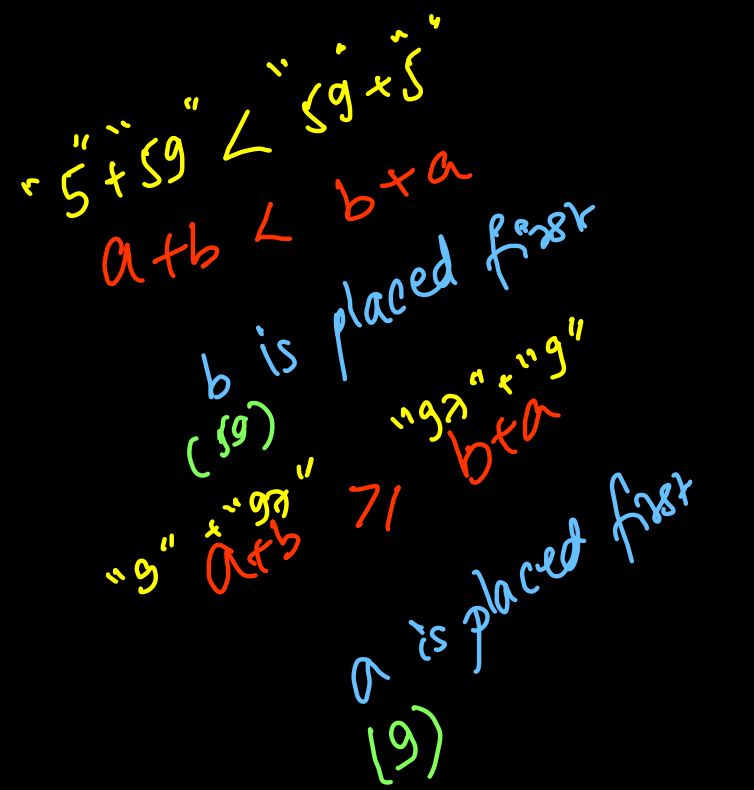
9297

5952

9 + 97 > 97 + 9

59 + 5 > 5 + 59

```java
public String largestNumber(int[] arr) {
    String[] copy = new String[arr.length];
    boolean allZero = true;
    for(int idx = 0; idx < arr.length; idx++)
    {
        if(arr[idx] != 0) allZero = false;
        copy[idx] = Integer.toString(arr[idx]);
    }

    if(allZero == true) return "0";

    Arrays.sort(copy, (a, b) -> {
        if((a + b).compareTo(b + a) < 0) return +1;
        return -1;
    });

    StringBuilder res = new StringBuilder("");
    for(String val: copy) res.append(val);
    return res.toString();
}
```




Time → O(n log r * l)

Space → O(n)

```java
class Movie{
    int duration;
    double ratings;
    String name;

    public Movie(int duration, double ratings, String name) {
        this.duration = duration;
        this.ratings = ratings;
        this.name = name;
    }
}
```

Not possible

```java
public static void comparableVSComparator() {
    Movie[] arr = new Movie[5];

    arr[0] = new Movie(duration: 180, ratings: 4.5, name: "Avengers");
    arr[1] = new Movie(duration: 150, ratings: 5.0, name: "Titanic");
    arr[2] = new Movie(duration: 100, ratings: 3.0, name: "Spiderman");
    arr[3] = new Movie(duration: 200, ratings: 5.0, name: "Avatar");
    arr[4] = new Movie(duration: 50, ratings: 1.0, name: "Thor");

    Arrays.sort(arr);
}
```

Runtime Error : How to compare 2 movie objects?

# Approach #2: Using Comparable

```java
class Movie implements Comparable<Movie> {
    int duration;
    double ratings;
    String name;

    public Movie(int duration, double ratings, String name) {
        this.duration = duration;
        this.ratings = ratings;
        this.name = name;
    }

    public int compareTo(Movie other) {
        return this.duration - other.duration;
        // Default Sorting: Based on Increasing Order of Duration
    }

    public String toString() {
        return "Name : " + this.name + " of " + this.duration
                + " Minutes " + " with " + ratings + " ratings";
    }
}
```

```java
Movie[] arr = new Movie[5];

arr[0] = new Movie(duration: 180, ratings: 4.5, name: "Avengers");
arr[1] = new Movie(duration: 150, ratings: 5.0, name: "Titanic");
arr[2] = new Movie(duration: 100, ratings: 3.0, name: "Spiderman");
arr[3] = new Movie(duration: 200, ratings: 5.0, name: "Avatar");
arr[4] = new Movie(duration: 50, ratings: 1.0, name: "Thor");

Arrays.sort(arr);

for (Movie val : arr)
    System.out.println(val);
```

```
architaggarwal@Archits-MacBook-Air Java OOPS % java OOPS_Codes.LambdaExpression
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
```

```java
class MovieDurationIncreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return a.duration - b.duration;
    }
}

class MovieDurationDecreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return b.duration - a.duration;
    }
}

class MovieLexicographicalComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return a.name.compareTo(b.name);
    }
}

class MovieRatingIncreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        if (a.ratings - b.ratings < 0)
            return -1;
        return +1;
    }
}
```

```java
Arrays.sort(arr, new MovieDurationIncreasingComparator());

for (Movie val : arr)
    System.out.println(val);

System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieDurationDecreasingComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieRatingIncreasingComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieLexicographicalComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");
```

