Dt : 12/11/2022

define Spliterator<T>?

=>Spliterator<T> is an interface from java.util package and which is used to retrieve elements from Collection<E> objects and Array Objects.

syntax:

```
  Spliterator<BookDetails> sp = ob.spliterator();
```

define forEach() method?

=>forEach() method introduced by Java8 version and which is also used to retrieve elements from Collection<E> objects directly.

Method Signature:

```
  public default void forEach(java.util.function.Consumer<? super T>);
```

faq:

wt is the diff b/w

(i)forEachRemaining()

(ii)forEach()

=>forEachRemaining() method is used executed using Iterator<E> object or Spliterator<T> object.

=>forEach() method is executed directly on Collection<T> objects.

==

****imp***

2.List<E>:

=>List<E> organizes elements based on index values and can hold duplicate elements.

=>The following are some important methods of List<E>:

```
  public abstract int size();
  public abstract boolean isEmpty();
  public abstract boolean contains(java.lang.Object);
  public abstract boolean add(E);
  public abstract boolean remove(java.lang.Object);
  public abstract boolean containsAll(java.util.Collection<?>);
  public abstract boolean addAll(java.util.Collection<? extends E>);
  public abstract boolean addAll(int, java.util.Collection<? extends E>);
  public abstract boolean removeAll(java.util.Collection<?>);
  public abstract boolean retainAll(java.util.Collection<?>);
  public default void replaceAll(java.util.function.UnaryOperator<E>);
  public default void sort(java.util.Comparator<? super E>);
  public abstract void clear();

  public abstract E get(int);
  public abstract E set(int, E);
  public abstract void add(int, E);
  public abstract E remove(int);
```

public abstract int indexOf(java.lang.Object);

public abstract int lastIndexOf(java.lang.Object);

public abstract java.util.Iterator<E> iterator();

public abstract java.util.ListIterator<E> listIterator();

public default java.util.Spliterator<E> spliterator();

public abstract java.util.List<E> subList(int, int);

public abstract java.lang.Object[] toArray();

public abstract <T> T[] toArray(T[]);

