JAVA

java collection :

1.making our own genric class template

import java.util.\*;

class OurGenericClass<T> {

    private T[] items;

    private *int* size;

    public OurGenericClass() {

        this.size = 0;

        items = (T[]) new Object[100];

    }

    public *void* add(T *item*) {

        items[size++] = *item*;

    }

    public T getItems(*int* *index*) {

        return items[*index*];

    }

}

public class SecondClass {

    public static *void* main(String *arg*[]) {

        OurGenericClass<Integer> list = new OurGenericClass<Integer>();

        list.add(1);

        list.add(1);

        list.add(1);

        System.out.println(list.getItems(0));

    }

};

2. to make genric class iterable we need to add ahead of our genric class : implement Iterable<T>

import java.util.\*;

class OurGenericList<T> implements Iterable<T> {

    private T[] items;

    private *int* size;

    public OurGenericList() {

        this.size = 0;

        items = (T[]) new Object[100];

    }

    // defining iterator

    public Iterator<T> iterator() {

        return new OurGenericListIterator(this);

    }

    private class OurGenericListIterator implements Iterator<T> {

        private OurGenericList<T> list;

        private *int* index = 0;

        public OurGenericListIterator(OurGenericList<T> *list*) {

            this.list = *list*;

        }

        public *boolean* hasNext() {

            return index < list.size;

        }

        @*Override*

        public T next() {

            return list.items[index++];

        }

    }

    public *void* add(T *item*) {

        items[size++] = *item*;

    }

    public T getItems(*int* *index*) {

        return items[*index*];

    }

}

public class SecondClass {

    public static *void* main(String *arg*[]) {

        OurGenericList<Integer> list = new OurGenericList<Integer>();

        list.add(1);

        list.add(1);

        list.add(1);

        // Iterator<Integer> iterator=list.iterator();

        // while(iterator.hasNext()){

        // System.out.println(iterator.next());

        // }

        for (*int* i : list) {

            System.out.println(i);

        }

    }

};

Genral method :

i)add

ii)addall

iii)remove

iv)removeall

v)clear

vi)size

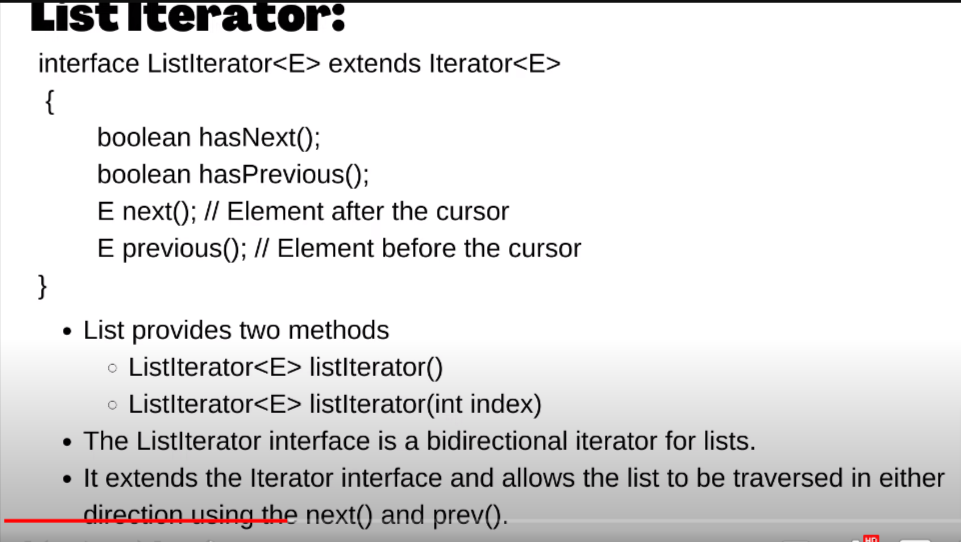
vii)iterate

* LIST

1.List interface

A)ArrayList B)LinkedList (doubly linked list) C)Vector

2.List Iterator



// Iterator<Integer> iterator=list.iterator();