Approach ③ Using Comparator

```
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
 ----------
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Thor of 50 Minutes  with 1.0 ratings
 ----------
```

```
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
 ----------
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
 ----------
```

→ functional interfaces in Java

eg Runnable, Comparable, Comparator, etc

↳ Anonymous classes / Two name

Interface with only one abstract function

```
interface Operation {
    int operation(int a, int b);
}
```

```
Operation sum = (a, b) -> a + b;
Operation prod = (a, b) -> a * b;
Operation sub = (a, b) -> a - b;
```

} Implementing Interface or overriding function in interface

```
private int operate(int a, int b, Operation op) {
    return op.operation(a, b);
}
```

```
LambdaFunctions myCalculator = new LambdaFunctions();
System.out.println(myCalculator.operate( a: 5,  b: 3, sum));
System.out.println(myCalculator.operate( a: 5,  b: 3, prod));
System.out.println(myCalculator.operate( a: 5,  b: 3, sub));
```

@ Functional Interface

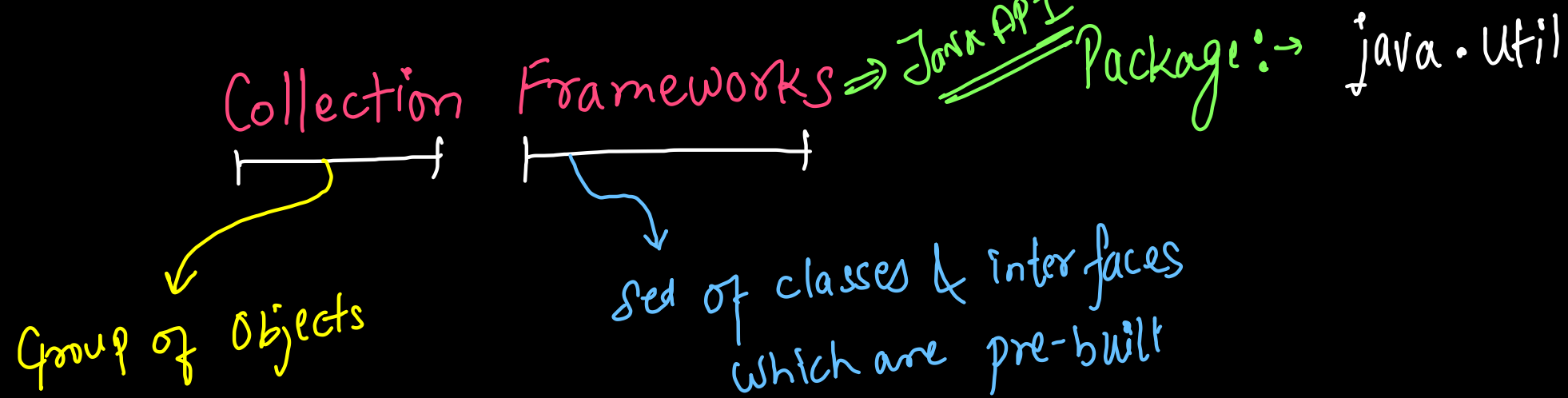
```java
interface Operation {
    public int applyOp(int a, int b);
}

class Sum implements Operation {
    public int applyOp(int a, int b) {
        return a + b;
    }
}

class Difference implements Operation {
    public int applyOp(int a, int b) {
        return a - b;
    }
}

class Product implements Operation {
    public int applyOp(int a, int b) {
        return a * b;
    }
}

class Division implements Operation {
    public int applyOp(int a, int b) {
        return a / b;
    }
}
```

