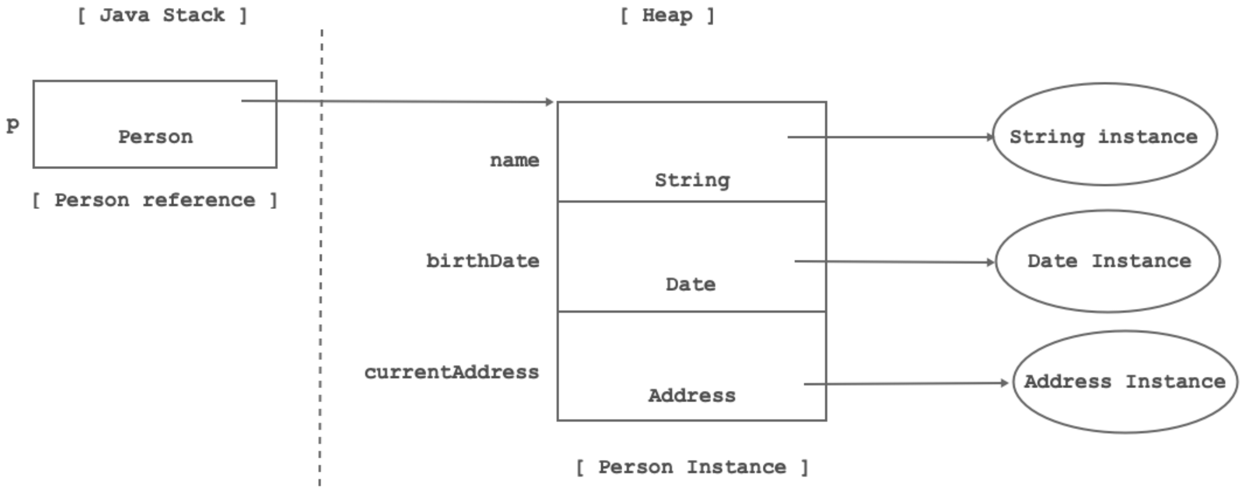
Day 21

Collection Framework

- Consider following example:

```
class Date{ }
class Addresss{ }
class Person{
  private String name = new String();
  private Date birthDate = new Date();
  private Address currentAddress = new Address();
}
class Program{
  public static void main(String[] args) {
    Person p = new Person( );
  }
}
```

- In Java, instance do not get space inside another instance. Rather instance contains reference of another instance.



Library

- In Java, .jar file is a library file.
- It can contain, menifest file, resources, packages.
- Package can contain sub package, interface, class, enum, exception, error, annotation types
- Example: rt.jar

Framework

- framework = collection of libraries + tools + rules/guidelines
- It is a development platform which contain reusable partial code on the top of it we can develop application.
- Examples:

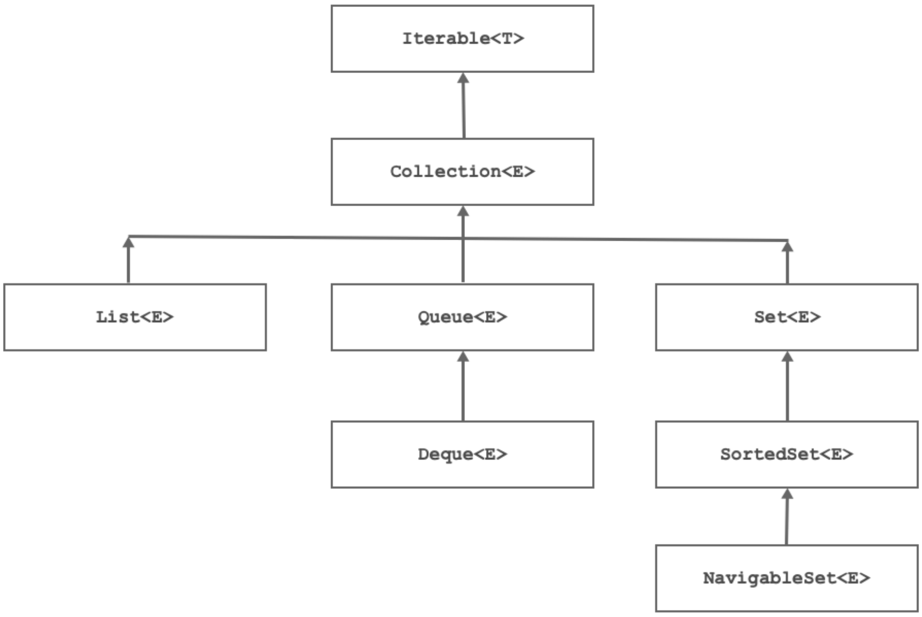
- JUnit: Unit testing framework which is used to write test case.
  - Apache Log4j2: Logging framework which is used to record activities.
  - AWT/Swing/Java-FX: GUI framework.
  - JNI: Framework to access native code
  - Struts: Readymade MVC based web application framework.
  - Hibernate: ORM based automatic persistence framework
  - Spring: Enterprise framework

Collection

- Any instance which contains multiple elements is called as collection.
- In java, data structure is also called as collection.

Collection Framework

- Collection framework is a library of data structure classes on the top of it we can develop Java application.
- In Java, collection framework talk about use not about implementation.
- In Java, when we use collection to store instance then it doesnt contain instance rather it contains reference of the instance.
- To use collection framework, we should import java.util package.



