Collections

1. **what is collections and what is it used for**?

collection is a combination of different types of containers

2. Available containers:

1. Set - Doesn't allow duplicates
    1. Hashset - No proper order,
    2. Treeset - Proper order.
2. List - Allow duplicates, proper order
    1. ArrayList
    2. LinkedList
3. Stack - Last in first out or first in last out (eg. Bank statements)
4. Queue - First in first out or Last in last out
5. Map - key value pair
    1.HashMap
    2.TreeMap
6. Vector - Size is not definite can be increased

3.Available collection methods:

retainAll() -- Retains only the elements in the list that are contained in the specified collection

clear() -- removes all elements from the list

set() -- replaces the element at the specified position in the list with the specified element

get() -- returns the element at the specified position in the list

indexOf() -- returns the index of the first occurrence of the specified element in the list, or -1 if the list does not contain the element

listIterator() -- returns a list iterator over the elements in the list

lastIndexOf() -- returns the index of the last occurrence of the specified element in the list, or -1 if the list does not contain the element

subList() -- returns a view of the portion of the list between the specified fromIndex and toIndex

sort() -- sorts the list according to the order induced by the specified Comparator

add() -- will return exception if couldn't add offer() -- return boolean value for the operation

remove() -- for removing values

poll() -- for removing values

peek() -- retrieves, but does not remove, the head of the queue, or returns null if the queue is empty

size() -- returns the number of elements in the collection

isEmpty() -- returns true if the collection is empty, false otherwise

put(key,value) -- associates the specified value with the specified key in the map

get(key) -- returns the value to which the specified key is mapped, or null if the map contains no mapping for the key

remove(key) -- removes the mapping for the specified key from the map if present

entrySet()=key,value -- returns a set view of the mappings contained in the map

keySet()=key -- returns a set view of the keys contained in the map

values() -- returns a collection view of the values contained in the map

putAll() -- copies all of the mappings from the specified map to this map

clear() -- removes all of the mappings from the map

replace() -- replaces the entry for the specified key only if it is currently mapped to some value

putIfAbsent() -- associates the specified value with the specified key in the map if the key is not already associated with a value

add():

set.add(1); --> set.add(new Integer(1))

set.add(null); --> cant add another null value continuous null values will be considered as one.

iterator:

Iterator it = set.iterator();

while(it.hasNext()){ --> checks if it has value to provide

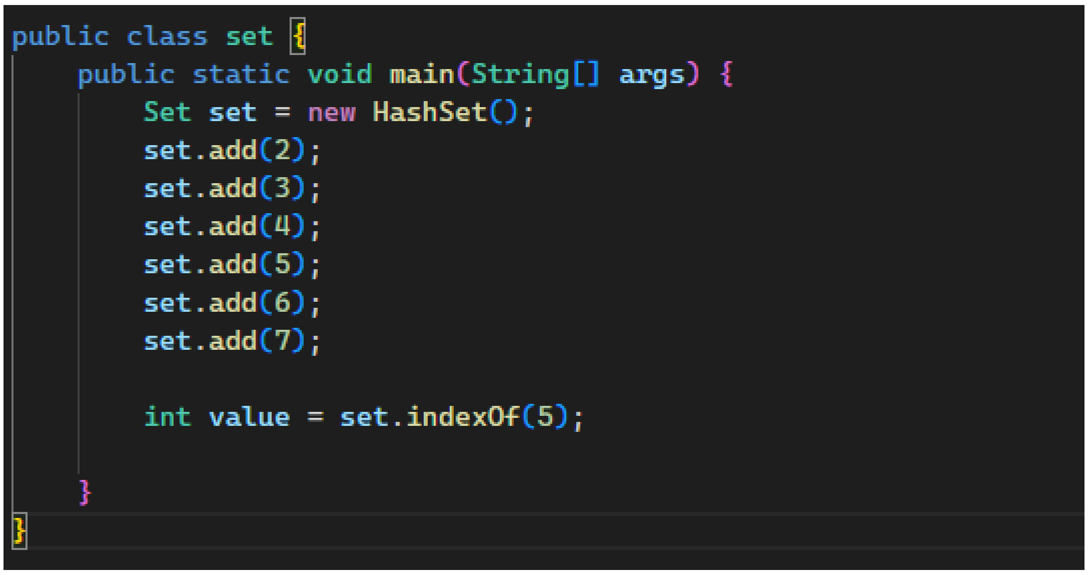
sout(it.next()); --> then it will print the value

