Day 20

Collection Framework

- Every value/data stored in data structure is called element.
- In java, data structure class is called collection.
- Framework is library of reusable classes/interafces that is used to develop applicaiton.
- Library of reusable data structure classes that is used to develop java application is called collection framework.
- Main purpose of collection framework is to manage data in RAM efficiently.
- Consider following Example:
    1. Person has-a birthdate
    2. Employee is a person
- In java, collection instance do not contain instances rather it contains reference of instances.
- If we want to use collection framework them we should import java.util package.

Iterable:

- It is a interface delared in java.lang package.
- All the collcetion classes implements Iterable interface hence we can traverse it using for each loop
- Methods of Iterable interface

1. Iterator iterator()
2. default Spliterator spliterator()
3. default void forEach(Consumer<? super T> action)

Collection

- Collection is interface declared in java.util package.
- It is sub interface of Iterable interface.
- It is root interface in collection framework interface hierarchy.
- Abstract Methods of Collection Interface

1. boolean add(E e)
2. boolean addAll(Collection<? extends E> c)
3. void clear()
4. boolean contains(Object o)
5. boolean containsAll(Collection<?> c)
6. boolean isEmpty()
7. boolean remove(Object o)
8. boolean removeAll(Collection<?> c)
9. boolean retainAll(Collection<?> c)
10. int size()
11. Object[] toArray()
12. T[] toArray(T[] a)

- Default methods of Collection interface

1. default Stream stream()
2. default Stream parallelStream()
3. default boolean removeIf(Predicate<? super E> filter)

List

- It is sub interface of java.util.Collection interface.
- It is ordered/sequential collection.
- ArrayList, Vector, Stack, LinkedList etc. implements List interface. It generally refered as "List collections".
- List collection can contain duplicate element as well multiple null elements.
- Using integer index, we can access elements from List collection.
- We can traverse elements of List collection using Iterator as well as ListIterator.
- It is introduced in jdk 1.2.
- Note: If we want to manage elements of non final type inside List collection then non final type should override "equals" method.
- Abstract methods of List Interface

1. void add(int index, E element)
2. boolean addAll(int index, Collection<? extends E> c)
3. E get(int index)
4. int indexOf(Object o)
5. int lastIndexOf(Object o)
6. ListIterator listIterator()
7. ListIterator listIterator(int index)
8. E remove(int index)
9. E set(int index, E element)
10. List subList(int fromIndex, int toIndex)

- Default methods of List interface

1. default void sort(Comparator<? super E> c)
2. default void replaceAll(UnaryOperator operator)

ArrayList

- It is resizable array.
- It implements List, RandomAccess, Cloneable, Serializable interfaces.
- It is List collection.
- It is unsynchronized collection. Using "Collections.synchronizedList" method, we can make it synchronized.

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

- Initial capacity of ArrayList is 10. If ArrayList is full then its capacity gets increased by half of its existing capacity.
- It is introduced in jdk 1.2

- Note: If we want to manage elements of non final type inside ArrayList then non final type should override "equals" method.
- Constructor(s) of ArrayList

1. public ArrayList()

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

2. public ArrayList(int initialCapacity)

```
ArrayList<Integer> list =
              new ArrayList<>(15);
List<Integer> list = new ArrayList<>(15);
Collection<Integer> list =
                new ArrayList<>(15)
```

3. public ArrayList(Collection<? extends E> c)

```
Collection<Integer> c = ArrayList<>();
List<Integer> list = new ArrayList<>(c);
```

```
Collection<Integer> c = Vector<>();
List<Integer> list = new ArrayList<>(c);
```

```
Collection<Integer> c = TreeSet<>();
List<Integer> list = new ArrayList<>(c);
```

```
Collection<Integer> c = ArrayDeque<>();
List<Integer> list = new ArrayList<>(c);
```

- Methods of ArrayList

1. public void ensureCapacity(int minCapacity)
2. protected void removeRange(int fromIndex, int toIndex)
3. public void trimToSize()

- Using illegal index, if we try to access element from any List collection then List methods throws IndexOutOfBounds Exception.

```
List<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
list.add(30);
Integer element = list.get(list.size());
//Output : IndexOutOfBoundsException
```

- If we want to sort elements of array then we should use Arrays.sort() method and to sort elements of List collection, we should use Collections.sort() method.

Vector

- It is resizable array.
- It implements List, RandomAccess, Cloneable, Serializable.
- It is List collection.
- It is synchronized collection.
- Default capacity of vector is 10. If vector is full then its capacity gets increased by its existing capacity.
- We can traverse elements of vector using Iterator, ListIterator as well as Enumeration.
- It is introduced in jdk 1.0.
- Note: If we want to manage elements of non final type inside Vector then non final type should override "equals" method.

Following classes are by default synchronized

1. Vector
2. Stack(Sub class of Vector)
3. Hashtable
4. Properties(Sub class of Hashtable)

Enumeration

- It is interface declared in java.util package.
- Methods of Enumeration I/F
    1. boolean hasMoreElements()
    2. E nextElement()
- It is used to traverse collection only in forward direction. During traversing, we can add, set or remove element from collection.
- It is introduced in jdk 1.0.
- "public Enumeration elements()" is a method of Vector class.

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

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

```
while( e.hasMoreElements())
{
    element = e.nextElement();
    System.out.println(element);
}
```

Iterator

- It is a interface declared in java.util package.
- It is used to traverse collection only in forward direction. During traversing, we can not add or set element but we can remove element from collection.
- Methods of Iterator

1. boolean hasNext()
2. E next()
3. default void remove()
4. default void forEachRemaining(Consumer<? super E> action)

- It is introduced in jdk 1.2

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

Integer element = null;
Iterator<Integer> itr = v.iterator();
while( itr.hasNext())
{
    element = itr.next();
    System.out.  println(element);
}
```

ListIterator

- It is subinterface of Iterator interface.
- It is used to traverse only List Collection in bidirection.
- During traversing we can add, set as well as remove element from collection.
- It is introduced in jdk 1.2
- Methods of ListIterator

1. boolean hasNext()
2. E next()
3. boolean hasPrevious()
4. E previous()
5. void add(E e)
6. void set(E e)
7. void remove()

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

Integer element = null;
    ListIterator<Integer> itr = v.listIterator();
while( itr.hasNext())
{
    element = itr.next();
    System.out.print(element+"  ");
}
System.out.println();
while( itr.hasPrevious())
{
    element = itr.previous();
    System.out.print(element+"  ");
}
```

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.