=>The following are the implementation classes of List<E>

(a)ArrayList<E>

(b)LinkedList<E>

(c)Vector<E>

Ex-program : DemoList1.java

```
package maccess;
import java.util.*;
public class DemoList1{
     @SuppressWarnings("removal")
     public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        String name=null;
        List<Integer> ob = null;
        try(s;){
          try {
               while(true) {
                    System.out.println("****Choice*****");
System.out.println("1.ArrayList\n2.LinkedList\n3.Vector\n4.exit"
);
                    System.out.println("Enter the Choice:");
                    switch(s.nextInt()) {
                    case 1:
                         ob = new ArrayList<Integer>();
                         name="ArrayList";
                         break;
                    case 2:
                         ob = new LinkedList<Integer>();
                         name="LinkedList";
                         break;
                    case 3:
                         ob = new Vector<Integer>();
                         name="Vector";
                         break;
                    case 4:
                         System.out.println("Operations stopped
of List");
                         System.exit(0);
                         break;
                    default:
                         System.out.println("Invalid
Choice...");
                    }//end of switch
                    System.out.println("****Operations on
"+name+"****");
                    xyz:
                    while(true) {
                         System.out.println("****Choice****");

System.out.println("1.add\n2.remove\n3.add(index,E)\n4.remove(in
dex)\n5.get(index)\n6.set(index,E)\n7.exit");
                         System.out.println("Enter the
Choice:");
                         switch(s.nextInt()) {
                         case 1:
                              System.out.println("Enter the
ele:");
                              ob.add(new ~~Integer~~(s.nextInt()));
                              System.out.println(ob.toString());
                              break;
                         case 2:
                              if(ob.isEmpty()) {
```

```
                                        System.out.println("List is
empty...");
                                    }else {
                                        System.out.println("Enter the
ele to be removed:");
                                        if(ob.remove(new
~~Integer~~(s.nextInt()))) {
                                        System.out.println("Ele
removed Successfully..");

     System.out.println(ob.toString());
                                        }else {
                                        System.out.println("Element
not founded...");
                                        }
                                    }
                                    break;
                               case 3:
                                    if(ob.isEmpty()) {
                                        System.out.println("List is
empty...");
                                    }else {
                                        System.out.println("Enter the
index:");
                                        int index1 = s.nextInt();
                                        if(index1>=0 &&
index1<ob.size()) {

     System.out.println("Enter the ele:");
                                             ob.add(index1,new
~~Integer~~(s.nextInt()));
                                             System.out.println("Ele
added...");

     System.out.println(ob.toString());
                                        }else {

     System.out.println("Invalid index value...");
                                        }
                                    }
                                    break;
                               case 4:
                                    if(ob.isEmpty()) {
                                        System.out.println("List is
empty...");
                                    }else {
```

```
                                   System.out.println("Enter the
index:");
                                   int index2 = s.nextInt();
                                   if(index2>=0 &&
index2<ob.size()) {
                                        ob.remove(index2);
                                        System.out.println("Ele
removed...");

     System.out.println(ob.toString());
                                   }else {

     System.out.println("Invalid index value..");
                                   }
                              }
                              break;
                         case 5:
                              if(ob.isEmpty()) {
                                   System.out.println("List is
empty...");
                              }else {
                                   System.out.println("Enter the
index:");
                                   int index3 = s.nextInt();
                                   if(index3>=0 &&
index3<ob.size()) {
                                        Integer ele =
(Integer)ob.get(index3);
                                        System.out.println("Ele
at index "+index3+" is "+ele.toString());

     System.out.println(ob.toString());
                                   }else {

     System.out.println("Invalid index...");
                                   }
                              }
                              break;
                         case 6:
                              if(ob.isEmpty()) {
                                   System.out.println("List is
empty...");
                              }else {
                                   System.out.println("Enter the
index:");
                                   int index4 = s.nextInt();
```

```
                                    if(index4>=0 &&
index4<ob.size()) {
     System.out.println("Enter the ele to be setted:");
                                         ob.set(index4, new
~~Integer~~(s.nextInt()));
     System.out.println(ob.toString());
                                    }else {
     System.out.println("Invalid index value..");
                                    }
                               }
                               break;
                          case 7:
                               System.out.println("Operations
Stopped on "+name);
                               break xyz;
                          default:
                               System.out.println("Invalid
Choice...");
                          }//end of switch
                     }//end of while

                }//end of loop
           }catch(Exception e) {e.printStackTrace();}
         }//end of try
     }
}
```

==

Assignment:

Construct application to perform the following operations on Product details?

1.addProduct

2.removeProduct ===>based on code

3.addProduct(index,E)

4.removeProduct(index)===>based on code

5.getProduct(index)===>based on code

6.setProduct(index,E)

7.exit

===

(a)ArrayList<E>:

=>ArrayList<E> organizes elements in sequence and which is NonSynchronized class.

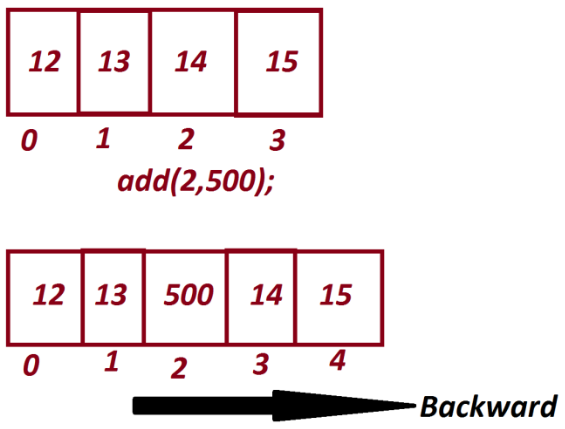
Limitation of ArrayList<E>:

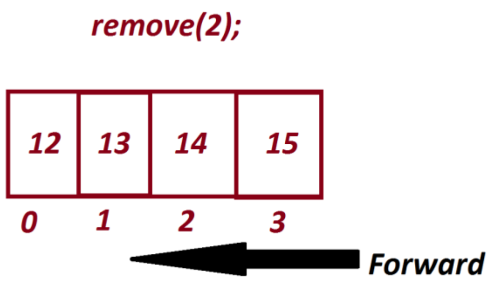
=>when we perform add() operation on ArrayList<E> the elements are moved backward,and when we perform remove() operation on ArrayList<E> the elements are moved forward,in this process if we perform more number of add() and remove() operations then performance of an application is degraded.

Note:

=>In realtime ArrayList<E> is used in applications where we have less number of add() and remove() operations.

=>This Limitation of ArrayList<E> can be overcomed using LinkedList<E>.





==