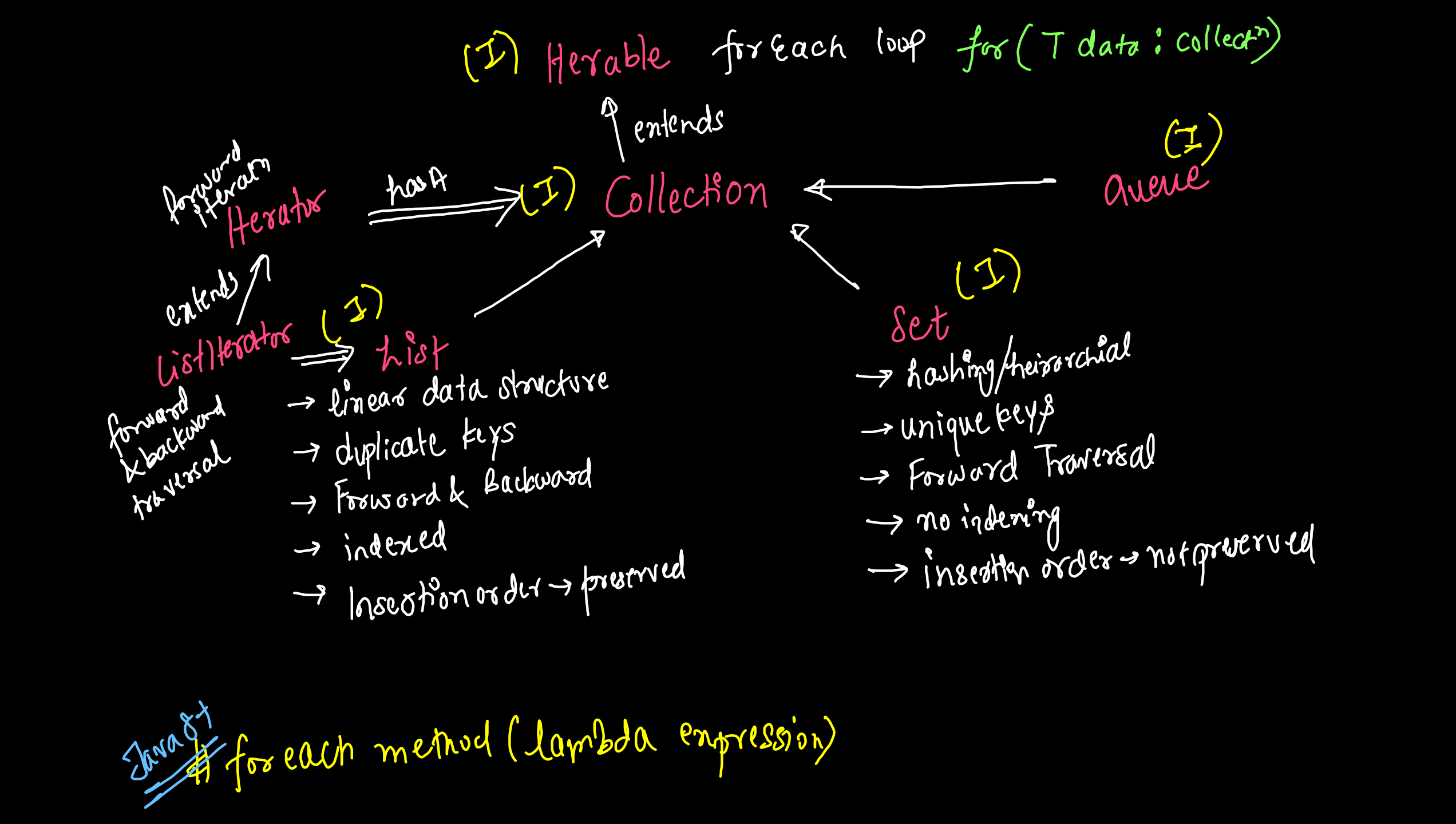
```java
Operation obj = new Sum();
System.out.println(obj.applyOp(a: 15, b: 10));

Operation obj2 = new Difference();
System.out.println(obj2.applyOp(a: 15, b: 10));

Operation obj3 = new Product();
System.out.println(obj3.applyOp(a: 15, b: 10));

Operation obj4 = new Division();
System.out.println(obj4.applyOp(a: 15, b: 10));
```

→ 25
→ 5
→ 150
→ 1

OR

```java
Operation sum = (a, b) -> a + b;
Operation diff = (a, b) -> a - b;
Operation prod = (a, b) -> a * b;
Operation div = (a, b) -> a / b;

System.out.println(sum.applyOp(a: 15, b: 10));
System.out.println(diff.applyOp(a: 15, b: 10));
System.out.println(prod.applyOp(a: 15, b: 10));
System.out.println(div.applyOp(a: 15, b: 10));
```


Collection
Frameworks
Java API
Package :-
java.util
Group of Objects
Set of classes & interfaces
which are pre-built


(I) Herable
for each loop
for (T data : collection)
extends
forward iterator
Iterator
has A
(I) Collection
(I) Queue
extends
ListIterator
(I) List
(I) Set
forward & backward traversal
→ linear data structure
→ duplicate keys
→ Foroward & Backward
→ indexed
→ Insertion order → preserved
→ hashing/heirarchial
→ unique keys
→ Forward Traversal
→ no indexing
→ insertion order → not preserved
Java 8
for each method (lambda expression)

List

implements

Arraylist
↓
Contiguous
not thread safe

imp.

Linkedlist
DLL
non contiguous

imp.

Enumeration

Vector
legacy class
thread safe
now
extends

Stack
legacy class

asymptotic

O(1) average

O(n) worst

Set

implements

HashSet
→ Completely random order

extends

LinkedHashSet
→ Insertn order preserved

extends

SortedSet

RedBlack Tree
(self balancing BST)

TreeSet
→ ordered

O(log₂n)
Insertn/Searching
Deletn

write

rollback (undo)

commit

google doc

delete

ET

read/write

Data Integrity ↓
Consistent ↓
Race condition

```java
Set<Integer> s1 = new HashSet<>();
s1.add(e: 30);
s1.add(e: 10);
s1.add(e: 40);
s1.add(e: 50);
s1.add(e: 20);
s1.add(e: 10); // Ignored

Set<Integer> s2 = new LinkedHashSet<>();
s2.add(e: 30);
s2.add(e: 10);
s2.add(e: 40);
s2.add(e: 50);
s2.add(e: 20);
s2.add(e: 10); // Ignored

Set<Integer> s3 = new TreeSet<>();
s3.add(e: 30);
s3.add(e: 10);
s3.add(e: 40);
s3.add(e: 50);
s3.add(e: 20);
s3.add(e: 10); // Ignored
```