[Collection Framework Interface Hierarchy]

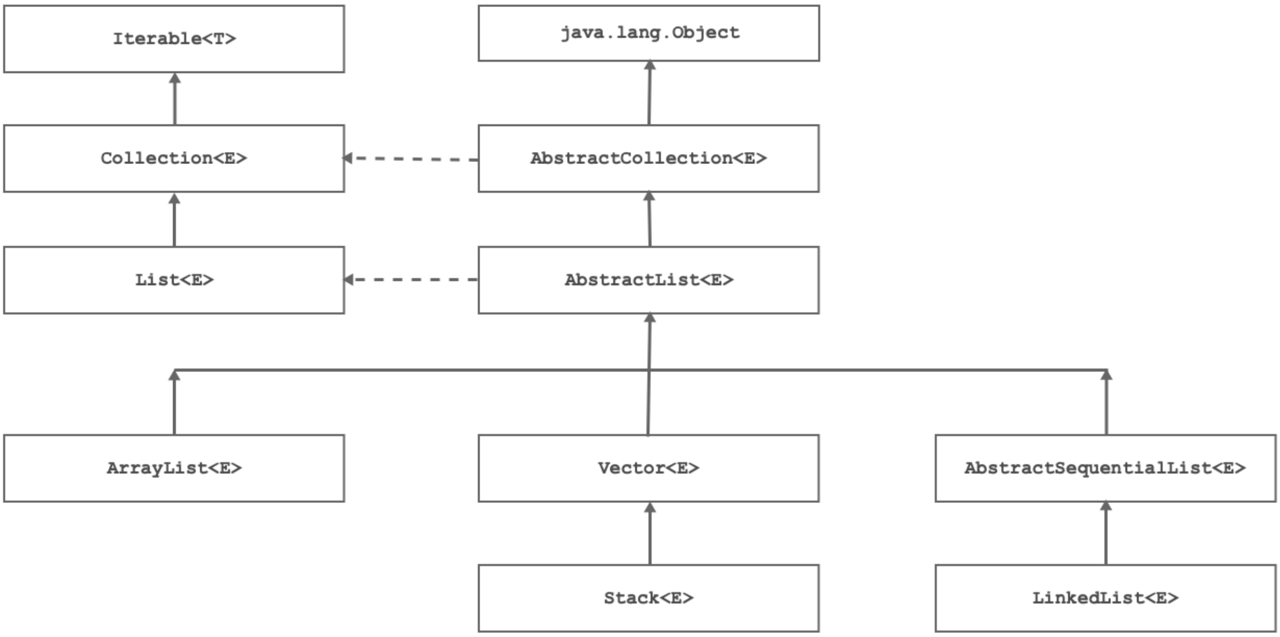
Iterable

- It is interface declared in java.lang package.
- It is introduced in jDK 1.5.
- Implementing this interface allows an object to be the target of the "for-each loop" statement.
- Methods:
  - Iterator iterator()
  - default Spliterator spliterator()
  - default void forEach(Consumer<? super T> action)

Collection

- Reference: https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html
- Value stored inside any collection(Array, Stack, Queue, LinkedList etc.) is called as element.
- It is interface declared in java.util package.
- It is root interface in the collection framework interface hierarchy.
- The JDK does not provide any direct implementations of Collection interface.
- Direct implementation classes of Collection interface are AbstractList, AbstractQueue, AbstractSet.
- List, Queue, Set are sub interfaces of java.util.Collection interface.
- Abstract methods of java.util.Collection interface:
  - boolean add(E e)
  - boolean addAll(Collection<? extends E> c)
  - void clear()
  - boolean contains(Object o)
  - boolean containsAll(Collection<?> c)
  - boolean isEmpty()
  - boolean remove(Object o)
  - boolean removeAll(Collection<?> c)
  - boolean retainAll(Collection<?> c)
  - int size()
  - Object[] toArray()
  - T[] toArray(T[] a)
- Default methods of java.util.Collection interface:
  - default Stream stream()
  - default Stream parallelStream()
  - default boolean removeIf(Predicate<? super E> filter)