}

generics --> to avoid datatype mismatch we go for generics

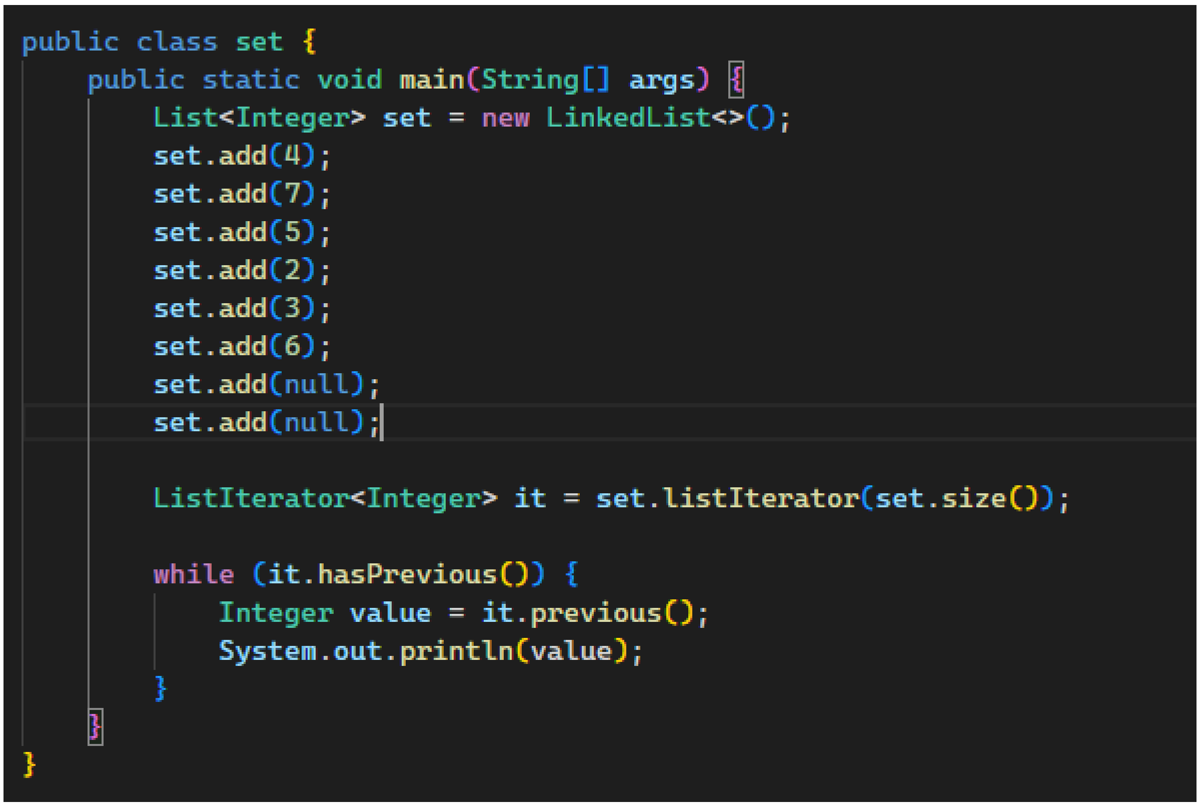
set<Integer> set = new HashSet<Integer>(); --> this will allow only integer data.