        // while(iterator.hasNext()){

        // System.out.println(iterator.next());

        // }

 // Incase of iterator

        // next return list[index++];

        // prev return [--index];

1.ArrayList method

List<Integer> list = new ArrayList<>();

        // 1.add method

        list.add(1);

        list.add(2);

        list.add(3);

        // 2. print arr

        System.out.println(list);

        // 3. remove

        list.remove(1);

        // 4.size

        System.out.println(list.size());

        // 5.get element at particular size

        System.out.println(list.get(0));

        // 6.set element

        System.out.println(list.set(0, 5));

        // 7. Creating a copy

        List<Integer> CpyList = new ArrayList<>(list);

        System.out.println(CpyList);

        // 7.1 to create a array list with some element of last array

        List<Integer> sliceList = list.subList(0, 2); // will not

        // going to take the element at the last index

        // subList only take the reference of last array , so if we change any element in last array it will going to be change in Sublist

        // 8. to get the first occurrence of element in list

        System.out.println(list.indexOf(3));

        // 9.to get the last index

        System.out.println(list.lastIndexOf(3));

2.LinkedList

 List<Integer> list = new LinkedList<>();

        // 1.add method

        list.add(1);

        list.add(2);

        list.add(3);

        list.add(4);

        list.add(5);

        list.add(6);

        // all method of Array list

        /\*

         \*1.get 2.set 3.add 4.add(0,2) 5.addAll(list) 6. remove(index) 7.removeAll() 8.creating a cpy of LinkedList  9.subList 10.indexOf(value) 11.lastIndexOf

         12.print

         \*/

 // Incase of iterator

        // next return list[index++];

        // prev return [--index];

 // converting linked list to arr

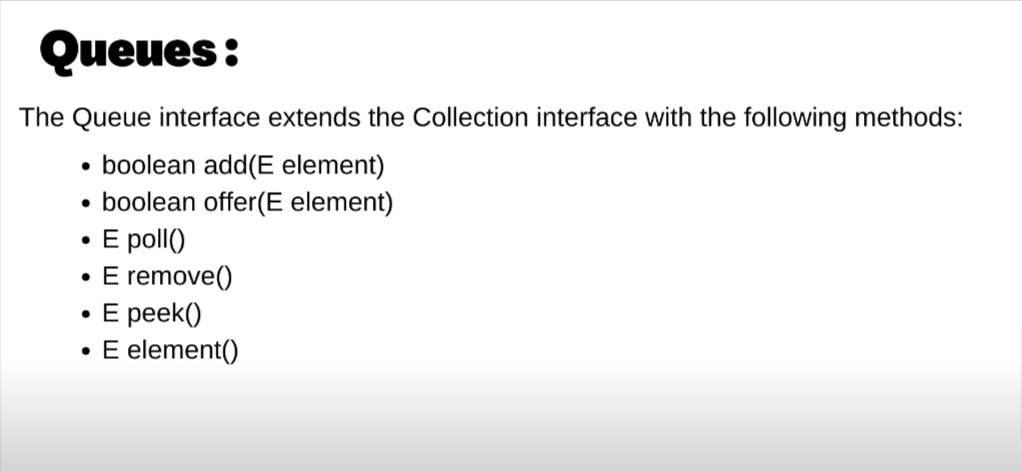
        Integer arr[]=list.toArray(new Integer[0]);

        for(*int* i : arr){

            System.out.println(i);