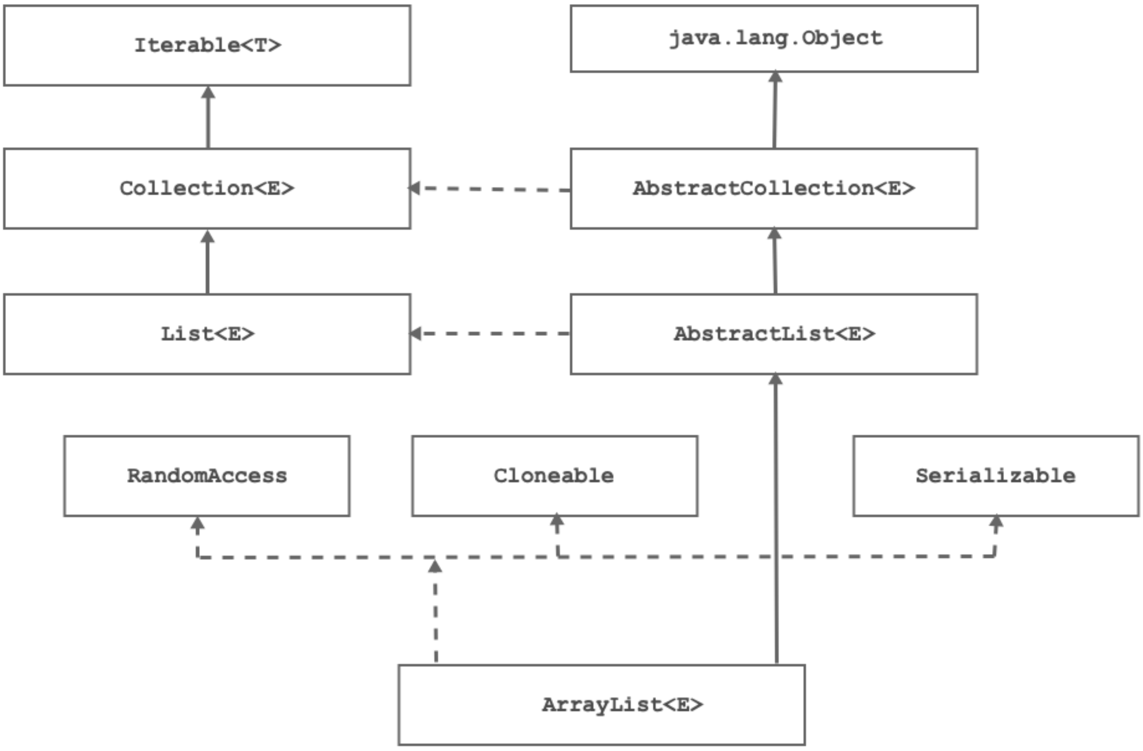
List



- This interface is a member of the Java Collections Framework and introduced in JDK 1.2

- It is sub interface of Collection interface. It means that all the methods of Collection interafce will be inherited into List interface.
- Direct implementation classes of List interfaces are AbstractList, ArrayList, Vector, Stack, LinkedList. These collection classes are called as List collections.
- Inside List collection we can store data in sequential fashion.
- We can store duplicate elements inside any List collection.
- We can store multiple null values inside List collection.
- With the help of integer index, we can access elements from List collection.
- We can traverse elements of any List collection using Iterator as well as ListIterator.
- This interface is a member of the Java Collections Framework.
- Abstract methods of java.util.List interface:
    - void add(int index, E element)
    - boolean addAll(int index, Collection<? extends E> c)
    - E remove(int index)
    - E get(int index)
    - E set(int index, E element)
    - int indexOf(Object o)
    - int lastIndexOf(Object o)
    - ListIterator listIterator()
    - ListIterator listIterator(int index)
    - List subList(int fromIndex, int toIndex)
- Default methods of java.util.List interface:
    - default void sort(Comparator<? super E> c)
    - default void replaceAll(UnaryOperator operator)
- Note: If we want to manage elements of non final type inside any List collection then we should override at least equals methods inside non final type.

ArrayList



- Array is collection of fixed elements. ArrayList is resizeable array.
- Implementation of ArrayList is based of array.
- ArrayList is List collection.
- Since ArrayList is List collection we can store elements sequentially.
- Since ArrayList is List collection, we can store duplicate elements as well as null elements inside ArrayList.
- Since ArrayList is List collection, we can access its elements using integer index.
- Since ArrayList is List collection, we can traverse its elements using Iterator as well as ListIterator.
- ArrayList implementation is unsynchronized. Using Collections.synchronizedList() method we can make it synchronized.
- If ArrayList is full then its capacity gets increased by half of existing capacity.
- This class is a member of the Java Collections Framework and introduced in JDK 1.2.
- Constructor Summary of ArrayList class:
  - public ArrayList()

    ```
    ArrayList<Intger> list = new ArrayList();
    ```

  - public ArrayList(int initialCapacity)

    ```
    ArrayList<Intger> list = new ArrayList( 15 );
    ```

  - public ArrayList(Collection<? extends E> c)

    ```
    Collection<Integer> c = new ArrayList<>( );
    c.add( 10 );
    c.add( 20 );
    c.add( 30 );

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

- Method Summary of ArrayList class:
  - public void ensureCapacity(int minCapacity)
  - protected void removeRange(int fromIndex, int toIndex)
  - public void trimToSize()
- Instantiation:

```
public static void main(String[] args){
    Collection<Integer> collection = new ArrayList<>(); //OK:
Upcasting
    List<Integer> list = new ArrayList<>(); //OK: Upcasting
    ArrayList<Integer> arrayList = new ArrayList<>();   //OK
}
```

