Collection Framework

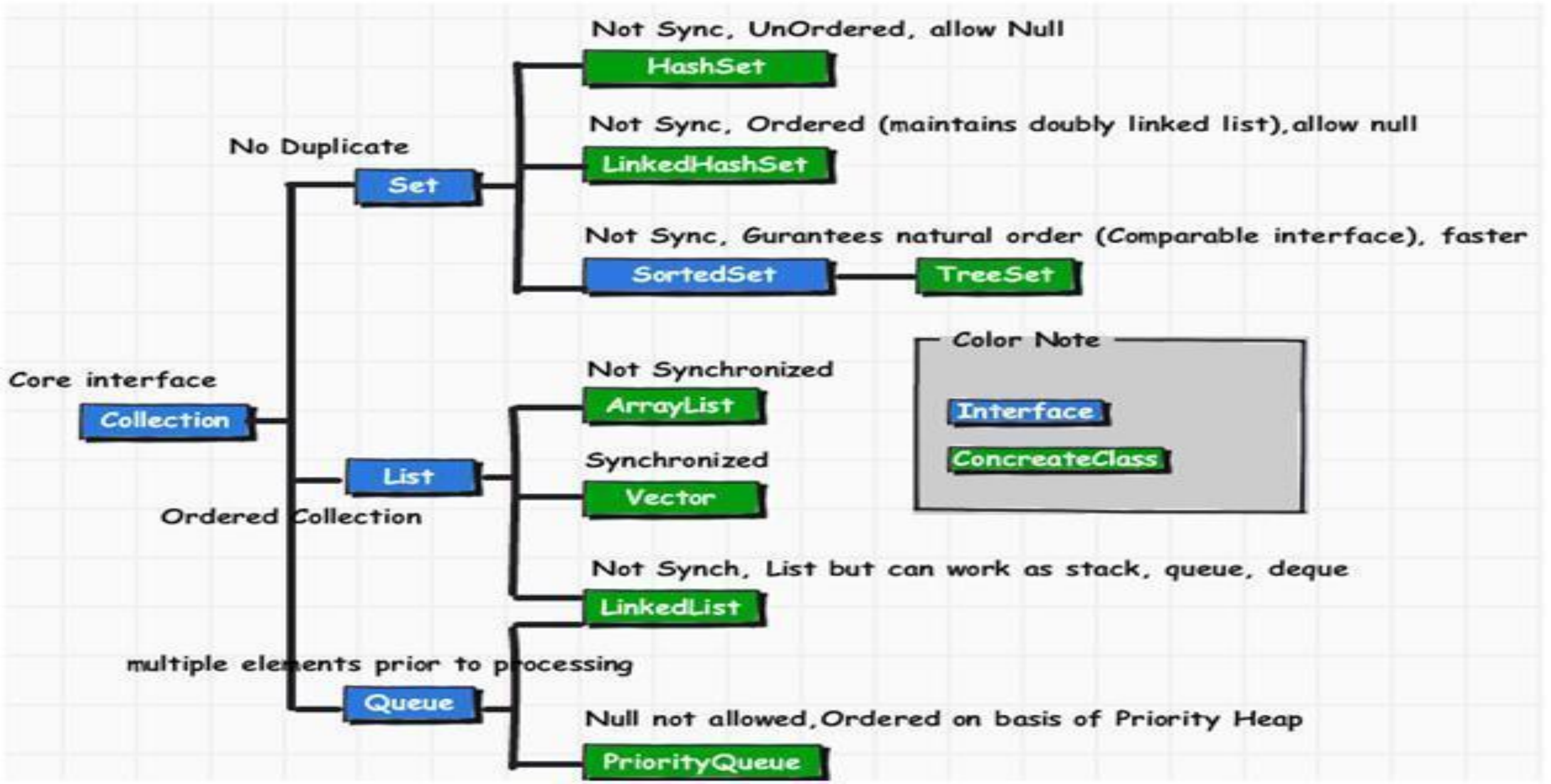
List Set Map

Prepared By: Minal Maniar

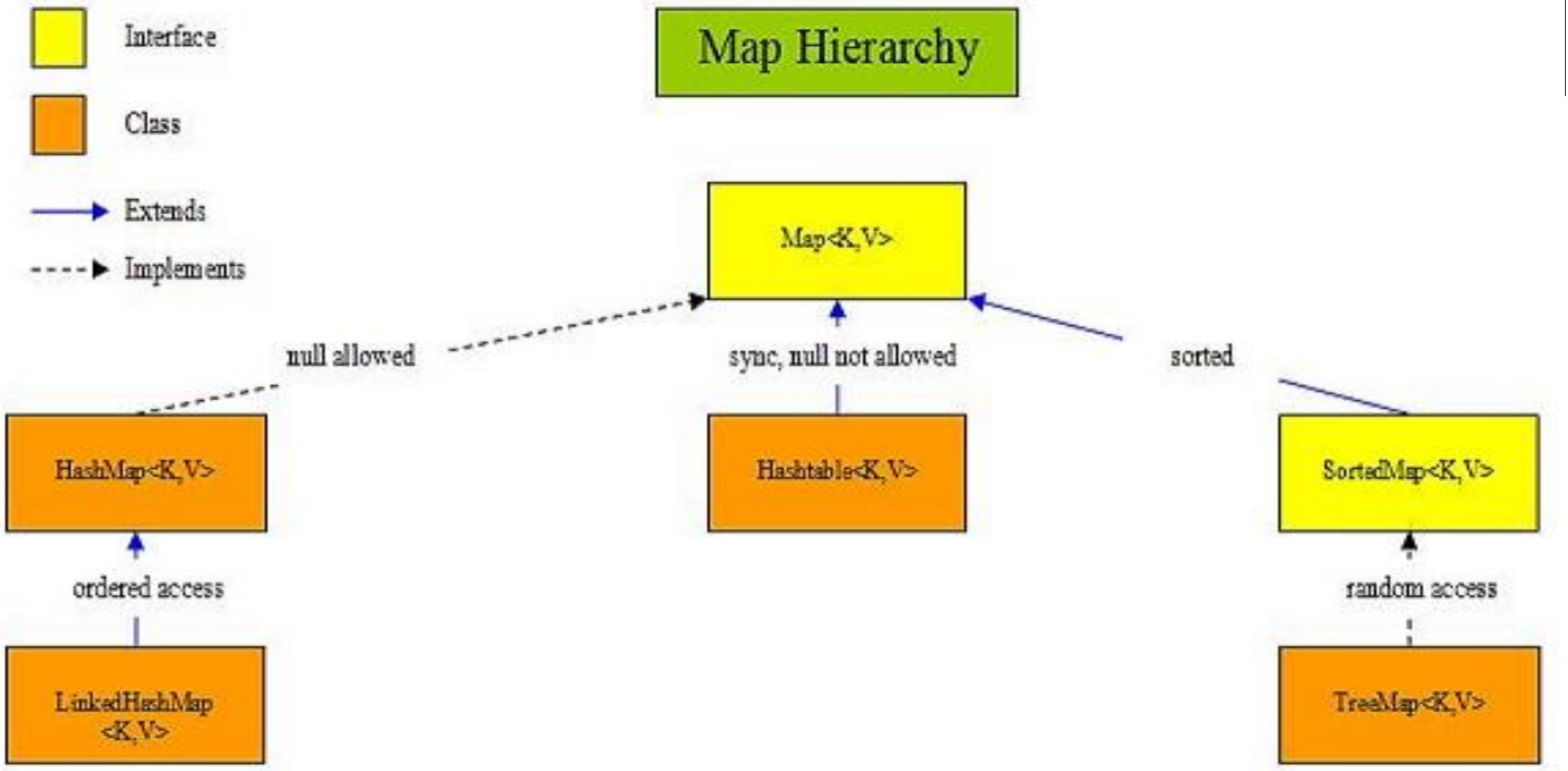
Collection Framework

- 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


Not Sync, UnOrdered, allow Null
HashSet
Not Sync, Ordered (maintains doubly linked list),allow null
No Duplicate
LinkedHashSet
Set
Not Sync, Gurantees natural order (Comparable interface), faster
SortedSet
TreeSet
Color Note
Not Synchronized
Core interface
ArrayList
Interface
Collection
Synchronized
ConcreateClass
List
Vector
Ordered Collection
Not Synch, List but can work as stack, queue, deque
LinkedList
multiple elements prior to processing
Queue
Null not allowed,Ordered on basis of Priority Heap
PriorityQueue