        }

* QUEUES :



isEmpty : to check wheter it is empty or not

.size() : to find the size

diffrence between :

i)add VS offer :

add- do not give any error or exception if we will not be able to add element

offer- give exception if operation failed (most prefered)

ii)pool VS remove :

pool : donot show any exception when operation falied

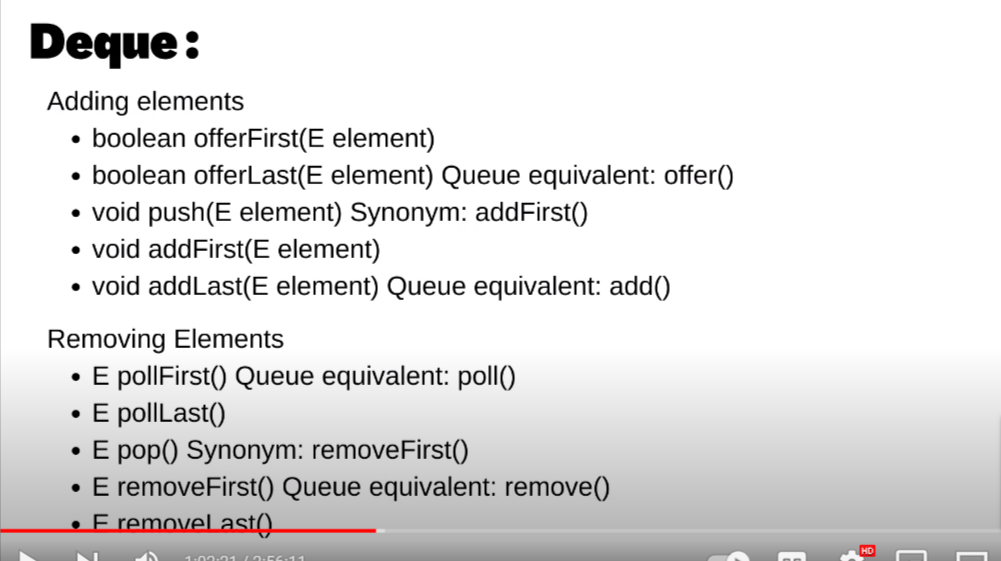
remove : donot show any exception when operation falied

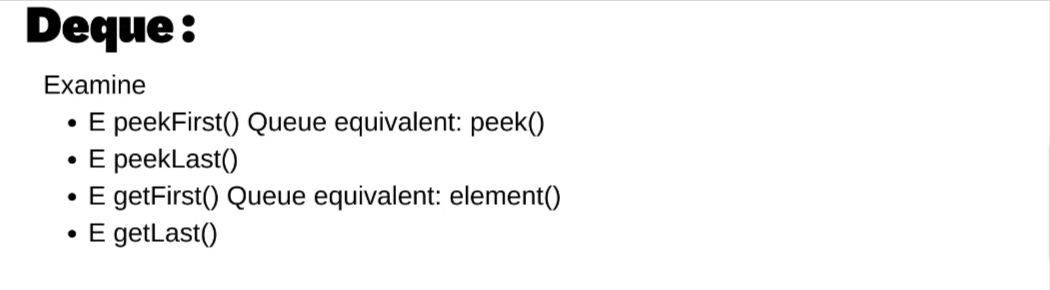
iii)peek vs element :

peek : do not give exception if there will be no top element

element: give exception

* DEQUE :





note :

i)FOR QUEUE WE use LinkedList interface of QUEUE

ii)FOR STACK WE use DEQUEUE interface of QUEUE

Queue implementation using LinkedList interface :

Queue<Integer> queue = new LinkedList<>();

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            queue.offer(sc.nextInt());

        }

        while (!queue.isEmpty()) {

            System.out.println(queue.peek());

            queue.remove();

        }

Stack implementation :

Stack<Integer> st = new Stack<>();

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            // for pushing element

            st.push(sc.nextInt());

        }

        while (!st.isEmpty()) {

            // for top element

            System.out.println(st.peek());

            st.pop();

        }

        // for size

        System.out.println(st.size());

Deque implementation :

Deque<Integer> dq=new ArrayDeque<>();

