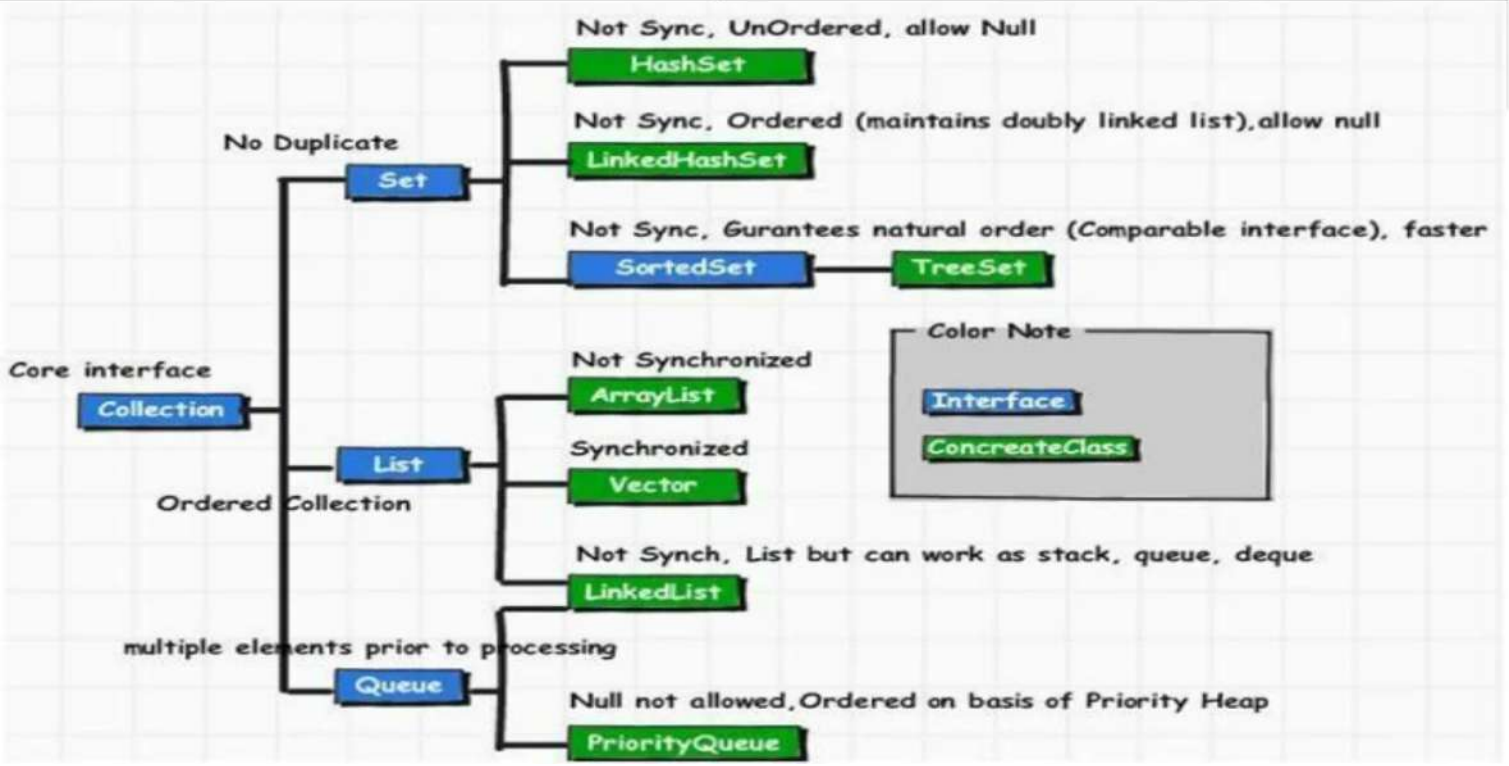