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.

Creating ArrayList

The ArrayList class possesses the following three constructors:

- A default constructor
- One that accepts a Collection object
- One that accepts an initial capacity

The capacity of a ArrayList object refers to how many elements the list can hold.

When more elements need to be added and the list is full, the size of the list will be automatically increased. The initial capacity of a ArrayList created with its default constructor is 10.

The following example creates 2 lists, one with a capacity of 10 and the second with a capacity of 20:

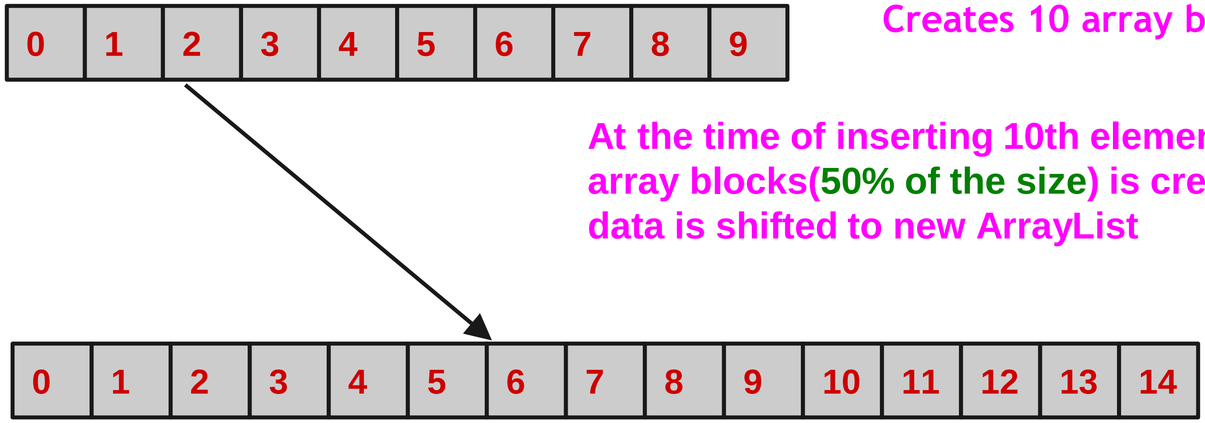
```
ArrayList list1 = new ArrayList();          ArrayList list2 = new ArrayList(20);
```

The ArrayList class supports generics. Here, a list of strings is created:

```
ArrayList<String> list3 = new ArrayList<String>();
```