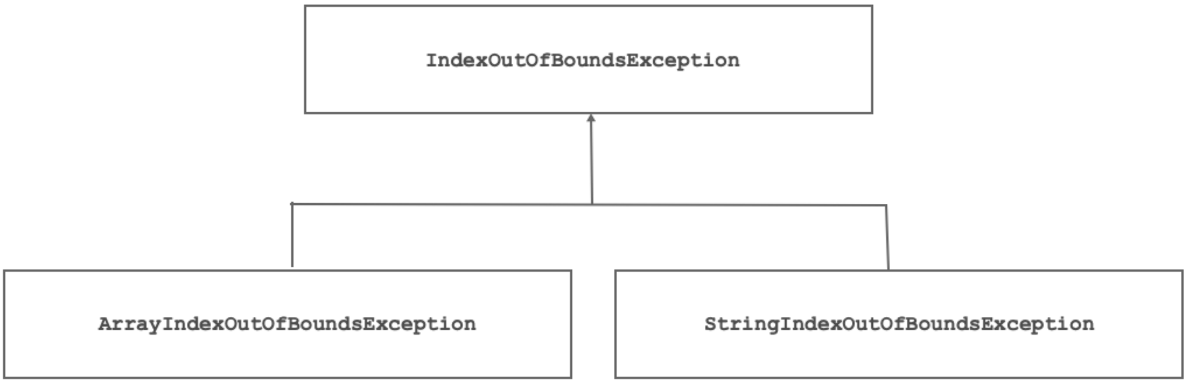
- How to add single element inside ArrayList?

```
public static void main(String[] args) {
    List<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(40);
    list.add(50);
    list.add(2, 30);
    System.out.println( list.toString());
}
```

```
public static List<Integer> getList( ){
    List<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(30);
    list.add(40);
    list.add(50);
    return list;
}
public static void main(String[] args) {
    List<Integer> list = Program.getList();
    System.out.println( list.toString()); //[10, 20, 30, 40, 50]
}
```

```
public static List<Integer> getList( ){
    List<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(30);
    list.add(40);
    list.add(50);
    return list;
}
public static void main(String[] args) {
    List<Integer> list = Program.getList();
    Integer element = null;
    for( int index = 0; index < list.size(); ++ index ) {
        element = list.get( index );
        System.out.println(element);
    }
}
```

```
    public static List<Integer> getList( ){
    List<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(30);
    list.add(40);
    list.add(50);
    return list;
  }
  public static void main(String[] args) {
    int[] arr = new int[ ] { 10, 20, 30 };
    //int element = arr[ arr.length ];
//ArrayIndexOutOfBoundsException

    String str = "CDAC";
    //char ch = str.charAt(str.length());
//StringIndexOutOfBoundsException

    List<Integer> list = Program.getList();
    Integer element = list.get( list.size() );
//IndexOutOfBoundsException
  }
```


IndexOutOfBoundsException
ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsException

```
    public static List<Integer> getList( ){
    List<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(30);
    list.add(40);
    list.add(50);
    return list;
  }
  public static void main(String[] args) {
    List<Integer> list = Program.getList();
    Integer element = null;
    Iterator<Integer> itr = list.iterator();
    while( itr.hasNext()) {
      element = itr.next();
      System.out.println(element);
    }
  }
```

```
    public static List<Integer> getList( ){
    List<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(30);
    list.add(40);
    list.add(50);
    return list;
  }
  public static void main(String[] args) {
    List<Integer> list = Program.getList();
    for( Integer element : list )
      System.out.println( element );
  }
```

```
    public static void main(String[] args) {
    List<Integer> list = Program.getList();
    /* Consumer<Integer> action = System.out::println;
    list.forEach(action); */
    list.forEach( System.out::println );
  }
```

```
    public static List<Integer> getList( ){
    List<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(30);
    list.add(40);
    list.add(50);
    return list;
  }
  public static void main(String[] args) {
    List<Integer> list = Program.getList();
    ListIterator<Integer> itr = list.listIterator();
    Integer element = null;
    while( itr.hasNext()) {
      element = itr.next();
      System.out.print( element+"   ");
    }
    System.out.println();
    while( itr.hasPrevious()) {
      element = itr.previous();
      System.out.print( element+"   ");
    }
  }
```

```
//Object[] elementData;
private static int capacity(List<Integer> list) throws Exception{
    Class<?> c = list.getClass();
    Field field = c.getDeclaredField("elementData");
    field.setAccessible(true);
    Object[] elementData = (Object[]) field.get(list);
    return elementData.length;
}
public static void main(String[] args) {
    try {
        List<Integer> list = Program.getList();
        System.out.println("Size        :   "+list.size()); //5

        int capacity = Program.capacity( list );
        System.out.println("Capacity    :   "+capacity);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

- How to add multiple elements inside ArrayList?

```
public static void main(String[] args){
    Collection<Integer> collection = new ArrayList<>();
    collection.add(30);
    collection.add(40);
    collection.add(50);

    //List<Integer> list = new ArrayList<>( collection );   //OK
    List<Integer> list = new ArrayList<>(  );
    list.add(10);
    list.add(20);
    list.addAll(collection);
    System.out.println(list);
}
```

```
public static void main(String[] args) {
    Collection<Integer> collection = new ArrayList<>();
    collection.add(30);
    collection.add(40);
    collection.add(50);

    List<Integer> list = new ArrayList<>(  );
    list.add(10);
    list.add(20);
    list.add(60);
    list.add(70);
    list.addAll(2, collection);
    System.out.println(list);
}
```

- How will you search single element inside ArrayList?

```
    public static List<Integer> getList( ){
    List<Integer> list = new ArrayList<>();
    for( int count = 1; count <= 10; ++ count  )
      list.add( count * 10 );
    return list;
  }
  public static void main(String[] args) {
    List<Integer> list = Program.getList(); //[10, 20, 30, 40, 50, 60,
70, 80, 90, 100]
    Integer key = new Integer(500);
    if( list.contains(key)) {
      int index = list.indexOf(key);
      System.out.println( key+" found at index : "+index);
    }else
      System.out.println(key+" not found.");
  }
```

- How will you search and remove multiple elements

```
    public static List<Integer> getList( ){
    List<Integer> list = new ArrayList<>();
    for( int count = 1; count <= 10; ++ count  )
      list.add( count * 10 );
    return list;
  }
    public static void main(String[] args) {
    List<Integer> list = Program.getList(); //[10, 20, 30, 40, 50, 60,
70, 80, 90, 100]
    Collection<Integer> keys = new ArrayList<>( );
    keys.add(30);
    keys.add(50);
    keys.add(70);
  if( list.containsAll(keys)) {
   list.removeAll(keys); //[10, 20, 40, 60, 80, 90, 100]
   //list.retainAll(keys);   //[30, 50, 70]
   System.out.println( list );
  }else
   System.out.println(keys+" not found.");
 }
```

- How will you search and remove single element from ArrayList?