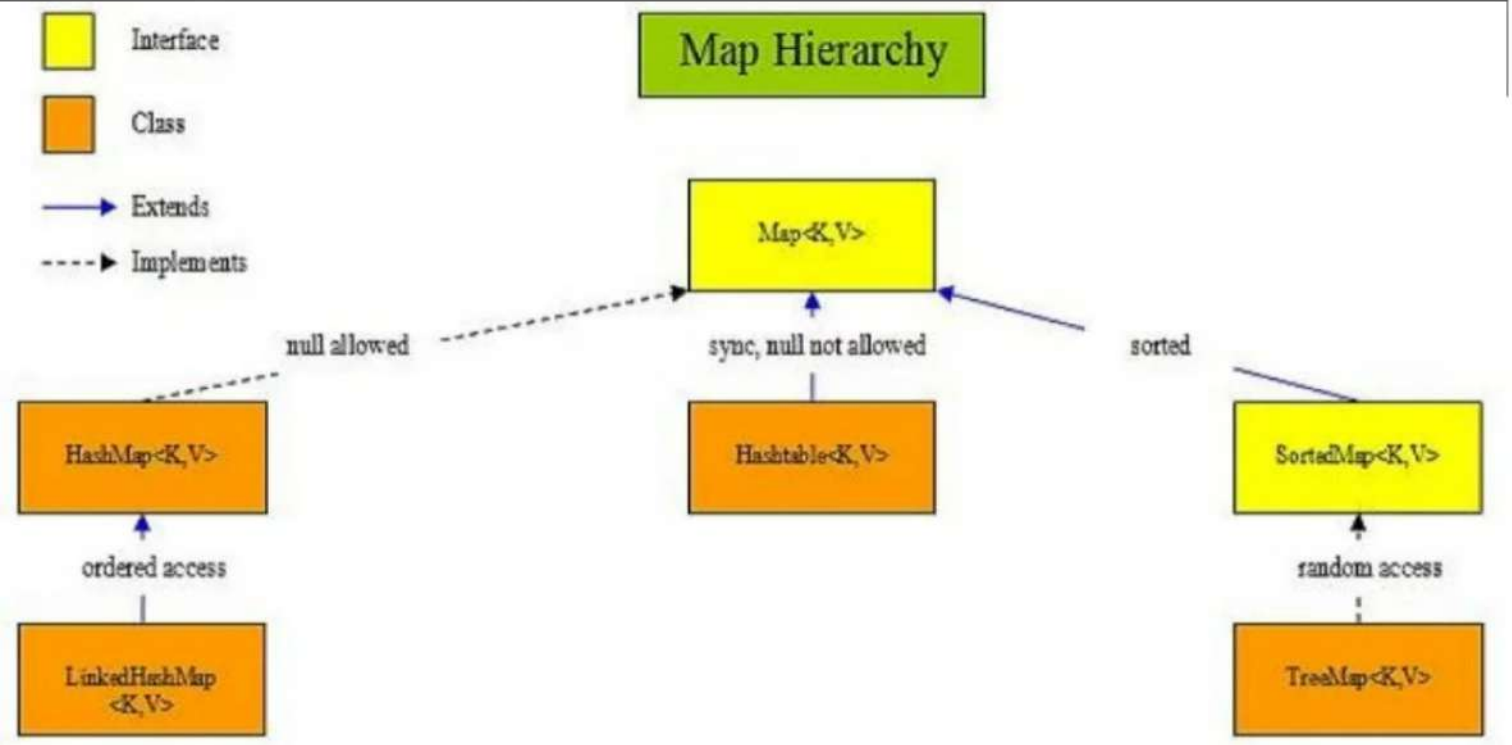
Collection Framework

