Day 9

Collection Framework

Types of Iterator

1. Fail Fast Iterator
2. Fail Safe Iterator(Not Fail Fast)

Fail Fast Iterator

- During traversing, using collection reference, if we try to modify state of collection and if iterator do not allows us to do the same then such iterator is called "Fail Fast" Iterator. In this case JVM throws ConcurrentModificationException.

```
Vector<Integer> v = new Vector<>();
v.add(10);
v.add(20);
v.add(30);
v.add(40);
v.add(50);

Integer element = null;
Iterator<Integer> itr = v.iterator();
while( itr.hasNext())
{
    element = itr.next();
    System.out.print(element+"  ");
    if( element == 50 )
        v.add(60); //ConcurrentModificationException
}
```

Fail safe Iterator

- During traversing, if itrator allows us to do changes in underlying collection then such iterator is called fail safe iterator.

```
Vector<Integer> v = new Vector<>();
v.add(10);
v.add(20);
v.add(30);
v.add(40);
v.add(50);

Integer element = null;
Enumeration<Integer> e = v.elements();
```

```
while( e.hasMoreElements())
{
    element = e.nextElement();
    System.out.print(element+" ");
    if( element == 50 )
        v.add(60); //OK
}
```

Stack

- It is linear data structure which is used to manage elements in Last In First Out order.
- It is sub class of Vector class.
- It is synchronized collection.
- It is List Collection.
- Methods of Stack class
    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();
    element = stk.pop();
    System.out.println("Popped element is : "+element);
}
```

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

```
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 a List collection.
- It implements List, Deque, Cloneable and Serializable interface.
- Its implementation is depends on Doubly linked list.
- It is unsynchronized collection. Using Collections.synchronizedList() method, we can make it synchronized.

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

- It is introduced in jdk 1.2.
- Note : If we want to manage elements of non-final type inside LinkedList then non final type should override "equals" method.
- Instantiation

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

Queue

- It is interface declared in java.util package.
- It is sub interface of Collection interface.
- It is introduced in jdk 1.5
- Option 1

```
Queue<Integer> que = new ArrayDeque<>();
que.add(10);
que.add(20);
que.add(30);
Integer ele = null;
while( !que.isEmpty())
{
    ele = que.element();
    System.out.println("Removed element is : "+ele);
    que.remove();
}
```

- Option 2

```
public static void main(String[] args)
{
    Queue<Integer> que = new ArrayDeque<>();
    que.offer(10);
    que.offer(20);
    que.offer(30);
```

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

Deque

- It is usually pronounced "deck".
- It is sub interface of Queue.
- It is introduced in jdk 1.6
- If we want to perform operations from bidirection then we should use Deque interface.

Set

- It is sub interface of java.util.Collection interface.
- HashSet, LinkedHashSet, TreeSet etc. implements Set interface. It is also called as Set collection.
- Set collections do not contain duplicate elements.
- It is introduced in jdk 1.2

TreeSet

- It is Set collection.
- It can not contain duplicate element as well as null element.
- It is sorted collection.
- Its implementation is based on TreeMap
- It is unsynchronized collection. Using "Collections.synchronizedSortedSet()" method we can make it synchronized.

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

- It is introduced in jdk 1.2
- Note : If we want to manage elements of non final type inside TreeSet then non final type should implement Comparable interface.
- Instantiation

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

Searching

- It is process of finding location(index/address/reference) of element inside collection.

- Commonly used searching techniques are:
    1. Linear / Sequential Search
    2. Binary Search
    3. Hashing
- In array, elements are stored sequentially. Hence time required to earch every element is different.
- Hashing is a searching algorithm which is used to search element in constant time(faster searching).
- In case array, if we know index of element then we can locate it very fast.
- Hashing technique is based on "hashcode".
- Hashcode is not a reference or address of the object rather it is a logical integer number that can be generated by processing state of the object.
- Generating hashcode is a job of hash function/method.
- Generally hashcode is generated using prime number.

```
//Hash Method
private static int getHashCode(int data)
{
    int result = 1;
    final int PRIME = 31;
    result = result * data + PRIME * data;
    return result;
}
```

- If state of object/instance is same then we will get same hashcode.
- In hashing, index is called slot.
- Hashcode is required to generate slot.
- If state of objects are same then their hashcode and slot will be same.
- By processing state of two different object's , if we get same slot then it is called collision.
- Collision resolution techniques:
    1. Seperate Chaining / Open Hashing
    2. Open Addressing / Close Hashing
        1. Linear Probing
        2. Quadratic Probing
        3. Double Hashing / Rehashing
- Collection(LinkedList/Tree) maintained per slot is called bucket.
- Load Factor = (Count of bucket / Total elements);
- In hashcode based collection, if we want manage elements of non final type then refernce type should override equals() and hashcode() method.
- hashCode() is non final method of java.lang.Object class.
- Syntax: public native int hashCode( );
- On the basis of state of the object, we want to generated hashcode then we should ovveride hashCode() method in sub class.

HashSet

- It Set Collection.
- It can not contain duplicate elements but it can contain null element.

- It's implementation is based on HashTable.
- It is unordered collection.
- It is unsynchronized collection. Using Collections.synchronizedSet() method, we can make it synchronized.
- It is introduced in jdk 1.2
- Note : If we want to manage elements of non final type inside HashSet then non final type should override equals and hashCode() method.
- Instantiation:

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

LinkedHashSet

- It is sub class of HashSet class.
- Its implementation is based on linked list and Hashtable.
- It is ordered collection.
- It is unsynchronized collection. Using Collections.synchronizedSet() method we can make it synchronized.

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

- It is introduced in jdk 1.4
- It can not contain duplicate element but it can contain null element.

Dictionary<K,V>

- It is abstract class declared in java.util package.
- It is super class of Hashtable.
- It is used to store data in key/value pair format.
- It is not a part of collection framework
- It is introduced in jdk 1.0
- Methods:

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

- Implementation of Dictionary is Obsolete.

Map<K,V>

- It is part of collection framework but it doesn't extend Collection interface.

- This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface.
- HashMap, Hashtable, TreeMap etc are Map collection's.
- Map collection stores data in key/value pair format.
- In map we can not insert duplicate keys but we can insert duplicate values.
- It is introduced in jdk 1.2
- Map.Entry<K,V> is nested interface of Map<K,V>.
- Following are abstract methods of Map.Entry interface.
    1. K getKey()
    2. V getValue()
    3. V setValue(V value)
- Abstract Methods of Map<K,V>

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

- An instance, whose type implements Map.Entry<K,V> interface is called enrty instance.

```java
class Pair<K,V> implements Entry<K, V>
{
        private K key;
        private V 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;
        }
}
```

```
class Program
{
    public static void main(String[] args)
    {
        Entry<Integer, String> e = new Pair<>();
        //Here pair instance is entry instance
    }
}
```

- Map is collection of entries where each entry contains key/value pair.

Hashtable

- It is Map<K,V> collection which extends Dictionary class.
- It can not contain duplicate keys but it can contain duplicate values.
- In hashtable, Key and value can not be null.
- It is synchronized collection.
- It is introduced in jdk 1.0
- In Hashtable, if we want to use instance non final type as key then it should override equals and hashCode method.