```
  public static List<Integer> getList( ){
  List<Integer> list = new ArrayList<>();
  for( int count = 1; count <= 10; ++ count   )
   list.add( count * 10 );
  return list;
 }
 public static void main(String[] args) {
  List<Integer> list = Program.getList(); //[10, 20, 30, 40, 50, 60,
70, 80, 90, 100]
  Integer key = new Integer(50);
  if( list.contains(key)) {
   //list.remove(key);
   int index = list.indexOf(key);
   list.remove(index);
   System.out.println( list );   //[10, 20, 30, 40, 60, 70, 80, 90,
100]
  }else
   System.out.println(key+" not found.");
 }
```

- How will you sort ArrayList?

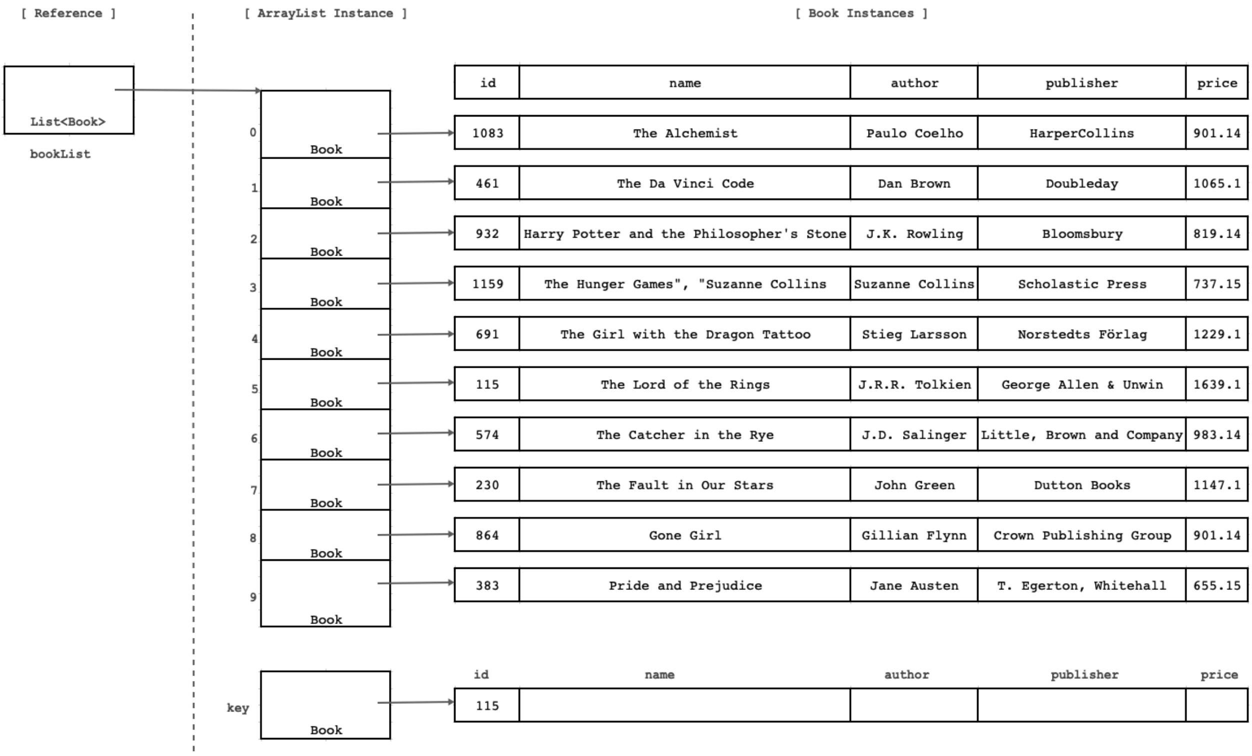
```
  public static void main(String[] args) {
  List<Integer> list = new ArrayList<>();
  list.add(50);
  list.add(10);
  list.add(30);
  list.add(20);
  list.add(40);

  System.out.println(list);   //[50, 10, 30, 20, 40]
  //Collections.sort( list );
  list.sort(null);
  System.out.println(list);   //[10, 20, 30, 40, 50]
 }
```

- How will you convert ArrayList into array?

```
  public static void main(String[] args) {
    List<Integer> list = new ArrayList<>();
    list.add(50);
    list.add(10);
    list.add(30);
    list.add(20);
    list.add(40);

    //Object[] arr = list.toArray();

    Integer[] arr = new Integer[ list.size() ];
    list.toArray(arr);

    System.out.println( Arrays.toString(arr));  //[50, 10, 30, 20, 40]
  }
```

- Using Arrays.asList() method