Storage in ArrayList

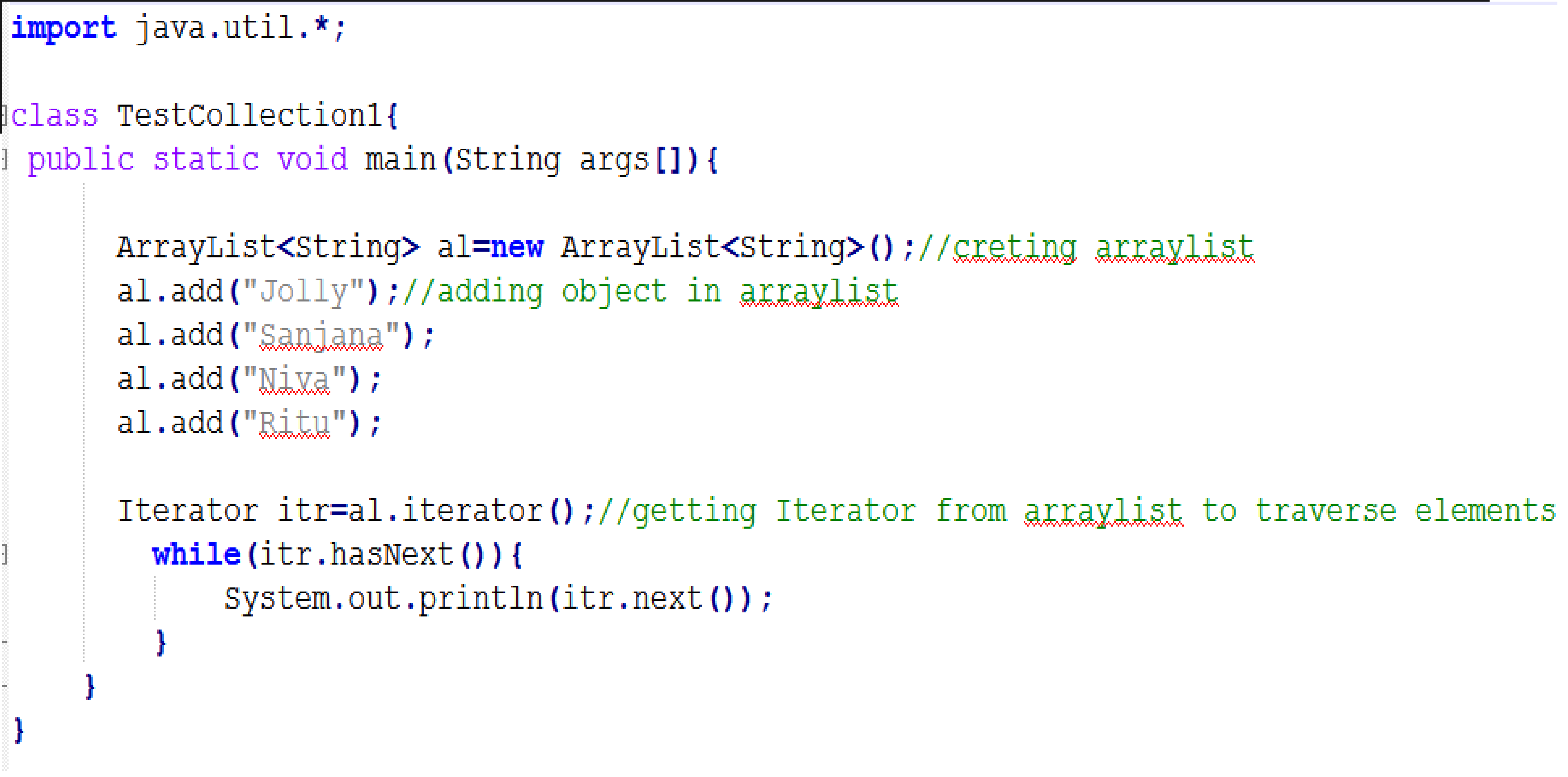
- ❑ Internally, both the ArrayList hold onto their contents using an Array. An ArrayList defaults to 50% increase in the size of its array when it inserts last element of initial capacity,
- ❑ Ex: ArrayList al = new ArrayList();