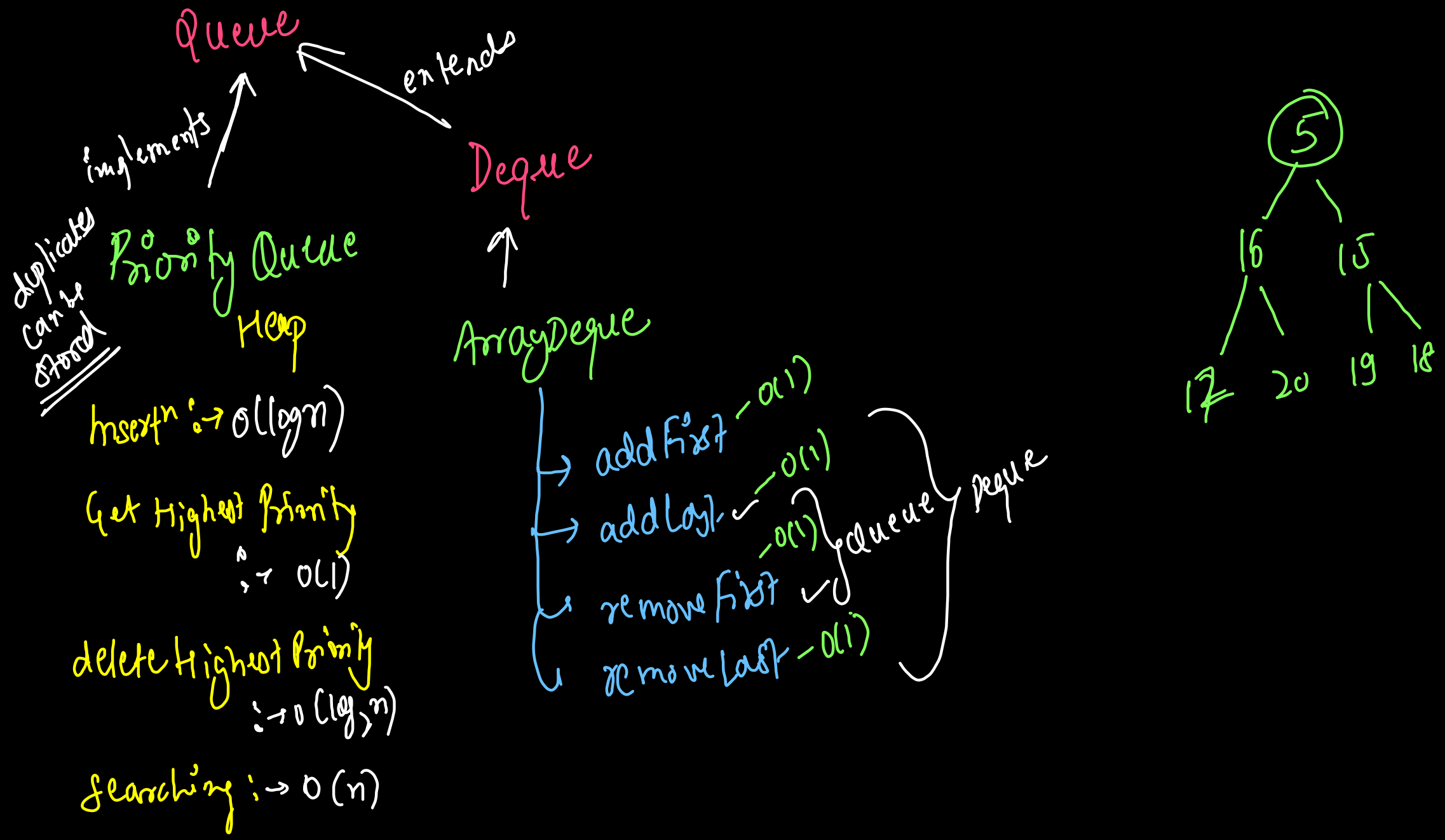
```java
for (Integer a : s1)
    System.out.print(a + " ");
System.out.println();

for (Integer a : s2)
    System.out.print(a + " ");
System.out.println();

for (Integer a : s3)
    System.out.print(a + " ");
System.out.println();
```

→ Random

→ Insertion order

→ Sorted (Inc)

