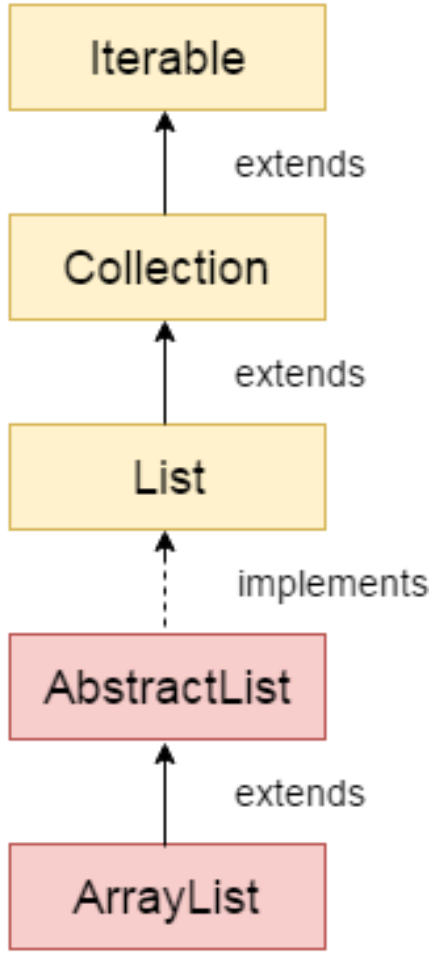
Jav Programming (TCS408)

Java Collections and Framework

ArrayList

- Java ArrayList class uses a dynamic array.
- for storing the elements. It is like an array, but there is no size limit.
- We can add or remove elements anytime. So, it is much more flexible than the traditional array.
- It is found in the java.util package.
- It is like the Vector in C++.
- It inherits the AbstractList class and implements List interface.



- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because the array works on an index basis.
- In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
- Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.
- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:

ArrayList<int> al = ArrayList<int>(); // does not work

ArrayList<Integer> al = new ArrayList<Integer>(); // works fine

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Method	Description
void add (int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add (E e)	It is used to append the specified element at the end of a list.
boolean addAll (Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll (int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear ()	It is used to remove all of the elements from this list.
void ensureCapacity(int requiredCapacity)	It is used to enhance the capacity of an ArrayList instance.
E get(int index)	It is used to fetch the element from the particular position of the list.
List<E> subList(int fromIndex, int toIndex)	It is used to fetch all the elements that lies within the given range.
int size()	It is used to return the number of elements present in the list.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

