Day 9

Stack

- It is a linear data structure which is used to store elements in Last In First Out(LIFO) order.
- It is a List collection.
- It is sub class of Vector class.
- It is synchronized collection.
- It is introduced in jdk 1.0
- Methods of Stack
    1. public boolean empty()
    2. public E push(E item)
    3. public E peek()
    4. public E pop()
    5. public int search(Object o)

```
Stack<Integer> stk = new Stack<Integer>();
stk.push(10);
stk.push(20);
stk.push(30);

Integer element = null;
while( !stk.empty())
{
    element = stk.peek();
    System.out.println("Popped element is       :       "+element);
    stk.pop();
}
```

- Since it is synchronized slower in performance. For faster performance, we should use ArrayDeque class.

```
public static void main(String[] args)
{
    Deque<Integer> stk = new ArrayDeque<>();
    stk.push(10);
    stk.push(20);
    stk.push(30);

    Integer element = null;
    while( !stk.isEmpty())
    {
        element = stk.peek();
        System.out.println("Popped element is   :       "+element);
        stk.pop();
    }
}
```

LinkedList

- It is List collection which implicitly use doubly linked list.
- It implements List, Deque, Cloneable, Serializable.
- It is unsynchronized collection. Collections.synchronizedList() method is used to make LinkedList synchronized.

```
List list = Collections.synchronizedList(new LinkedList(...));
```

- This class is a member of the Java Collections Framework.
- It is introduced in jdk1.2.
- Instantiation:

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

Queue

- It is an interface declared in java.util package.
- It is sub interface of Collection.
- This interface is a member of the Java Collections Framework.
- It is introduced in jdk 1.5
- Queues typically, but do not necessarily, order elements in a FIFO (first-in-first-out) manner.
- Methods of Queue

1. boolean add(E e)
2. boolean offer(E e)
3. E remove()
4. E poll()
5. E element()
6. E peek()

- Each of these methods exists in two forms:
    - one throws an exception if the operation fails
    - Other returns a special value (either null or false, depending on the operation).
- Queue implementaion method-I

```
Queue<Integer> que = new ArrayDeque<>( );
que.add(10);
que.add(20);
que.add(30);

Integer element = null;
while( !que.isEmpty())
{
```

```
    element = que.element();
    System.out.println("Removed element is : "+element);
    que.remove();
}
```

- Queue implementaion method-II

```
Queue<Integer> que = new ArrayDeque<>( );
que.offer(10);
que.offer(20);
que.offer(30);

Integer element = null;
while( !que.isEmpty())
{
    element = que.peek();
    System.out.println("Removed element is : "+element);
    que.poll();
}
```

Deque

- It is sub interface of Queue interface.
- This interface is a member of the Java Collections Framework.
- It is introduced in jdk 1.6
- The name deque is short for "double ended queue" and is usually pronounced "deck".
- A linear collection that supports element insertion and removal at both ends.

Set

- It is sub interface of Collection interface.
- HashSet, LinkedHashSet and TreeSet implements Set collection. These are also called as Set collection.
- Set collection do not contain duplicate elements.
- This interface is a member of the Java Collections Framework.
- It is introduced in jdk 1.2
- Method names of Collection and Set interface are same.

TreeSet

- It is a Set collection.
- It's implementation is based on TreeMap<K,V>.
- It can contain duplicate element as well as null element.
- It is sorted collection.
- It is unsynchronized collection. Using synchronizedSortedSet method we can make it synchronized.

```
SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));
```

- This class is a member of the Java Collections Framework.
- It is introduced in jdk 1.2
- Note : If we want to manage elements of non final type inside TreeSet then non final type must implement Comparable interface or we should use Comparator implementation.
- Instantiation

```
Set<Integer> set = new TreeSet<>();
```

HashSet

- It is Set collection.
- It's implementation is based on Hashtable(actually HashMap).
- It can not contain duplicate elements but it can contain null elements
- It is unordered collection.
- It is unsynchronized collection. Using synchronizedSet method we can make it synchronized.

```
   Set s = Collections.synchronizedSet(new HashSet(...));
```

- This class is a member of the Java Collections Framework.
- It is introduced in jdk 1.2
- Note : If we want to manage elements of non final type inside HashSet then not final type should override equals() and hashCode() method.
- Instantiation

```
Set<Integer> set = new HashSet<>();
```

Searching

- It is a process of finding location(address / refernce / index) of element in collection.
- Searching Techniques:
    1. Linear Search
    2. Binary Search
    3. Hashing
- If we add element in array sequntially then number of comparisions to search element in array is also different.
- If we want to search every element in constant time then we should use Hashing.
- Hashing is a searching algorithm based on hashcode which is used to search element in constant time.
- Hashcode is logical integer number that can be generated by procecessing state of the object/instance.
- Hashcode is not a address or reference of instance. Hash function / method is responsible for generating Hashcode.
- According to hashing theory, to generate hashcode we should use prime number.

```
//Hash function / method
public static int getHashCode( int element )
{
    int result = 1;
    final int PRIME = 37;
    result = result * element + PRIME * element;
    return result;
}
```

- If state of object is same then we get same hashcode.

```
int x = 15; //Hashcode is 570
int y = 15; //Hashcode is 570
```

- Hashcode is required to generate index. In hashing index is called slot.
- By processing hashcode of object if we get same slot then it is called collision.
- Consider case I

```
int x = 239;    //Hashcode : 9082;  --> Slot : 3 --> Collision
int y = 358;    //Hashcode : 13604  --> Slot : 3 --> Collision
```

- Consider case II

```
int x = 15; //Hashcode is 570   --> Slot : 3 --> Collision
int y = 15; //Hashcode is 570   --> Slot : 3 --> Collision
```

- If we want to avoid collision then we should use collision resolution techniques
    1. Seperate Chaining / Open Hashing
    2. Open Addressing / Closed Hashing
        1. Linear Probing
        2. Quadratic Probing
        3. Double Hashing / Rehashing
- To avoid collision, we maintain one collection(Linked List, Tree) it is called bucket.
- Load Factor= (Number of buckets) / (Number of elements);

hashCode

- It is non final and native method of java.lang.Object class.
- Syntax: public int hashCode( );

LinkedHashSet

- It is a Set collection whose implementation is based on LinkedList and Hashtable.