- The Java language supports fixed-size arrays to store data.
- Developer typically require a data structure which is flexible in size, so that they can add and remove items from this data structure on request. To avoid that every developer has to implement his custom data structure the Java library provide several default implementations for this via **the collection framework.**
- The java.util package contains one of Java's most powerful subsystems: collections.
- Collections were added by the initial release of Java 2, and enhanced by Java 2 - version 1.4.
- Java collections are dynamic in size, e.g. a collection can **contain a flexible number of objects.**

Collection Hierarchy



Map Hierarchy



Iterator

Iterators provide a means of traversing a set of data. It can be used with arrays and various classes in the Collection Framework. The Iterator interface supports the following methods:

- next: This method returns the next element
- hasNext: This method returns true if there are additional elements
- remove: This method removes the element from the list

The ListIterator interface, when available, is an alternative to the Iterator interface. It uses the same methods and provides additional capabilities including:

- Traversal of the list in either direction
- Modification of its elements
- Access to the element's position

ListIterator

The methods of the ListIterator interface include the following:

- next: This method returns the next element
- previous: This method returns the previous element
- hasNext: This method returns true if there are additional elements that follow the current one
- hasPrevious: This method returns true if there are additional elements that precede the current one
- nextIndex: This method returns the index of the next element to be returned by the next method
- previousIndex: This method returns the index of the previous element to be returned by the previous method
- add: This method inserts an element into the list (optional)
- remove: This method removes the element from the list (optional)
- set: This method replaces an element in the list (optional)

The List Interface

- The List Interface extends **Collection** and declares behavior of a collection that stores a sequence of elements.
- Developer typically require a data structure which is flexible in size, so that they can add and remove items from this data structure on request. To avoid that every developer has to implement his custom data structure the Java library provide several default implementations for this via **the collection framework.**
- The java.util package contains one of Java's most powerful subsystems: collections.
- Collections were added by the initial release of Java 2, enhanced by Java2-version1.4
- Java collections are dynamic in size, e.g. a collection can **contain a flexible number of objects.**

The ArrayList Class

What is unique feature of ArrayList?

- **ArrayList in Java** is most frequently used collection class after HashMap in Java.
- Java ArrayList represents an automatic re-sizeable array and used in place of array. Since we can not modify size of an array after creating it, we prefer to use ArrayList in Java which resize itself automatically once it gets full.
- It implements List interface and allow null not thread safe.
- Java ArrayList also maintains insertion order of elements and allows duplicates.
- ArrayList supports both Iterator and ListIterator for iteration but it's recommended to use ListIterator as it allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the Iterator's current position in the list.

ArrayList Example

```
import java.util.*;

class TestCollection1{
 public static void main(String args[]){

      ArrayList<String> al=new ArrayList<String>();//creting arraylist
      al.add("Jolly");//adding object in arraylist
      al.add("Sanjana");
      al.add("Niva");
      al.add("Ritu");

      Iterator itr=al.iterator();//getting Iterator from arraylist to traverse elements
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

Store Object in ArrayList

```java
import java.util.*;
public class TestCollection3{
    public static void main(String args[]){
        //Creating user-defined class objects
        Student s1 = new Student(101,"Ramesh",20);
        Student s2 = new Student(102,"Mahesh",21);
        Student s2 = new Student(103,"Suresh",22);

        ArrayList<Student> al=new ArrayList<Student>();
        al.add(s1);//adding Student class object
        al.add(s2);
        al.add(s3);

        Iterator itr=al.iterator();
        //traversing elements of ArrayList object
        while(itr.hasNext()){
            Student st=(Student)itr.next();
            System.out.println(st.rollno+" "+st.name+" "+st.age);
        }
    }
}
```