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            // for pushing element

            dq.offerFirst(sc.nextInt());

            dq.offerLast(sc.nextInt());

        }

        while (!dq.isEmpty()) {

            // for top element

            System.out.println(dq.peek());

            dq.pop();

        }

        // for size

        System.out.println(dq.size());

Comprable and comprator

diffrence :

i)comprable is natural oddering where as comprator is total odering

ii)comprable compare interface should be implemented by class itself where as comprator interface should be implemented by taking two object of class

ii)while implementing comprable interface we have to implement toCompare method where as to implement Comprator interface we have to implement compare method

implementation useing comprable :

import java.util.\*;

 class Student implements Comparable<Student>{

    public *int* math;

    public *int* physics;

    public Student(*int* *math*,*int* *physics*){

        this.math=*math*;

        this.physics=*physics*;

    }

    @*Override*

    public *int* compareTo(Student *s*){

        return *s*.math -this.math;

    }

};

public class SecondClass {

    public static *void* main(String *arg*[]) {

        PriorityQueue<Student> pq = new PriorityQueue<>();

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            pq.offer(new Student(sc.nextInt(),sc.nextInt()));

        }

        while (!pq.isEmpty()) {

            Student st=pq.remove();

            System.out.println("Math : "+st.math + "Physics: "+st.physics);

        }

    }

};

IMPLEMENTATION OF COMPRATOR INTERFACE :

import java.util.\*;

class Cmp implements Comparator<Integer> {

    @*Override*

    public *int* compare(Integer *a*, Integer *b*) {

        return *a* - *b*;

    }

};

public class SecondClass {

    public static *void* main(String *arg*[]) {

        PriorityQueue<Integer> pq = new PriorityQueue<>(new Cmp());

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            pq.offer(sc.nextInt());

        }

        while (!pq.isEmpty()) {

            System.out.println(pq.remove());

        }

    }

};

* PRIORITY QUEUE :

// Minheap

PriorityQueue<Integer> pq = new PriorityQueue<>();

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            pq.offer(sc.nextInt());

        }

        while (!pq.isEmpty()) {

            System.out.print(pq.remove());

        }

// Max heap

 Comparator<Integer>cmp=new Comparator<Integer>() {

            public *int* compare(Integer *n1*,Integer *n2*){

                // if(n1<n2){

                //     return 1;

                // }

                // else if(n1>n2){

                //     return -1;

                // }

                // else{

                //     return 0;

                // }

                return *n2*-*n1*;

            }

        };

        PriorityQueue<Integer> pq = new PriorityQueue<>(cmp);

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            pq.offer(sc.nextInt());

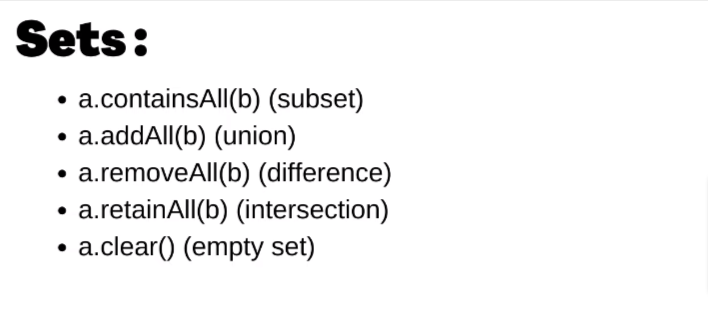
        }

        while (!pq.isEmpty()) {

            System.out.print(pq.remove());

        }

SET :



1.to add element : use set.add(element);

2.to remove element use : set.remove(element)

3.To check a element is present or not : set.contain(element)

3. to get the size : set.size();

4.for traversal : for(int i:set){}

1.hasset (unordered\_Set in c++)

  Set<Integer> set = new HashSet<>();

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            set.add(sc.nextInt());

        }

        System.out.print(set);

