Day 10

Synchronized collection Classes:

```
1. Vector<E>
2. Stack<E>
3. Hashtable<E>
4. Properties
```

Vector

```
- It is resizable array.
- It is synchronized collection.
- It is a List collection.
- It implements following interfaces:
        1. java.util.List<E>
        2. java.util.RandomAccess
        3. java.lang.Cloneable
        4. java.io.Serializable
- Default capacity of Vecot<E> is 10. If Vector is full then its capacity
gets increased by its existing capacity.
- We can traverse elements of Vector<E> using Enumeration<E>, Iterator<E>
and ListIterator<E>.
- It is java since 1.0.
- If we want to manage elements of non final type in Vector then non final
type should override equals method.
- Constructors of Vector<E> class:
        - public Vector()
        - public Vector(int initialCapacity)
        - public Vector(int initialCapacity, int capacityIncrement)
        - public Vector(Collection<? extends E> c)
- Methods of Vector<E> class
        - public void addElement(E obj)
        - public int capacity()
        - public void copyInto(Object[] anArray)
        - public E elementAt(int index)
        - public Enumeration<E> elements()
        - public void ensureCapacity(int minCapacity)
        - public E firstElement()
        - public int indexOf(Object o, int index)
        - public void insertElementAt(E obj,int index)
        - public E lastElement()
        - public int lastIndexOf(Object o, int index)
        - public boolean removeElement(Object obj)
        - public void removeElementAt(int index)
        - protected void removeRange(int fromIndex, int toIndex)
        - public void setElementAt(E obj, int index)
        - public void setSize(int newSize)
        - public void trimToSize()
```

Enumeration

```
- Enumeration<E> is interface declared in java.util package.
- Methods of Enumeration interface:
        1. boolean hashMoreElements( );
        2. E nextElement( );
- It is introduced in jdk 1.0.
- It is used to traverse collection in forward direction only. Using
enumeration, we can not add, set or remove element from underlying
collection.
- If we want to get reference of vector then we should use "Enumeration<E>
elements()" method.
```

```
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.println(element);
}
```

Iterator

```
- It is interface declared in java.util package.
- Methods of java.util.Iterator<E> interface
        1. boolean hasNext();
        2. E next( );
        3. default void remove( );
        4. default void forEachRemaining(Consumer<? super E> c);
- It is introduced in jdk 1.2
- It is used to traverse collection in forward direction only. During
traversing, user iterator, we can not add or set element but remove
element from underlying collection.
```

> What is the difference between Enumeration and Iterator

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

ListIterator

```
- It is sub interface of Iterator interface.
- Methods of java.util.ListIterator<E> interface
        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()
        8. int previousIndex()
        9. int nextIndex()
- It is designed to traverse only List collections
- Using ListIterator<E> we can traverse list collection in bidirection.
- During traversing, using ListIterator<E> we can add, set and remove
element from underlying collection.
- It is introduced in jdk 1.2.
```

> What is the difference between Iterator and ListIterator?

```
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+"      ");
}
```

```
- According to C++ language, if we use any object as a pointer then such
object is called smart pointer.
- Iterator is a smart pointer that is used to traverse collection.
- Traversing is a process of visiting elements in collection.
```

Types of Iterator

```
1. Fail-Fast Iterator
2. Fail-Safe Iterator
```

Fail-Fast Iterator

```
- During traversing, using collection instance, if iterator do not allows
us to modify state of collection then such iterator is called "Fail-Fast"
iterator.
```

```
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.println(element);
        if( element == 50 )
                v.add(60);      //ConcurrentModificationException
}
```

Fail-Safe Iterator

```
- During traversing, using collection instance, if iterator allows us to
modify state of collection then such iterator is called "Fail-Safe"
iterator.
```

```
public static void main(String[] args)
{
	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.println(element);
		if( element == 50 )
			v.add(60);	//OK
	}
}
```

> What is the difference between ArrayList and Vector?

Stack

```
- It linear/sequential data structure/collection in which we can manage
elements in Last In First Out( LIFO )Order.
- It is sub class of vector
- It is synchronized collection.
- It is introduced in JDK 1.0
- Methods of Stack<E>
	1. public boolean empty()
	2. public E push(E item)
	3. public E peek()
	4. public E pop()
	5. public int search(Object o)
- Application of Stack
	1. To maintain F.A.R
	2. Expression conversion and evaluation
	3. To implement DFS algorithm, Stack is used
	4. To reverse string
	5. To convert Decimal to binary.
- Application of Stack
	- To show recent files
	- To implement undo-redo feature
	- To maintain call history, call log application use stack
	- To maintain email history, mailbox applcation use stack
	- To maintain Transaction history, bank application applcation use
stack
	- To maintain cart
```

LinkedList

```
- It is a class declared in java.util package.
- It implements following interfaces:
        1. java.util.List<E>
        2. java.util.Deque<E>
        3. java.lang.Cloneable
        4. java.io.Serializable
- Its implementation is based on Doubly linked list.
- It is list collection.
- It is unsyncronized collection. Using "Collections.synchronizedList()"
method, we can make it synchronized.
List list = Collections.synchronizedList(new LinkedList(...));
- It is introduced in jdk 1.2.
- If we want to manage elements of non final type in LinkedList<E> then
non final type should override "equals" method.
- Constructors of LinkedList<E> class
        1. public LinkedList()
        2. public LinkedList(Collection<? extends E> c)
```

> What is difference between ArrayList and LinkedList? What is difference between Array and ArrayList ?

Queue

```
- It is a interface declared in java.util package
- It is sub interface of Collection<E> interface.
- It is introduced in jdk 1.5
- Methods of java.util.Queue<E> interface
        1. boolean add(E e)
        2. E element()
        3. boolean offer(E e)
        4. E peek()
        5. E poll()
        6. E remove()
```

```
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();
}
```

```
public static void main(String[] args)
{
        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<E> interface
- It is introduced in jdk 1.6
- It allows us to perform insertion and deletion operation from both ends.
```

```
Deque<Integer> que = new ArrayDeque< >();
que.offerLast(5);
que.offerLast(20);
que.offerLast(30);
que.offerLast(40);
que.offerLast(500);
que.pollFirst();
que.offerFirst(10);
que.pollLast();
que.offerLast(50);
Integer element = null;
while( !que.isEmpty())
{
        element = que.peekLast();
        System.out.println("Removed element is : "+element);
        que.pollLast();
}
```

Set

```
- It is sub interface of java.util.Collection interface.
- HashSet<E>, LinkedHashSet<E> and TreeSet<E> implements Set<E> interface.
It is also called as Set collection.
- Set collection do not contain duplicate elements.
- Method names of Collection ans Set are same.
```

TreeSet

```
- It is Set collection.
- Its implementation is based on TreeMap.
- It doesn't contain duplicate element and null element
- It is sorted collection.
- It is unsynchronized collection. Using
"Collections.synchronizedSortedSet()" method we can make it synchronized.
- It is introduced in jdk1.2
- If we want to manage elements of non final type in TreeSet then non
final type should implement java.lang.Comparable interface.
- Instantiation:
        Set<Integer> set =  new TreeSet<>( );
```