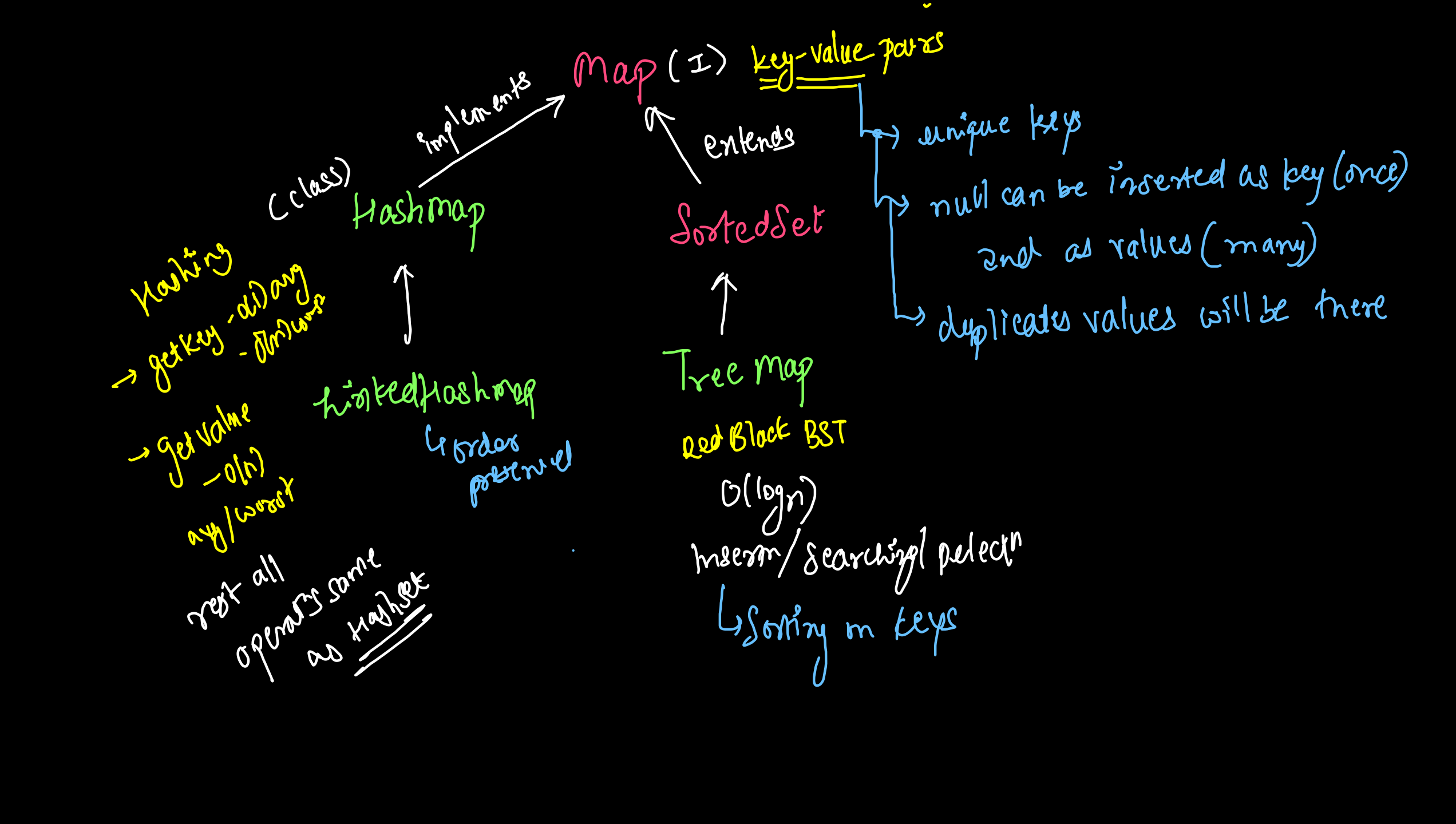
```
50 20 40 10 30
30 10 40 50 20
10 20 30 40 50
```


Queue
extends
implements
Deque
duplicate can be stored
Priority Queue
Heap
ArrayDeque
Insertn :→ O(log n)
Get Highest Priority :→ O(1)
delete Highest Priority :→ O(log n)
Searching :→ O(n)
addFirst - O(1)
addLast - O(1)
removeFirst - O(1)
removeLast - O(1)
Queue
Deque
5
16
15
17
20
19
18

```java
Queue<Integer> q1 = new ArrayDeque<>();
q1.add(e: 30);
q1.add(e: 10);
q1.add(e: 10);
q1.add(e: 20);
q1.add(e: 40);
q1.remove();

System.out.println(q1);

Deque<Integer> q2 = new ArrayDeque<>();
q2.addFirst(e: 30);
q2.addLast(e: 50);
q2.addLast(e: 10);
q2.add(e: 20);
q2.add(e: 30);
q2.remove();
q2.removeFirst();
q2.removeLast();

System.out.println(q2);
```

```java
Queue<Integer> q3 = new PriorityQueue<>();
q3.add(e: 30);
q3.add(e: 50);
q3.add(e: 10);
q3.add(e: 20);
q3.add(e: 60);
q3.add(e: 70);
q3.add(e: 90);
q3.add(e: 20);
q3.add(e: 30);

System.out.println(q3); // Not Necessarily Sorted (Heap Order Property)

// Heap Sort -> Sorted
while (q3.size() > 0) {
    System.out.print(q3.remove() + " ");
}
```