- It is ordered collection.
- It can not contain duplicate elements but it can contain null element.
- It is unsynchronized collection. Using synchronizedSet method we can make it synchronized.

```
Set s = Collections.synchronizedSet(new LinkedHashSet(...));
```

- This class is a member of the Java Collections Framework.
- It is introduced in jdk 1.4
- Note : If we want to manage elements of non final type inside LinkedHashSet then not final type should override equals() and hashCode() method.

Dictionary

- It is a abstract class declared in java.util package.
- If we want to store data in key/value format then we should use sub class of Dictionary.
- Hashtable is sub class of Dictionary class.
- It is introduced in jdk 1.0.
- This class is obsolete / Deprecated. New implementations should implement the Map interface, rather than extending this class.
- Instantiation

```
Dictionary<Integer,String> d = new Hashtable<>();//Upcasting
```

- Methods of Dictionary<K,V> class:

1. public abstract V put(K key, V value);
2. public abstract V get(Object key)
3. public abstract V remove(Object key)
4. public abstract int size()
5. public abstract boolean isEmpty()
6. public abstract Enumeration keys()
7. public abstract Enumeration elements()

Map<K,V>

- It is a interface declared in java.util package.
- It represents key/value pair collection.
- This interface is a member of the Java Collections Framework but it doesnt extend Collection interface.
- It is introduced in jdk 1.2
- This interface takes the place of the Dictionary class.
- HashMap, TreeMap, Hashtable implements Map<K,V> interface. It is also called Map collection.
- Map collection can not contain duplicate keys but it can contain duplicate values.

Map.Entry<K,V>

- It is nested interface of Map<K,V> interface.
- In case of Map, entry is an instance whose type implements Map.Entry<K,V> interface.
- Abstract Methods of Map.Entry<K,V> interface

1. K getKey()
2. V getValue()
3. V setValue(V value)

Methods of Map<K,V> interface

1. V put(K key, V value);
2. void putAll(Map<? extends K,? extends V> m)
3. void clear()
4. boolean containsKey(Object key)
5. boolean containsValue(Object value)
6. int size()
7. boolean isEmpty()
8. V get(Object key)
9. V remove(Object key)
10. Set keySet()
11. Collection values()
12. Set<Map.Entry<K,V>> entrySet()

- Map is collection of entries where each entry(Key/Value pair) is instance Map.Entry<K,V> interface.

```
class MapEntry<K,V> implements Entry<K, V>
{
        private K key;
        private V value;
        public MapEntry()
        {       }
        public MapEntry(K key, V value)
        {
                this.key = key;
                this.value = value;
        }
        @Override
        public K getKey()
        {
                return this.key;
        }
        @Override
        public V getValue()
        {
                return this.value;
        }
        @Override
        public V setValue(V value)
        {
                this.value = value;
```

```
                return this.value;
        }
}
public class Program
{
        public static void main1(String[] args)
        {
                Entry<Integer, String> entry = new MapEntry<>( 1, "DAC");
                Integer key = entry.getKey();
                String value = entry.getValue();
                System.out.println(key + " " +value);
        }
}
```

Hashtable

- It is sub class of Dictionary class which implements Map<K,V> interface.
- It can not contain duplicate keys but it can contain duplicate values.
- It can not contain null key and null value.
- It is synchronized collection.
- It is a member of the Java Collections Framework.
- It is introduced in jdk1.0
- Instantiation:

```
Map<Integer, String> map = new Hashtable( );
```

- If we want to use instance non final type as a key then it should override equals() and hashCode() method.

HashMap<K,V>

- It is a Map<K,V> collection.
- It's implementation is based on Hashtable.
- It can not contain duplicate keys but it can contain duplicate values.
- It can contain null key and null value.
- It is unsynchronized collection. Using synchronizedMap() method we can make it synchronized.

```
   Map m = Collections.synchronizedMap(new HashMap(...));
```

- This class is a member of the Java Collections Framework.
- It is introduced in jdk 1.2
- Note : If we want to use instance non final type as a key then it should override equals() and hashCode() method.

LinkedHashMap<K,V>

- It is sub class of HashMap<K,V> class.

- Its implementation is based on LinkedList and Hashtable.
- It is ordered collection.
- It is unsuchronized collection. Using synchronizedMap() method we can make it synchronized.

```
Map m = Collections.synchronizedMap(new LinkedHashMap(...));
```

- This class is a member of the Java Collections Framework.
- It is introduced in jdk 1.4
- Note : If we want to use instance non final type as a key then it should override equals() and hashCode() method.

TreeMap<K,V>

- It is Map<K,V> collection whose implementation is based on Red Black Tree.
- It can not contain duplicate keys but it can contain duplicate values.
- It can not contain null key but it can contain null value.
- It is unsynchronized collection. Using synchronizedSortedMap() method we can make it synchronized.

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

- It maintains entries sorted according to the key.
- Note : If we want to use instance of non final type as a key then non final type should implement Comparable interface.
- This class is a member of the Java Collections Framework.
- It is introduced in jdk1.2
- Instantiation

```
Map<Integer,String> map = new TreeMap<>( );
```