    }

note :in hashset insertion ordered is not retain element will be in sorted order to maintain order we can use linkHashSet

2.LinkHashSet

 Set<Integer> set = new LinkedHashSet<>();

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            set.add(sc.nextInt());

        }

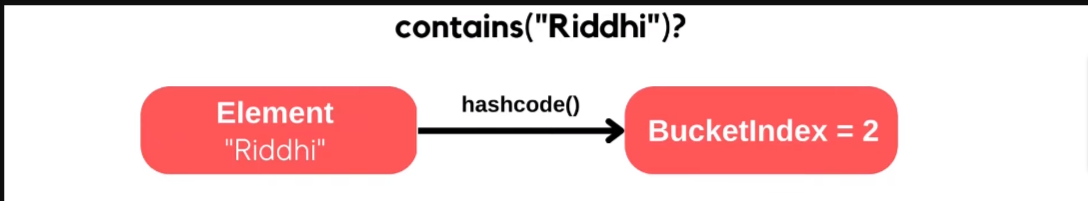
        System.out.print(set);

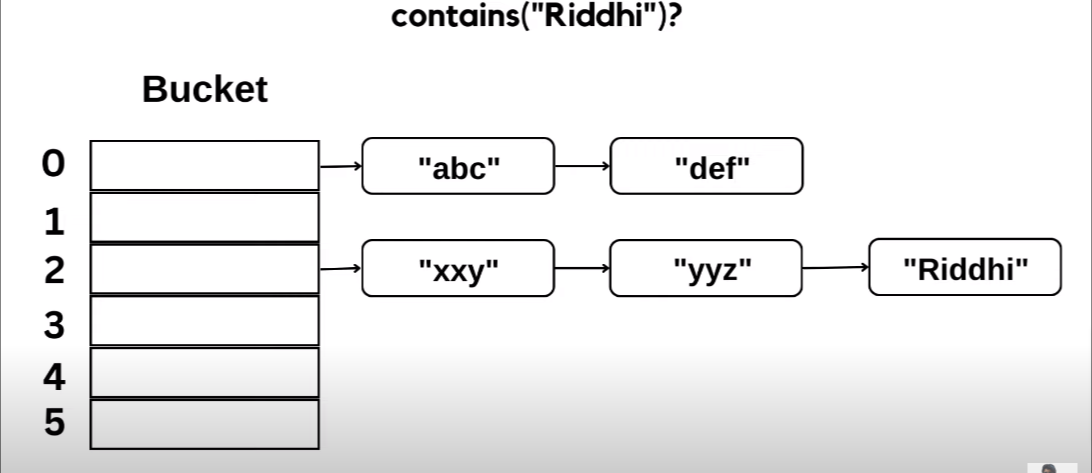
* wrong ans in



because :

1)





so , when we are calling contains method then it find the hasconde crosspounding to that hash code it get bucket index no , but we have’nt implemented the hascode and equall method to override the existing method of hasSet.

import java.util.\*;

class Student {

    private *int* math;

    private *int* physics;

    public Student(*int* *math*, *int* *physics*) {

        this.math = *math*;

        this.physics = *physics*;

    }

    // implementing hashCode

    @*Override*

    public *int* hashCode() {

         final *int* prime = 31;

*int* result = 1;

        result = prime \* result + math;

        result = prime \* result + physics;

        return result;

    }

    // implementing equal

    public *boolean* equals(Object *obj*) {

        if (this == *obj*) {

            return true;

        }

        if (*obj* == null) {

            return false;

        }

        if (getClass() != *obj*.getClass()) {

            return false;

        }

        // typecasting the obj

        Student other = (Student) *obj*;

        if (math != other.math) {

            return false;

        }

        if (physics != other.physics) {

            return false;

        }

        return true;

    }

};

public class SecondClass {

    public static *void* main(String *arg*[]) {

        Set<Student> set = new HashSet<>();