```
    public static void main(String[] args) {
    List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );
    System.out.println( list.getClass().getName());
//java.util.Arrays$ArrayList
    System.out.println( list ); //[10, 20, 30, 40, 50]
  }
```



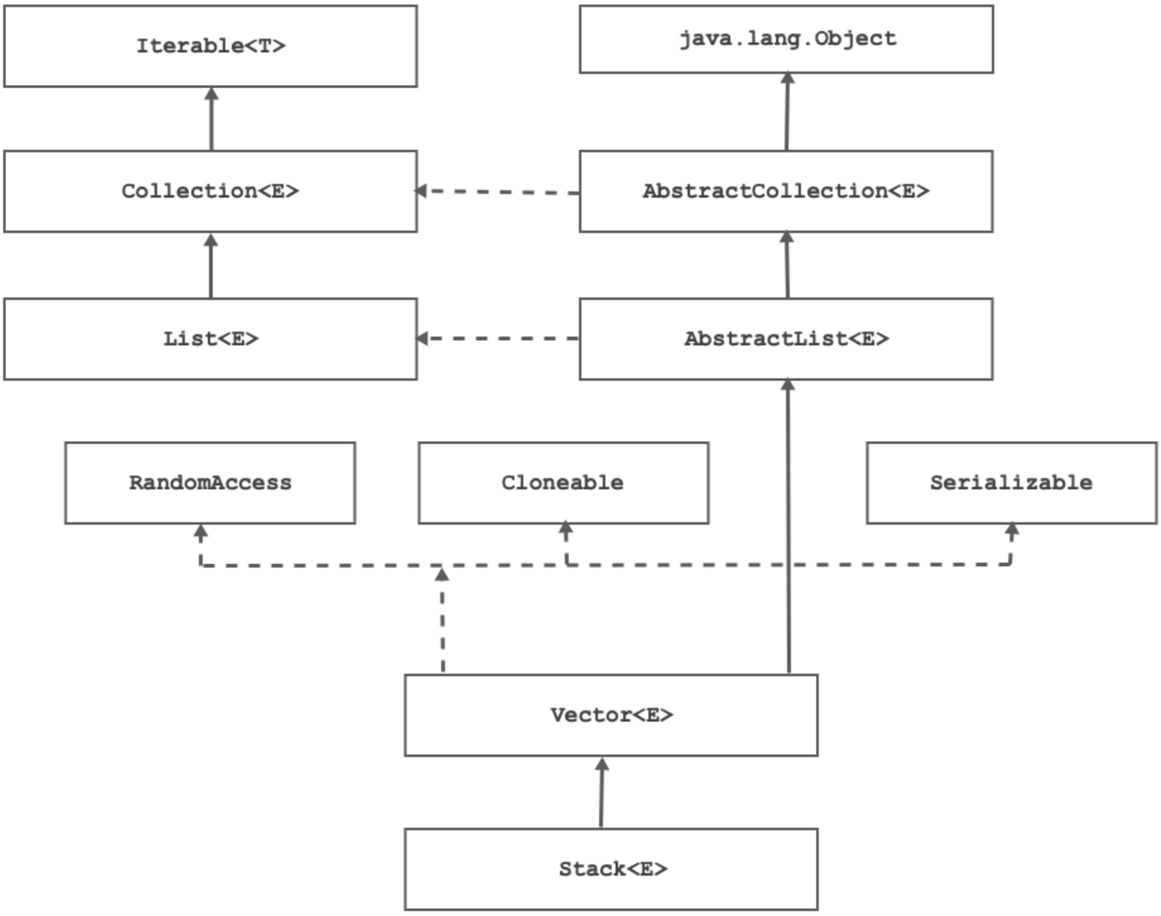
id	name	author	publisher	price
1083	The Alchemist	Paulo Coelho	HarperCollins	901.14
461	The Da Vinci Code	Dan Brown	Doubleday	1065.1
932	Harry Potter and the Philosopher's Stone	J.K. Rowling	Bloomsbury	819.14
1159	The Hunger Games", "Suzanne Collins	Suzanne Collins	Scholastic Press	737.15
691	The Girl with the Dragon Tattoo	Stieg Larsson	Norstedts Förlag	1229.1
115	The Lord of the Rings	J.R.R. Tolkien	George Allen & Unwin	1639.1
574	The Catcher in the Rye	J.D. Salinger	Little, Brown and Company	983.14
230	The Fault in Our Stars	John Green	Dutton Books	1147.1
864	Gone Girl	Gillian Flynn	Crown Publishing Group	901.14
383	Pride and Prejudice	Jane Austen	T. Egerton, Whitehall	655.15

id	name	author	publisher	price
115				

Which collection classes are by default synchronized in Java?

- java.util.Vector
- java.util.Stack
- java.util.Hashtable
- java.util.Properties

Vector



- Vector is a class declared in java.util package.
- Vector is List collection whose implementation is based on array.
- Since Vector is List collection, it is ordered/sequential collection.
- Since Vector is List collection, it can contain duplicate elements as well as null elements
- Since Vector is List collection, we can traverse its elements using Iterator as well as ListIterator.
- We can traverse elements of Vector using java.util.Enumeration , java.util.Iterator as well as ListIterator.
- Vector is Synchronized collection.
- Default capacity is 10 elements. Once Vector is full it gets double capacity.
- It was introduced in JDK 1.0. Hence it is also called as legacy class.

Travsering using Enumeration

- Enumeration is interface declared in java.util package.

- It was introduced in JDK 1.0.

- Methods of Enumeration I/F:

  - boolean hasMoreElements()
  - E nextElement()

- Using Enumeration we can traverse limited collections. For Example: Vector, Hashtable etc.

- Using Enumeration, we can traverse collection only forward direction. During traversing we can not add, set or remove elements from underlying collection.

```
   public static void main(String[] args){
   Vector<Integer> v = new Vector<>();
   for( int count = 1; count <= 10; ++ count )
     v.add(count);

   Integer element = null;
   Enumeration<Integer> e = v.elements() ;
   while( e.hasMoreElements()) {
     element = e.nextElement();
     System.out.println(element);
   }
 }
```

Travsering using Iterator

- Iterator is interface declared in java.util package.

- This interface is a member of the Java Collections Framework.

- Methods of Iterator interface:

  - boolean hasNext()
  - boolean hasNext()
  - default void remove()
  - default void forEachRemaining(Consumer<? super E> action)

- Iterator takes the place of Enumeration in the Java Collections Framework. Iterators differ from enumerations in two ways:

  - Iterators allow the caller to remove elements from the underlying collection during the iteration.
  - Method names have been improved.