Duplicates allowed
↳ Data not stored in sorted form but deleted from HP → LP


Map (I)
key-value pairs
implements
extends
(class)
HashMap
SortedSet
unique keys
null can be inserted as key (once)
and as values (many)
duplicates values will be there
Hashing
→ getKey - O(1) avg
- O(n) worst
→ getValue
- O(n)
avg/worst
rest all operations same as HashSet
LinkedHashmap
↳ order preserved
TreeMap
Red Black BST
O(log n)
insertion/searching/deletion
↳ Sorting on keys

```java
Map<String, Integer> m1 = new HashMap<>();
m1.put(key: "Delhi", value: 30);
m1.put(key: "Delhi", value: 10);
m1.put(key: null, value: 40);
m1.put(key: null, value: 50);
m1.put(key: "Mumbai", value: null);
m1.put(key: "Kolkatta", value: null);

Map<String, Integer> m2 = new LinkedHashMap<>();
m2.put(key: "Delhi", value: 30);
m2.put(key: "Delhi", value: 10);
m2.put(key: null, value: 40);
m2.put(key: null, value: 50);
m2.put(key: "Mumbai", value: null);
m2.put(key: "Kolkatta", value: null);

Map<String, Integer> m3 = new TreeMap<>();
m3.put(key: "Delhi", value: 30);
m3.put(key: "Delhi", value: 10);
// m3.put(null, 40); Tremap key cannot be null
m3.put(key: "Mumbai", value: null);
m3.put(key: "Kolkatta", value: null);
```

```java
for (String a : m1.keySet())
    System.out.print(a + " -> " + m1.get(a) + " ");
System.out.println();

for (String a : m2.keySet())
    System.out.print(a + " -> " + m2.get(a) + " ");
System.out.println();

for (String a : m3.keySet())
    System.out.print(a + " -> " + m3.get(a) + " ");
System.out.println();
```

↳ random order

↳ insertion order

↳ sorted order (on keys)

```
null -> 50 Delhi -> 10 Kolkatta -> null Mumbai -> null
Delhi -> 10 null -> 50 Mumbai -> null Kolkatta -> null
Delhi -> 10 Kolkatta -> null Mumbai -> null
```

```java
Map<Student, Integer> m4 = new HashMap<>();

Student st1 = new Student();
st1.rollNo = 1;
Student st2 = new Student();
st2.rollNo = 2;
Student st3 = new Student();
st3.rollNo = 3;
Student st4 = new Student();
st4.rollNo = 1;
Student st5 = st2;
```

Object — Override

st1 — 4k — 500
st2 — 6k — 700
st3 — 8k — 100
st4 — 10k — 500
st5 — 6k — 700

```java
m4.put(st1, value: 10);
m4.put(st2, value: 20);
m4.put(st3, value: 30);
m4.put(st4, value: 40);
m4.put(st5, value: 50);

System.out.println(m4);
```

```java
class Student {
    int marks;
    int rollNo;
    String name;

    @Override
    public int hashCode() {
        return Integer.hashCode(rollNo);
    }

    @Override
    public boolean equals(Object other) {
        if (this.hashCode() == other.hashCode())
            return true;
        return false;
    }
}
```

custom hashing

If there hashcode will not be there then 4 objects hashing based on address

3 keys ⇒ st1 == st4, st2 == st5

```
{OOPS_Codes.Student@1=40, OOPS_Codes.Student@2=50, OOPS_Codes.Student@3=30}
```

```java
Map<ArrayList<Integer>, Integer> m5 = new HashMap<>();

ArrayList<Integer> a1 = new ArrayList<>();
a1.add(e: 10);
a1.add(e: 20);

ArrayList<Integer> a2 = new ArrayList<>();
a2.add(e: 10);
a2.add(e: 20);

ArrayList<Integer> a3 = a1;

m5.put(a1, value: 100);
m5.put(a2, value: 200);
m5.put(a3, value: 300);

System.out.println(m5);
```