        Scanner sc = new Scanner(System.in);

*int* n = sc.nextInt();

        for (*int* i = 0; i < n; i++) {

            set.add(new Student(sc.nextInt(), sc.nextInt()));

        }

        System.out.print(set.contains(new Student(56, 56)));

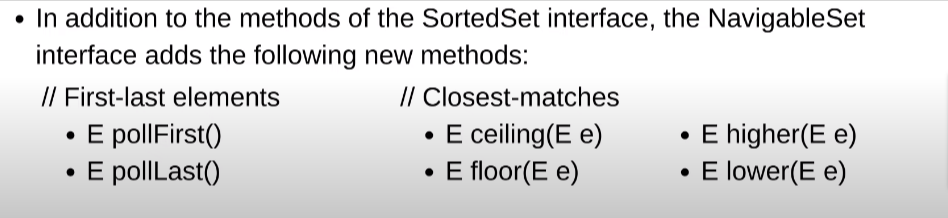
    }

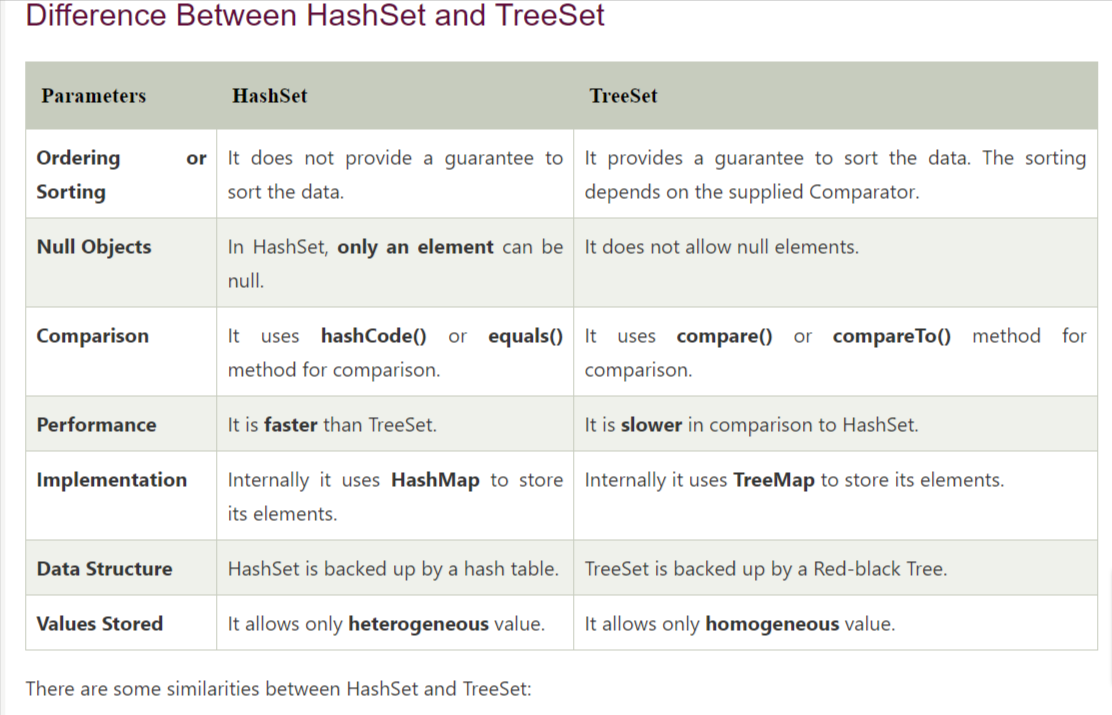
};

3.TreeSet (set in c++)