Set:



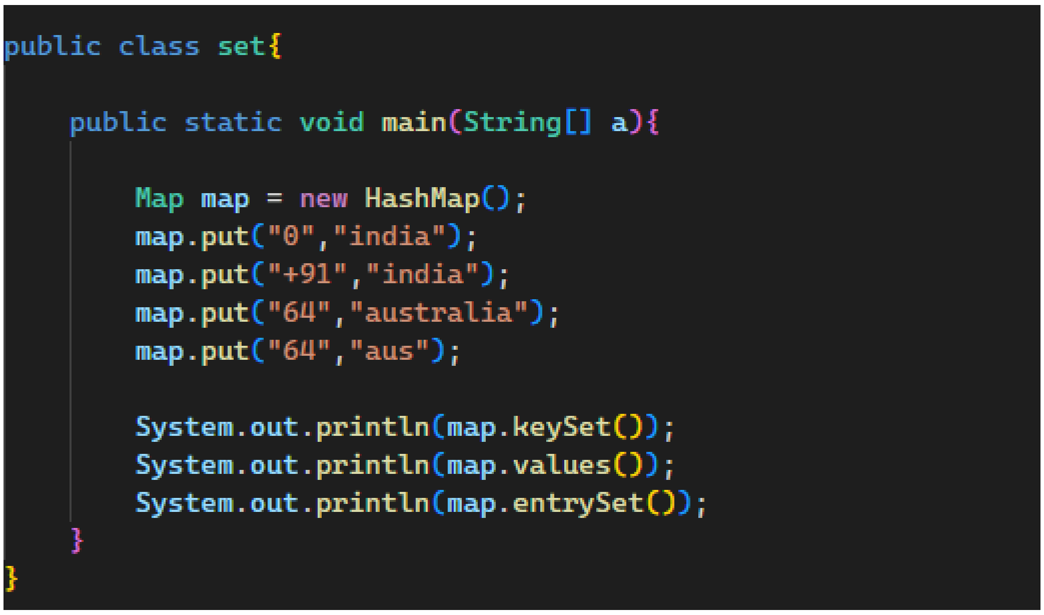
```
public class set {
    public static void main(String[] args) {
        Set set = new HashSet();
        set.add(2);
        set.add(3);
        set.add(4);
        set.add(5);
        set.add(6);
        set.add(7);

        int value = set.indexOf(5);

    }
}
```

LinkedList:



```
public class set {
    public static void main(String[] args) {
        List<Integer> set = new LinkedList<>();
        set.add(4);
        set.add(7);
        set.add(5);
        set.add(2);
        set.add(3);
        set.add(6);
        set.add(null);
        set.add(null);

        ListIterator<Integer> it = set.listIterator(set.size());

        while (it.hasPrevious()) {
            Integer value = it.previous();
            System.out.println(value);
        }
    }
}
```

Map:



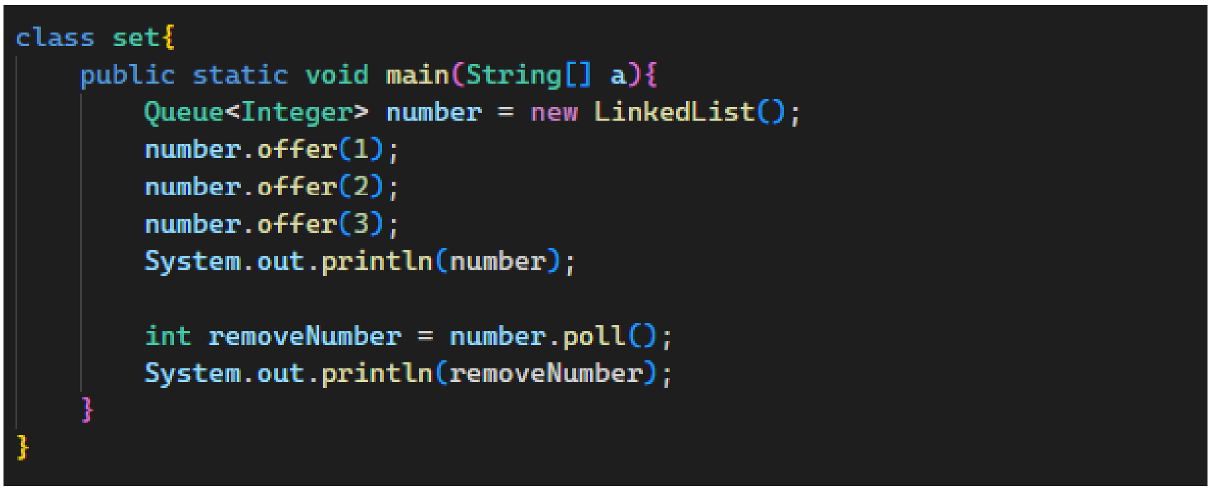
```
public class set{

    public static void main(String[] a){

        Map map = new HashMap();
        map.put("0","india");
        map.put("+91","india");
        map.put("64","australia");
        map.put("64","aus");

        System.out.println(map.keySet());
        System.out.println(map.values());
        System.out.println(map.entrySet());
    }
}
```

map.keySet(); --> print only keys

map.values(); --> print only values

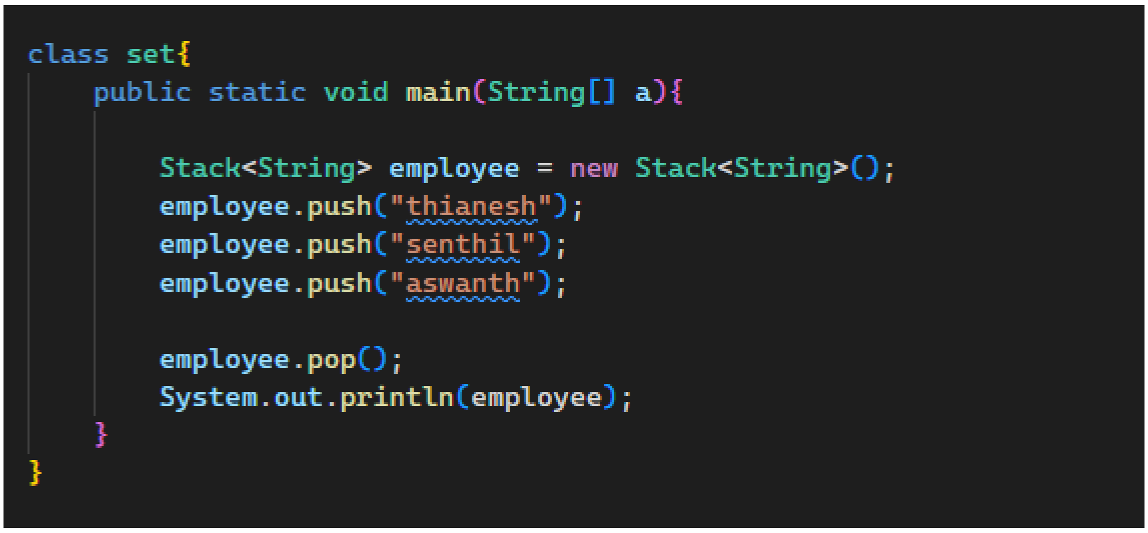
map.entrySet(); --> print both key and values - 0=india

Queue:



```
class set{
    public static void main(String[] a){
        Queue<Integer> number = new LinkedList();
        number.offer(1);
        number.offer(2);
        number.offer(3);
        System.out.println(number);

        int removeNumber = number.poll();
        System.out.println(removeNumber);
    }
}
```