Creates 10 array blocks initially

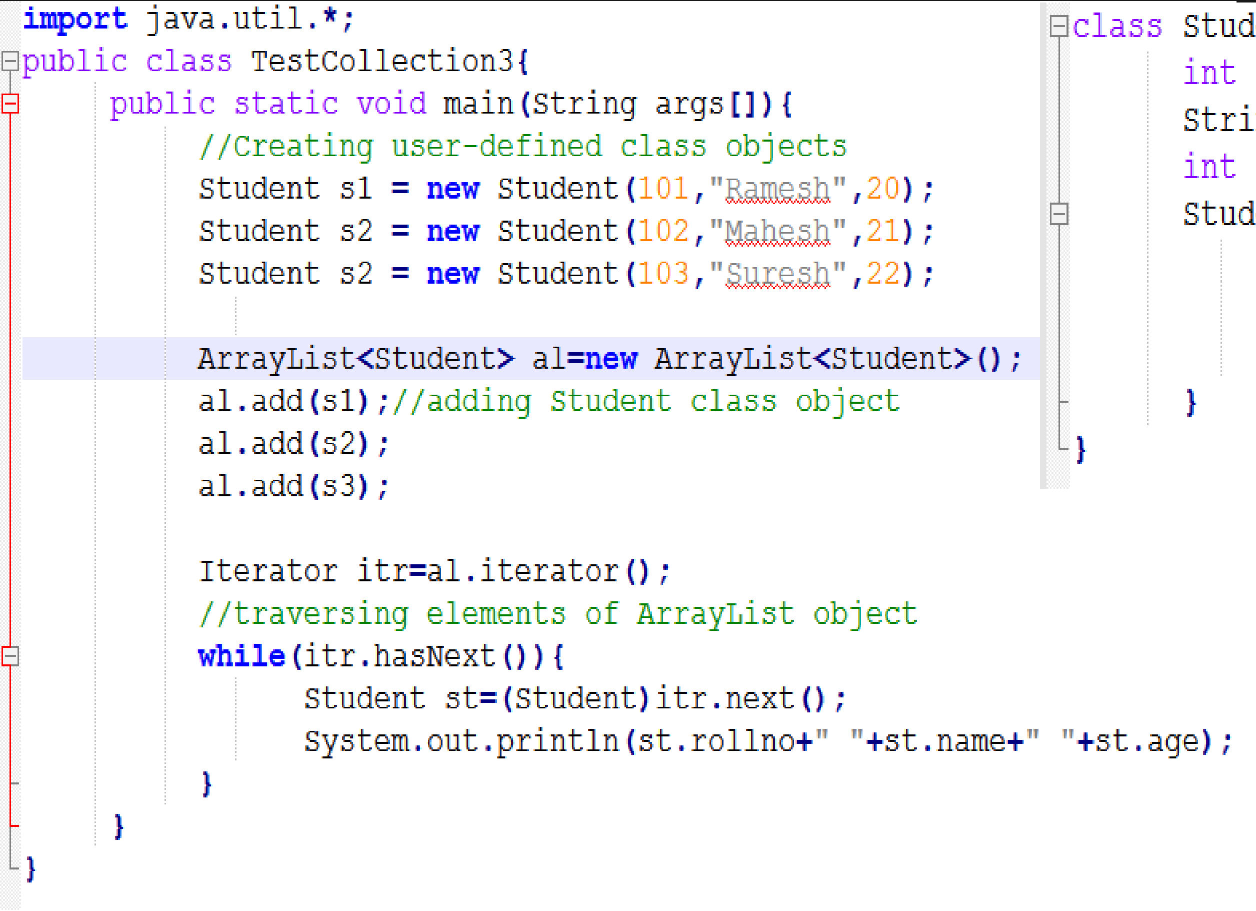
At the time of inserting 10th element, 5 array blocks(50% of the size) is created, old data is shifted to new ArrayList

ArrayList Example

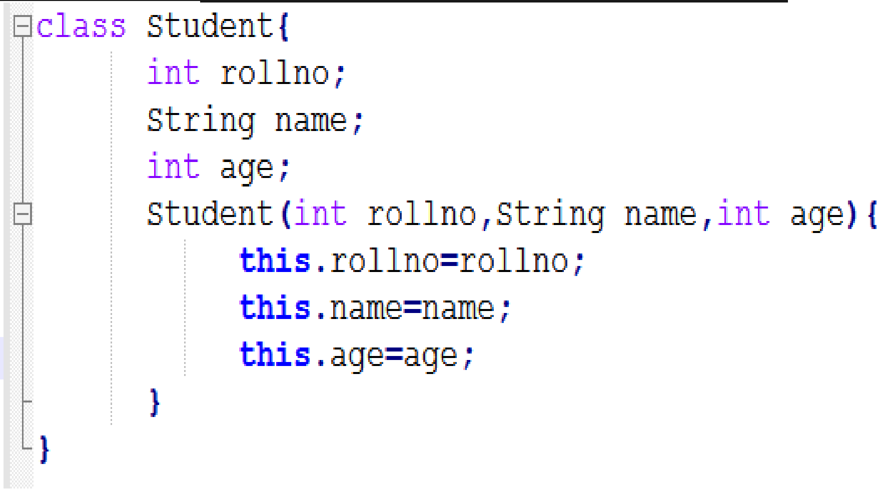


```
import java.util.*;

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