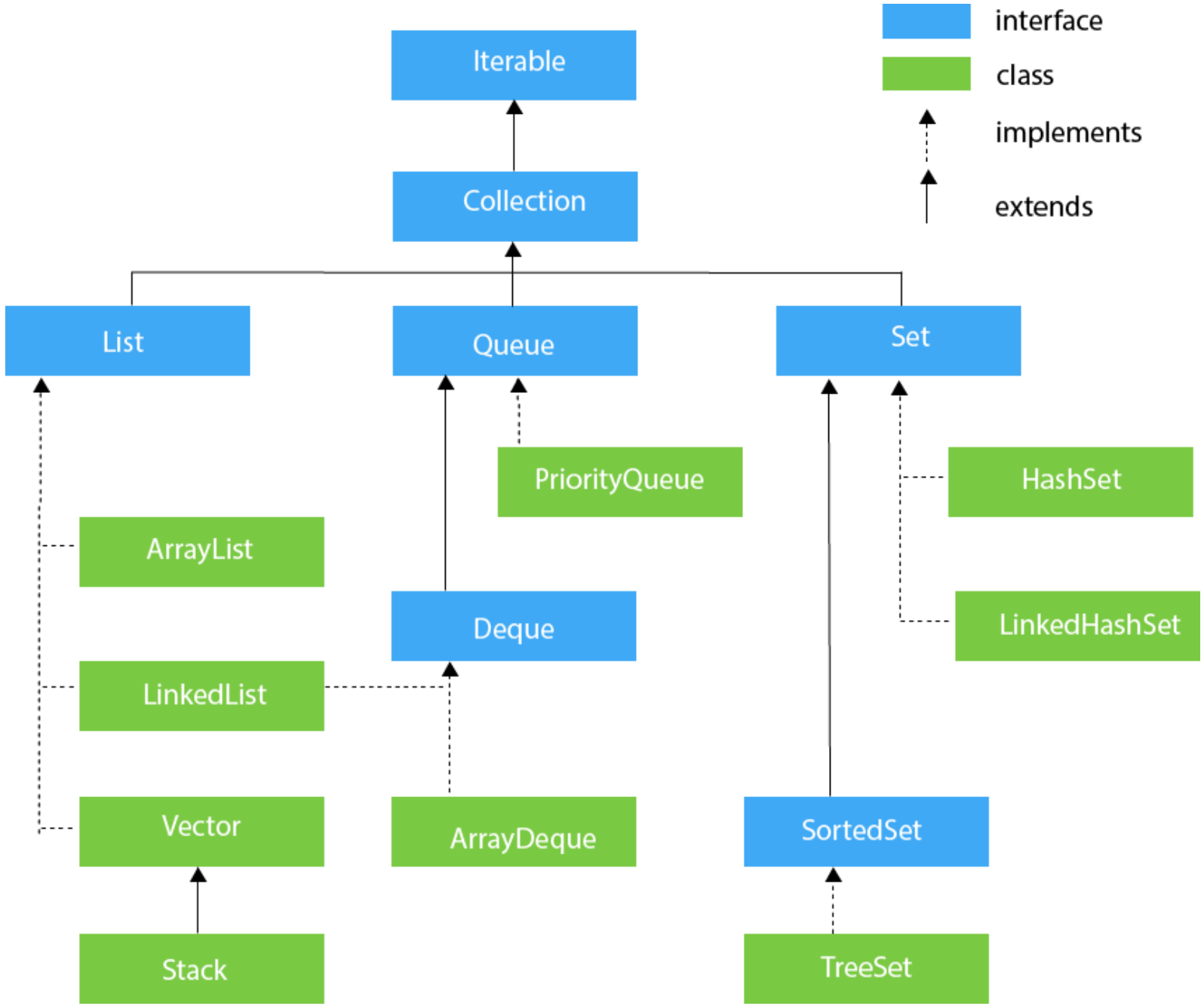
Collections in Java

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

- Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

- A Collection represents a single unit of objects, i.e., a group.


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

Java Non-generic Vs. Generic Collection

- Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.
- Java new generic collection allows you to have only one type of object in a collection. Now it is type-safe, so typecasting is not required at runtime.

```
ArrayList list=new ArrayList();//creating old non-generic arraylist
```

```
ArrayList<String> list=new ArrayList<String>();//creating new generic arr
aylist
```

```
import java.util.*;
 public class ArrayListExample1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();   //Creating arraylist
     list.add("Mango");   //Adding object in arraylist
       list.add("Banana");
     list.add("Grapes");
System.out.println(list);
 }
}
```

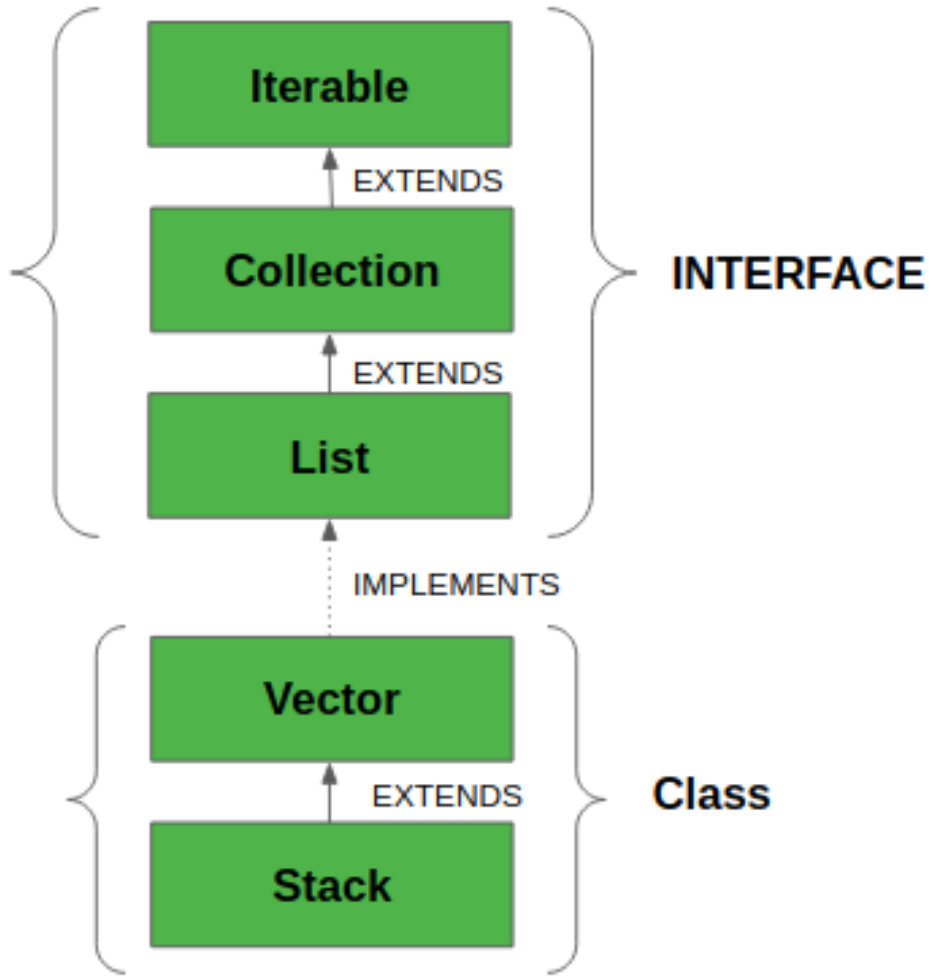
If we add:

Integer a=new Integer(10);

list.add(a); will give error

Vector

- Variable argument to the method concept can also be achieved using Vector.
- Array of objects.
- Can store any number of objects that may vary in size or types.
- Class in java.util package.
- Vector implements a dynamic array that means it can grow or shrink as required.
- Arrays can be easily implemented using Vectors.
- If increment is specified, Vector will expand according to it in each allocation cycle but if increment is not specified then vector's capacity get doubled in each allocation cycle.


Iterable
EXTENDS
Collection
EXTENDS
List
INTERFACE
IMPLEMENTS
Vector
EXTENDS
Stack
Class

Vector Methods

Method	Description
addELement(item)	Adds an item into vector
elementAt(n)	Returns an item from nth position of a vector
size()	Returns the size of vector
removeElement(item)	Removes specified item from the list
removeElementAt(n)	Removes an item from nth position
removeAllElements( )	Removes all elements of a vector
copyInto(array)	Used to copy all elements of a vector into the array
insertELementAt(item,n)	Insert an item into a vector at nth position.

Vector Methods

```
import java.util.*;
public class Main
{
        public static void main(String[ ] args) {
                Vector list=new Vector();
                int i;
                Integer s1=new Integer(123);
                String s2=new String("PQR");
                list.addElement(s1);
                 list.addElement(s2);
                 System.out.println(list);
        }

}
```

O/P: [123,PQR]

```
import java.io.*;
import java.util.*;
class Main{
        public static void main(String[] args)
        {               int n = 5;

// Declaring the Vector with initial size n
                Vector<Integer> v = new Vector<Integer>(n);
                for (int i = 1; i <= n; i++)
                        v.add(i);
                System.out.println(v);
// Remove element at index 3
                v.remove(3);
                System.out.println(v);
                for (int i = 0; i < v.size(); i++)
                        System.out.print(v.get(i) + " ");
        } }
```

ArrayList/Vector

ArrayList	Vector
1) ArrayList is not synchronized.	Vector is synchronized.
2) ArrayList increments 50% of current array size if the number of elements exceeds from its capacity.	Vector increments 100% means doubles the array size if the total number of elements exceeds than its capacity.
3) ArrayList is not a legacy class. It is introduced in JDK 1.2.	Vector is a legacy class.
4) ArrayList is fast because it is non-synchronized.	Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.

Vector Methods

```
import java.util.*;
public class Main
{
        public static void main(String[] args) {
                Vector list=new Vector();
                int i;
                for(i=0;i<args.length;i++)
                {
                   list.addElement(args[i]);
                }
                System.out.println(list);
                String s[]=new String[list.size()];
                list.copyInto(s);
                for(i=0;i<args.length;i++)
                {
                   System.out.println(s[i]);
                }
        }
}
```

O/P:

```
[2,3,4,5]
2
3
4
5
```