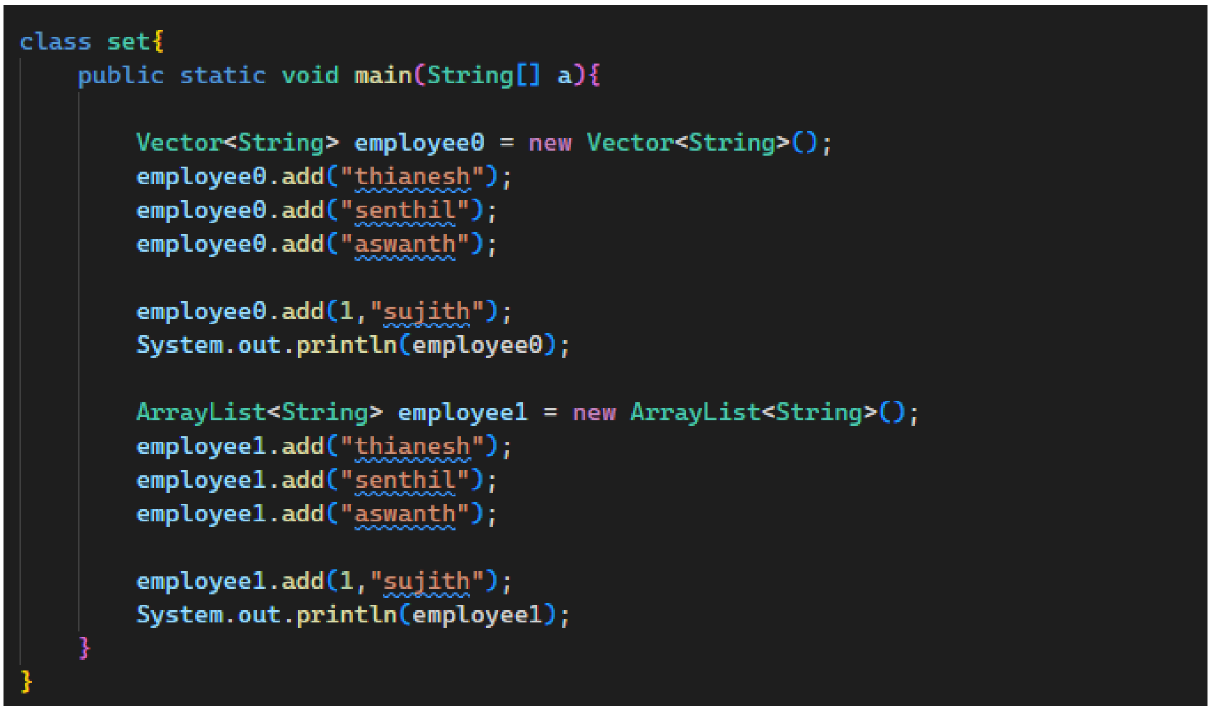
Stack:



```
class set{
    public static void main(String[] a){

        Stack<String> employee = new Stack<String>();
        employee.push("thianesh");
        employee.push("senthil");
        employee.push("aswanth");

        employee.pop();
        System.out.println(employee);
    }
}
```

Vector:

It is synchronous. It is a stopping code. If tried to access another change simultaneously it throw ConcurrentModificationException.



```
class set{
    public static void main(String[] a){

        Vector<String> employee0 = new Vector<String>();
        employee0.add("thianesh");
        employee0.add("senthil");
        employee0.add("aswanth");

        employee0.add(1,"sujith");
        System.out.println(employee0);

        ArrayList<String> employee1 = new ArrayList<String>();
        employee1.add("thianesh");
        employee1.add("senthil");
        employee1.add("aswanth");

        employee1.add(1,"sujith");
        System.out.println(employee1);
    }
}
```

employee.add(1,"sujith"); // adding values to specific index is possible

employee.remove("aswanth"); //to remove values

Task:

1.difference between priority queue and array deQueue?

A PriorityQueue is used when you need to process elements based on their priorities, while an ArrayDeque is used when you need to efficiently add and remove elements from both ends of a collection, maintaining the insertion order.