```
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);
        }
    }
}
```



```
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

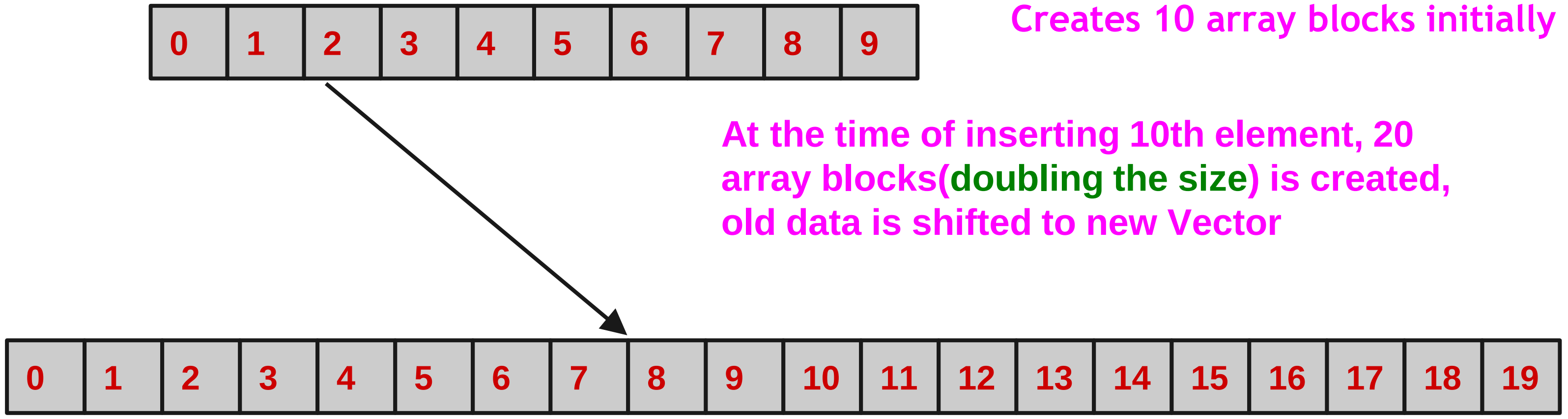
What is unique feature of Vector?

- ❑ **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.

Storage in Vector

How do Vector stores Elements?

- Internally, both the ArrayList and Vector hold onto their contents using an Array. A Vector defaults to doubling the size of its array when it inserts last element of initial capacity,
- Ex: Vector v = new Vector();



Creates 10 array blocks initially

At the time of inserting 10th element, 20 array blocks(doubling the size) is created, old data is shifted to new Vector

Performance and Usage Pattern of Vector

- Vector are good for retrieving elements from a specific position in the container or for adding and removing elements from the end of the container. All of these operations can be performed in constant time - However, adding and removing elements from any other position proves more expensive - Linear. These operations are more expensive because you have to shift all elements at index *i* and higher over by one element. So, if you want to index elements or add and remove elements at the end of the array, use either a Vector or an ArrayList.
- It's always best to set the object's initial capacity to the largest capacity that your program will need.
- By carefully setting the capacity, you can avoid paying the penalty needed to resize the internal array later. If you don't know how much data you'll have, but you do know the rate at which it grows, Vector does possess a slight advantage since you can set the increment value.