```
 public static void main(String[] args){
   Vector<Integer> v = new Vector<>();
   for( int count = 1; count <= 10; ++ count )
     v.add(count);

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

Travsering using ListIterator

- It is subinterface of Iterator interface which is declared in java.util package.
- We can use it to traverse only List collections(ArrayList, Vector, Stack, LinkedList etc.)
- We can use ListIterator to traverse collection in bidirection. During travsering, using iterator we can add/set/remove element from collection.
- Method Summary
  - void add(E e)
  - void set(E e)
  - void remove()
  - boolean hasNext()
  - E next()
  - boolean hasPrevious()
  - E previous()
  - int nextIndex()
  - int nextIndex()
- This interface is a member of the Java Collections Framework.
- It is introduced in JDK 1.2

```
   public static void main(String[] args){
   Vector<Integer> v = new Vector<>();
   for( int count = 1; count <= 10; ++ count )
    v.add(count);

   Integer element = null;
   ListIterator<Integer> itr = v.listIterator();
   //ListIterator<Integer> itr = v.listIterator( 4 );
   //ListIterator<Integer> itr = v.listIterator( v.size() );
   while( itr.hasNext()) {
    element = itr.next();
    System.out.print(element+"   ");
   }
   System.out.println();
   while( itr.hasPrevious()) {
    element = itr.previous();
    System.out.print(element+"   ");
   }
  }
```

What is the difference between Enumeration and Iterator

- Using Enumeration we can traverse collection only in forward direction. During traversing, using Enumeration, we can not add/set/remove element from underlying Collection. Using Iterator we can traverse collection only in forward direction. During traversing, using Iterator, we can not add/set element but we can remove element from underlying Collection.
- We can use Enumeration for few Collections only but we can use Iterator for any collection that implements Iterable interface.
- Enumeration method names are long bur Iterator methods names are short.
- Enumeration was introduced in JDK 1.0 whereas Iterator was introduced in JDK1.2.

What is the difference between Iterator and ListIterator

- Using Iterator we can traverse any Collection which implements Iterable interface but Using ListIterator we can traverse any List collection.
- Using Iterator we can traverse collection only in forward direction whereas using ListIterator we can traverse collection in bidirection.
- During traversing, using iterator, we can not add/set element from underlying collection but we can remove element. During traversing, using ListIterator, we can add/set/remove element from underlying collection.

What do you know about fail-fast and not fail-fast(i.e. fail-safe) iterator

or What do you know about ConcurrentModificationException?

- During traversing, without iterator, if we try to make changes in underlying collection and if we get ConcurrentModificationException then such iterator is called as fail-fast Iterator.

```
public static void main(String[] args){
    Vector<Integer> v = new Vector<>();
    for( int count = 1; count <= 10; ++ count )
        v.add(count);

    Integer element = null;
    Iterator<Integer> itr = v.iterator();
    while( itr.hasNext()) {
        element = itr.next();
        System.out.println(element);
        if( element == 10 )
            v.add(11);  //ConcurrentModificationException
    }
}
```

- During traversing, without iterator, if we try to make changes in underlying collection and if we do not get ConcurrentModificationException then such iterator is called as fail-safe Iterator. Such iterators works by creating copy of the Collection.

```
public static void main1(String[] args){
    Vector<Integer> v = new Vector<>();
```

```
        for( int count = 1; count <= 10; ++ count )
            v.add(count);

        Integer element = null;
        Enumeration<Integer> e = v.elements() ;
        while( e.hasMoreElements()) {
            element = e.nextElement();
            System.out.println(element);
            if( element == 10 )
                v.add(11);  //OK
        }
        System.out.println(v);
    }
```

What is the difference between ArrayList and Vector?

- Synchronization: ArrayList collection in unsynchronized whereas Vector is collection synchronized.
- Capacity: Incase of arrayList, capacity gets increased by half of existing capacity. In case of vector, capacity get increased by existing capacity.
- Traversing: We can traverse elements of ArrayList using Iterator and ListIterator whereas we can traverse elements of vector using Enumeration, Iterator and ListIterator.
- Legacy: ArrayList collection is introduced in JDK 1.2 whereas Vector collection is introduced in JDK 1.0.