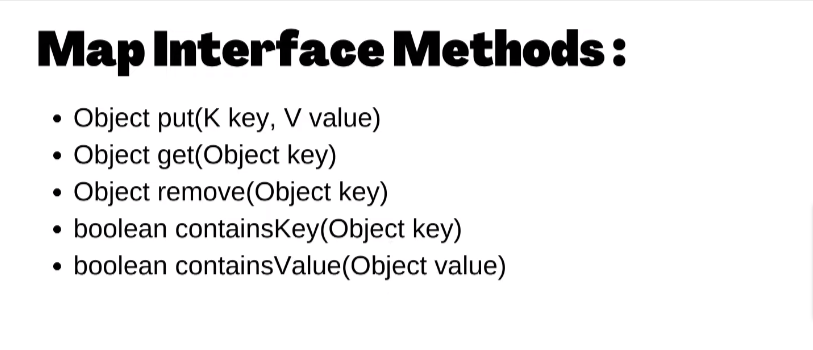
NavigableSet : inteface

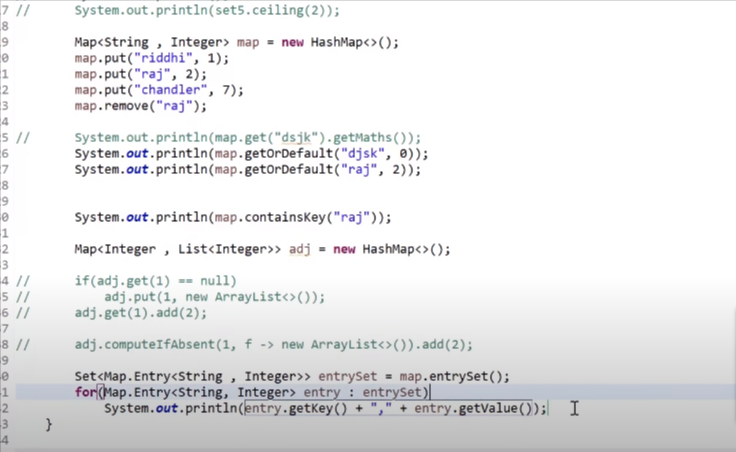
NavigableSet<Integer>set=new TreeSet<>();





Map : map are not iterable

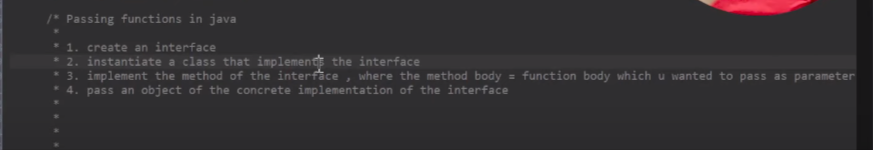




HashTree

Lamda fuction

1.passing function in java



a)creating interface : filterCondition.java

package Interface;

public interface FiltringCondition {

abstract void print();

b)creating a class which will going to implement method of interface

package My\_package;

import Interface.FiltringCondition;

public class Cat implements FiltringCondition {

private String name;

private int age;

public Cat() {};

public void print() {

System.***out***.println("meow");

}

}

c)creating a function

public static void printable(FiltringCondition Things) {

Things.print();

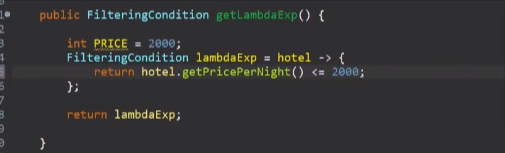
* Anonymous Inner Class
* package My\_package;
* import java.util.\*;
* import Interface.FiltringCondition;
* public class Myclass {
* public static boolean printable(FiltringCondition Things,int a) {
* return Things.print(a);
* }
* public static void main(String arg[]) {
* boolean ans=*printable*(new FiltringCondition(){
* public boolean print(int a) {
* return a%2!=0;
* }
* },8);
* System.***out***.println(ans);
* }
* }
* lamda expression
* package My\_package;
* import java.util.\*;
* import Interface.FiltringCondition;
* public class Myclass {
* public static boolean printable(FiltringCondition Things,int a) {
* return Things.print(a);
* }
* public static void main(String arg[]) {
* boolean ans=*printable*((int a)->{
* return a%2!=0;
* },7);
* System.***out***.println(ans);
* }
* }

note : lamda expression convert the written pice of code into the anomyous inner class.

ii) while using lambda expression the intrface class should be with 1 method only othrewise it start giving error due to ambiguity because the labda expression implementing things using anomyous inner class .