hashcode() overridden & data is compared

```
{[10, 20]=300}
```

Forward Iteration

# Iterable & Iterator

extended by

Collection Interface

⇓

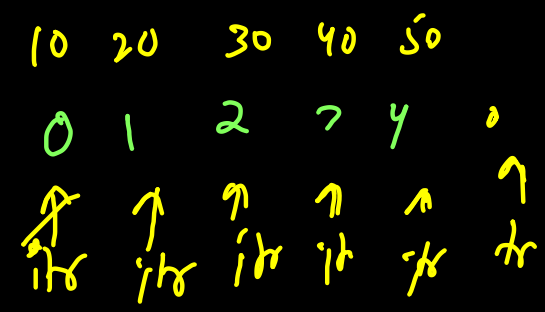
Syntatical Sugar

for each loop

```
for( object T: collection){

}
```

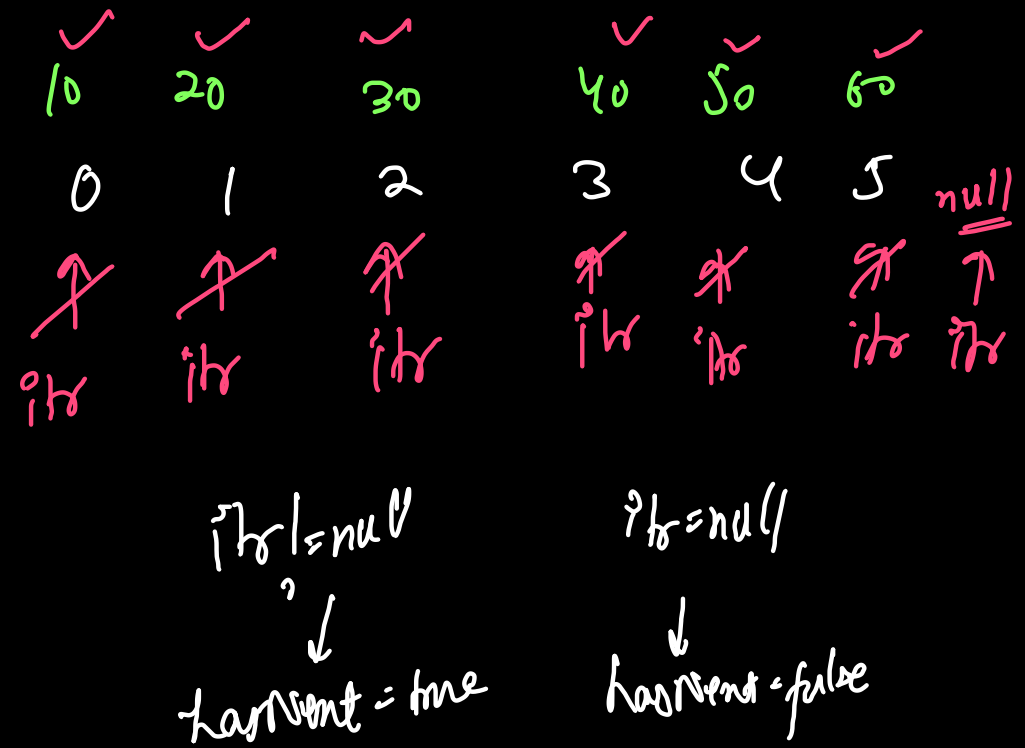
Collection HAS A iterator interface reference variable

- next() :→ return current element & setup for next element
  - If no elements found, throws Exception
- hasNext() :→ true if element(s) are remaining to be traversed otherwise false



```java
ArrayList<Integer> arr = new ArrayList<>();
arr.add(e: 10);
arr.add(e: 20);
arr.add(e: 30);
arr.add(e: 40);
arr.add(e: 50);
arr.add(e: 60);

// Iterable : For Each Loop : Syntatical Sugar
for (Integer data : arr) {
    System.out.print(data + " ");
}
System.out.println();

// For Each Method (Java 8+ Feature)
arr.forEach((data) -> System.out.print(data + " "));
System.out.println();

// Iterator:
Iterator<Integer> itr = arr.iterator();
while (itr.hasNext() == true) {
    System.out.print(itr.next() + " ");
}
```

hasA relationship



```java
// Enumeration: Iterate on Vector and Stack
Vector<Integer> v = new Vector<>();
v.add(e: 10);
v.add(e: 20);
v.add(e: 30);
v.add(e: 40);
v.add(e: 50);
v.add(e: 60);

Enumeration<Integer> e = v.elements();
while (e.hasMoreElements() == true) {
    System.out.print(e.nextElement() + " ");
}
System.out.println();

// List Iterator
ListIterator<Integer> li = arr.listIterator();
while (li.hasNext() == true) {
    System.out.print(li.next() + " ");
}
System.out.println();

ListIterator<Integer> bi = arr.listIterator(arr.size());
while (bi.hasPrevious() == true) {
    System.out.print(bi.previous() + " ");
}
System.out.println();
```

```
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
60 50 40 30 20 10
```

→ n th element is not included!

# Custom Iterators in Java

(1) Peeking Iterator LC 284

(2) Flatten Nested List Iterator LC 341

(3) BST Iterator - I LC 173

(4) BST Iterator - II Leetcode Locked - Codestudio

(5) Two Sum in BST LC 653

# Peeking Iterator

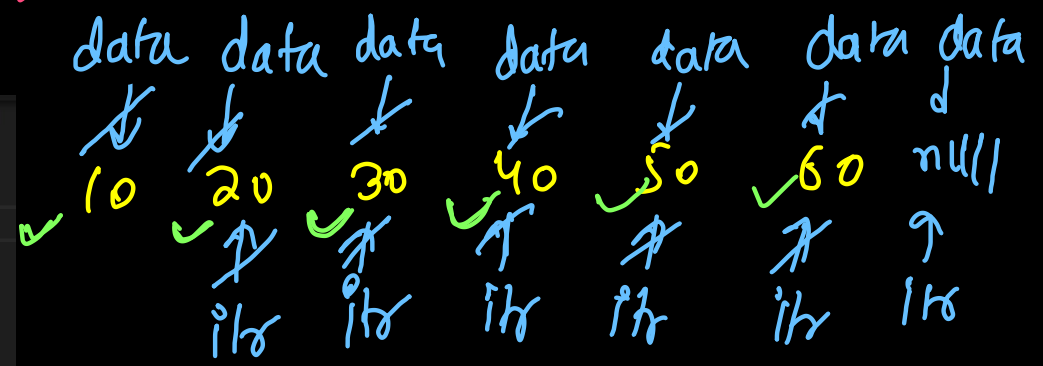
```java
class PeekingIterator implements Iterator<Integer> {
    Iterator<Integer> itr;
    Integer data;

    public PeekingIterator(Iterator<Integer> itr) {
        this.itr = itr;
        next();
    }

    public Integer peek() {
        return data;
    }

    @Override
    public Integer next() {
        Integer temp = data;

        if(itr.hasNext() == true){
            data = itr.next();
        } else {
            data = null;
        }

        return temp;
    }

    @Override
    public boolean hasNext() {
        return (data != null);
    }
}
```