Stack

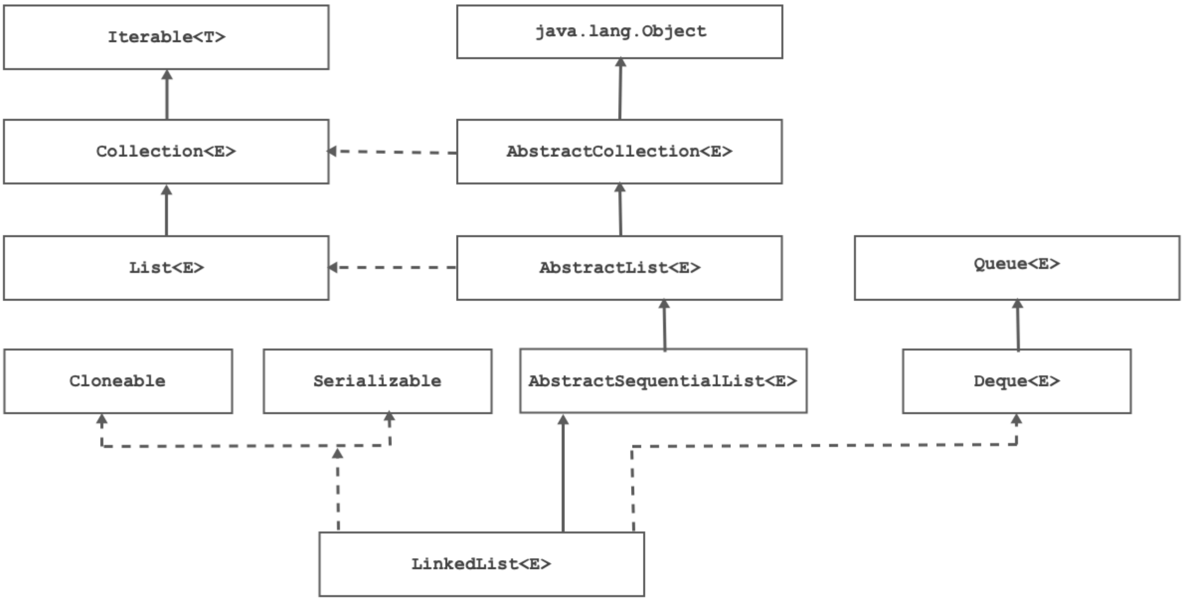
- It is sub class of java.util.Vector class.
- IN Java, Stack is synchronized collection.
- If we want to perform operations in Last In First Out(LIFO) order/manner then we should use Stack.
- Method Summary Stack:
  - public boolean empty()
  - public E push(E item)
  - public E peek()
  - public E peek()
  - public int search(Object o)

```
    public static void main(String[] args) {
        Stack<Integer> stk = new Stack<>();
        stk.push(10);
        stk.push(20);
        stk.push(30);
        stk.push(40);
        stk.push(50);

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

- If we want, unsynchronized implementation of Stack then we should use Deque implementation

```
public static void main(String[] args) {
    Deque<Integer> stk = new ArrayDeque<>();
    stk.push(10);
    stk.push(20);
    stk.push(30);
    stk.push(40);
    stk.push(50);

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

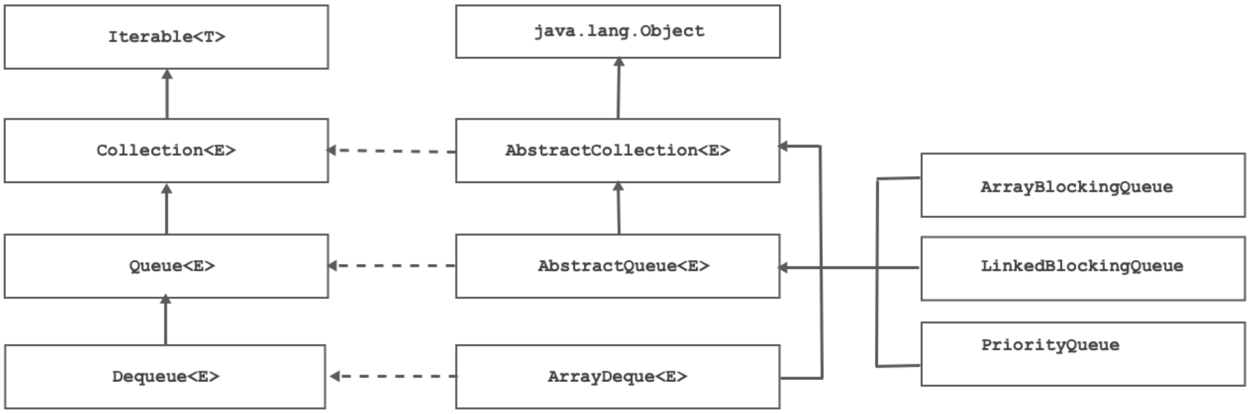
LinkedList



- It is a class declared in java.util package. Its implementation is based on Doubly LinkedList.
- LinkedList class implements List as well as Deque interface.
- Since it is List collection, It stored elements in sequential manner.
- Since it is List collection, It can contain duplicate elements as well as null elements
- Since it is List collection, We can access its elements using integer index.
- Since it is List collection, We can traverse its elements using Iterator and ListIterator
- LinkedList collection is unsynchronized. Using Collections.synchronizedList() method we can make it synchronized.

- This class is a member of the Java Collections Framework.It is introduced in JDK 1.2

Queue



- It is sub interface of Collection interface.
- If we want to perform operations in First In First Out order then we should use Queue implementation.
- This interface is a member of the Java Collections Framework.
- It is introduced in JDK 1.5
- Method Summary of Queue interface:
    - boolean add(E e)
    - boolean offer(E e)
    - E remove()
    - E poll()
    - E element()
    - E peek()
- Consider following code:

```
public static void main(String[] args) {
    Queue<Integer> que = new ArrayDeque<>();
    que.add(10);
    que.add(20);
    que.add(30);
    que.add(40);
    que.add(50);
    //que.add(null);    //Not Allowed

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

- Consider following code:

```java
public static void main(String[] args) {
    Queue<Integer> que = new ArrayDeque<>();
    que.offer(10);
    que.offer(20);
    que.offer(30);
    que.offer(40);
    que.offer(50);
    //que.offer(null);  //Not Allowed

    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.
- The name deque is short for "double ended queue" and is usually pronounced "deck".
- This interface is a member of the Java Collections Framework.
- It is introduced in JDK 1.6