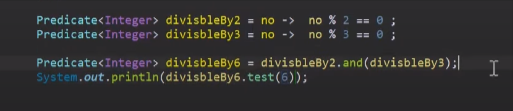
iii)when using this with the lambda function ,the the this will point to the class inwhich lamba function is defined

iv)when we define a varibale in lambda expression theen we can not change the value of varible due to value capturing .



note : we can not change the Value of PRICE because of value capture .

* pedicate is built in java interface



method :

1. .test();
2. .and();
3. .or()
4. package My\_package;
5. import java.util.\*;
6. import java.util.function.Consumer;
7. import Interface.FiltringCondition;
8. public class Myclass {
9. public static boolean printable(FiltringCondition Things,int a) {
10. return Things.print(a);
11. }
12. public static void main(String arg[]) {
13. List<Integer>list=new ArrayList<Integer>();
14. list.add(1);
15. list.add(2);
16. list.add(3);
18. Consumer<Integer> consumer= a->System.***out***.println(a);
20. list.forEach(consumer);
21. }

EXCEPTION HANDLING :

TYPE OF EXCEPTION –

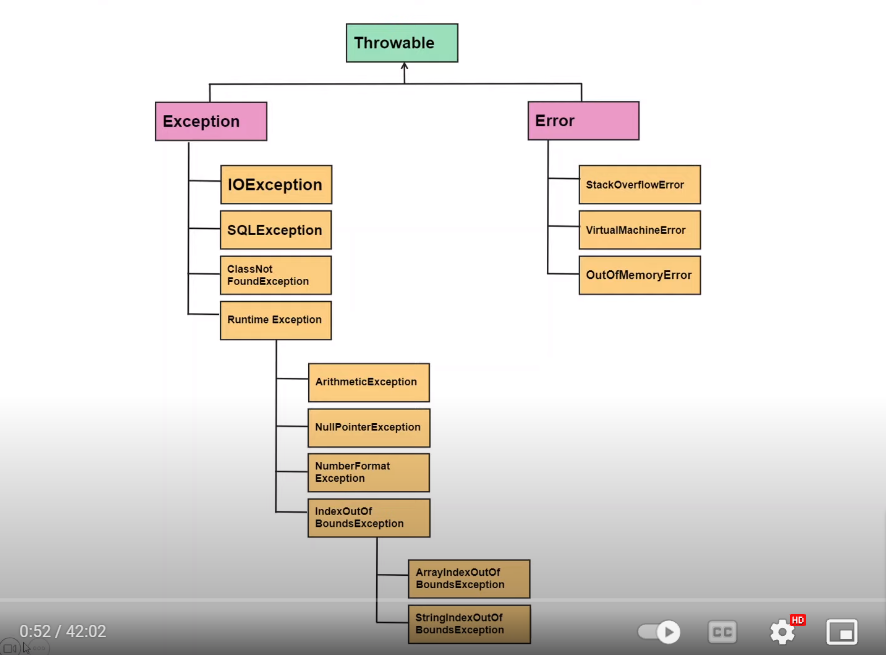
1)CHECKED OR COMPILE TIME EXCEPTION

2)UNCHECKED OR RUN TIME EXCEPTION

Diffrence between error and excption :

Error : with the error our programm will not going to run

Excption : with exception our programm will going to run and exception can be checked at compile time or run time



Example of compile time exception :

1. IOExcepiton
2. SQLExcepiton
3. ClassNotFoundExcepiton