1) Constructor

2) peek()

3) next()

4) hasNext()




peek() → 10

next() → 10, d=20, i=30

peek() → 20

next() → 20, d=30, i=40

peek() → 30

next() → 30, d=40, i=50

peek() → 40

next() → 40, d=50, i=60

peek() → 50

next() → 50, d=60

hasNext()
↳ data != null

peek → 60

next() → 60, d=null

hasNext()
↳ data == null

```java
class PeekingIterator implements Iterator<Integer> {
    Iterator<Integer> itr;
    Integer data;

    public PeekingIterator(Iterator<Integer> itr) {
        this.itr = itr;
        next();
    }

    public Integer peek() {
        return data;
    }

    @Override
    public Integer next() {
        Integer temp = data;
        if(itr.hasNext() == true){
            data = itr.next();
        } else {
            data = null;
        }

        return temp;
    }

    @Override
    public boolean hasNext() {
        return (data != null);
    }
}
```

Extra functionality

hasnext or collection

Iterator on collection

For List (AL, LL, Vector, Stack)
↳ peek, next, hasNext → O(1)

For Queue (ArrayDeque & PriorityQueue)
↳ peek, next, hasNext → O(1)

For Set (Hashmap)
↳ peek, next, hasNext → Θ(1) avg

ForSet (TreeSet)
↳ peek → O(1)
hasnext → O(1)
next → Θ(1) avg

# Flatten Nested List Iterator

NestedList ⇒ List< NestedInteger >

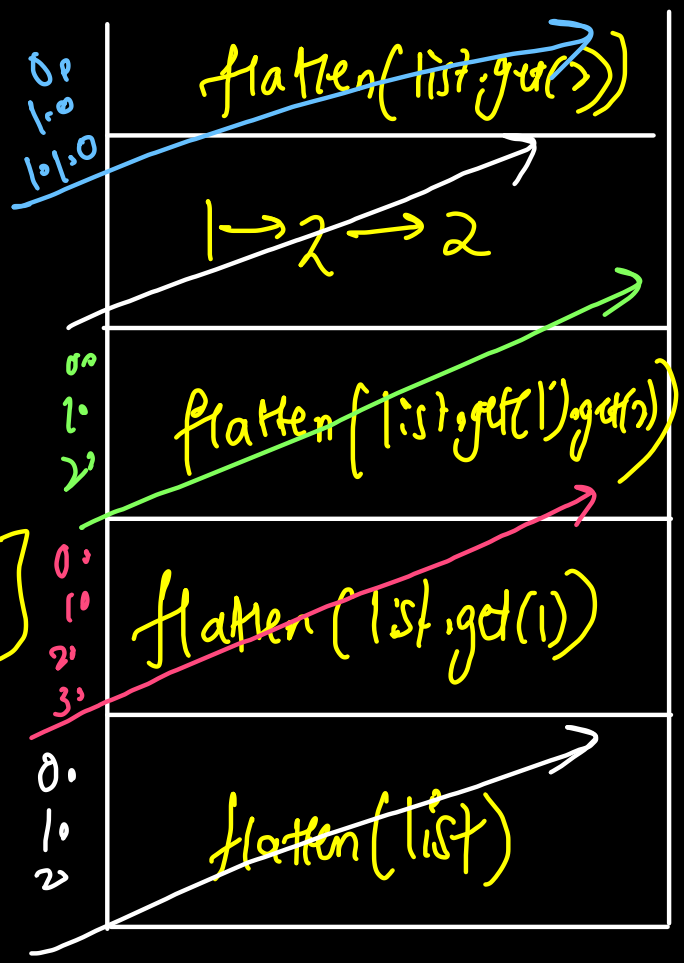
[ 10, [ 20, 30, [40, 50, [ ] ], 60 ], [ 70 [ 80 [ 90 ] ] ] ]

↓ convert to 1D list of integers

[ 10, 20, 30, 40, 50, 60 70, 80, 90 ]

↓ iterator (inbuilt) next & hasNext

```java
public interface NestedInteger {

    // @return true if this NestedInteger holds a single integer, rather than a nested list.
    public boolean isInteger();

    // @return the single integer that this NestedInteger holds, if it holds a single integer
    // Return null if this NestedInteger holds a nested list
    public Integer getInteger();

    // @return the nested list that this NestedInteger holds, if it holds a nested list
    // Return empty list if this NestedInteger holds a single integer
    public List<NestedInteger> getList();
}
```


[10, 20, 30, 40, 50, 60, ]