```java
class Student{
    int rollno;
    String name;
    int age;
    Student(int rollno,String name,int age){
        this.rollno=rollno;
        this.name=name;
        this.age=age;
    }
}
```

The Vector Class

What is unique feature of Vector?

- **Vector** is a Concrete class - ordered collection (add/remove elements at the end) and implements dynamic resizable array. It is similar to **ArrayList** but with few differences.
- **Vectors are synchronized:** Any method that touches the Vector's contents is thread safe. This means if one thread is working on Vector, no other thread can get a hold of it. Unlike ArrayList, only one thread can perform an operation on vector at a time.

When to use ArrayList and when to use Vector?

- It totally depends on the requirement. If there is a need to perform “thread-safe” operation the vector is your best bet as it ensures that only one thread access the collection at a time.

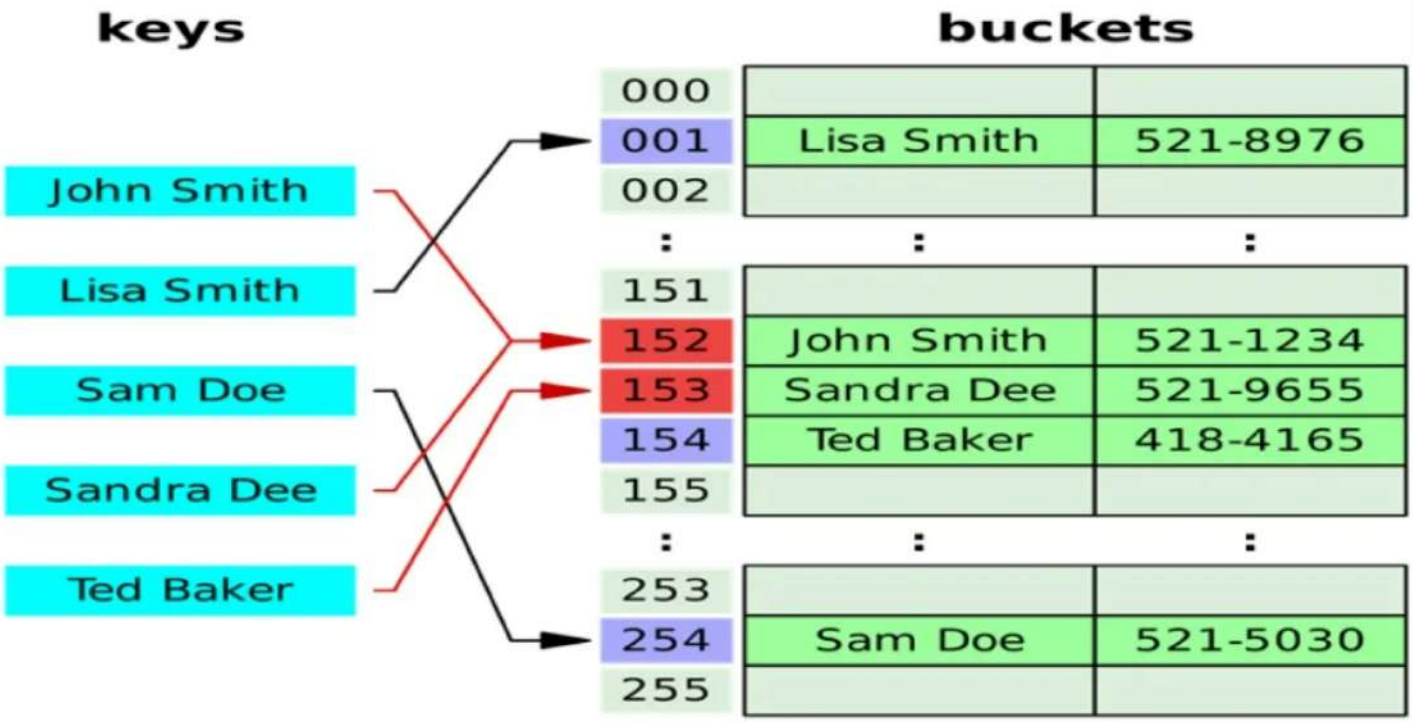
- **Performance:** Synchronized operations consumes more time compared to non-synchronized ones so if there is no need for thread safe operation, ArrayList is a better choice as performance will be improved because of the concurrent processes.

The Set Interface

What is unique feature of Set?

- **Set** is a **collection** interface - Collection that cannot contain duplicate elements. (Ex. Set of playing cards)
- **Set Implementations**
  - **java.util.HashSet** - Stores unique elements in Hash Table
  - **java.util.TreeSet** - Stores unique elements in Hash Table in sorted order (Ascending)
  - java.util.LinkedHashSet - Stores unique elements in Hash Table and maintain insertion order
  - Note: Refer Set_Map.txt document (in shared folder of Google Drive) for Examples of HashSet and TreeSet

MAP – Key value pair



The Map Interface

What is unique feature of Map?

- The java.util.Map interface represents a mapping between a key and a value
- The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit different from the rest of the collection types.

❑ Map Implementations

- **java.util.HashMap – Stores key valye pair**
- java.util.Hashtable - Similar to HashMap, but sunchronized
- java.util.EnumMap - use enum values as keys for lookup
- java.util.IdentityHashMap - uses reference equality when comparing elements
- java.util.LinkedHashMap - maintains a linked list of the entries and in the order in which they were inserted
- **java.util.TreeMap - provides an efficient means of storing key/value pairs in sorted order**
- java.util.WeakHashMap - stores only weak references to its keys

❑ Most commonly used Map implementations are HashMap and TreeMap

❑ Note: Refer Set_Map.txt document (in shared folder of Google Drive) for Examples of HashSet and TreeSet