Declare a Vector

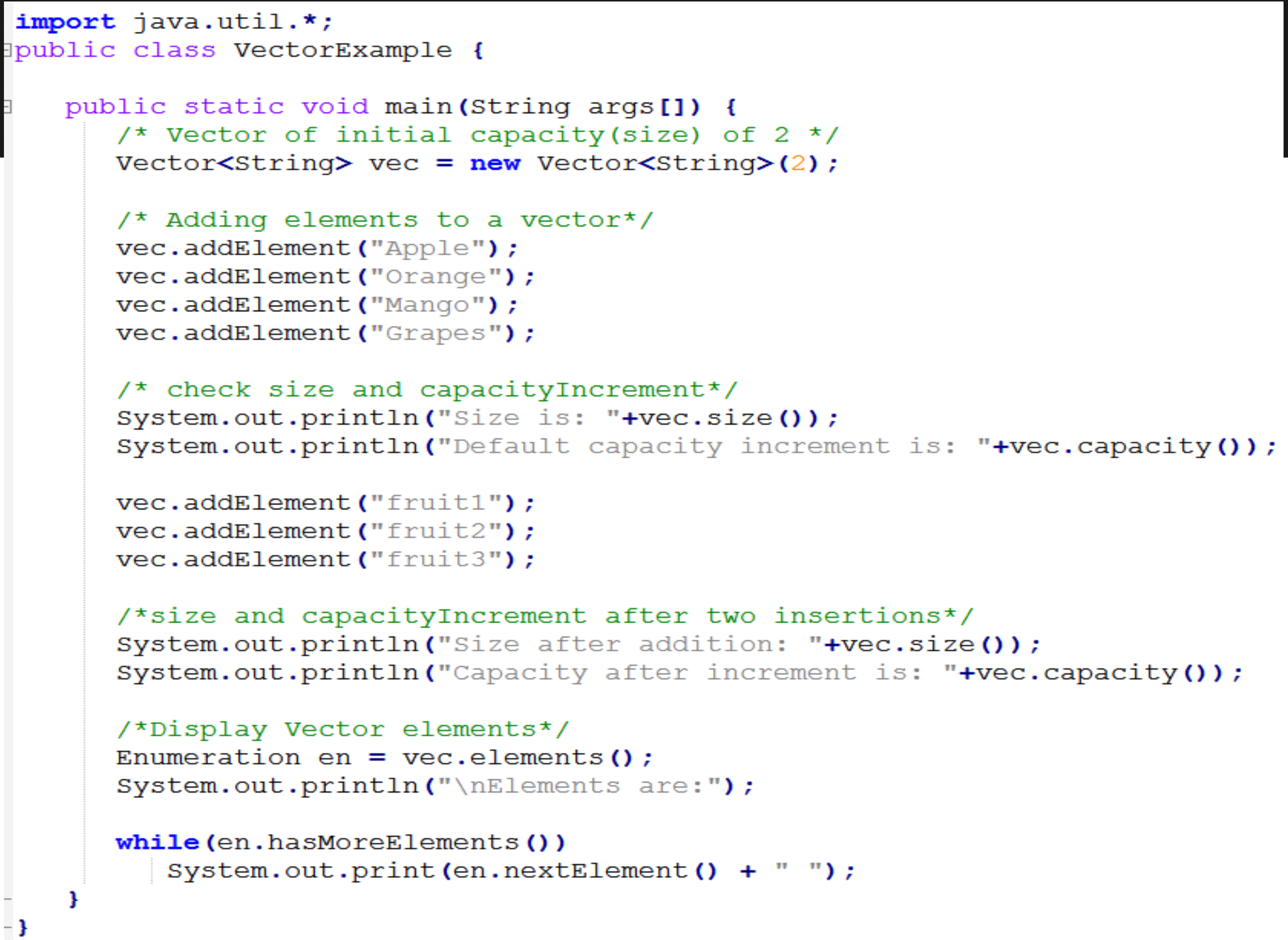
3 ways to declare a Vector

1. `Vector vec = new Vector(); // Creates Vector default size is 10.`
2. `Vector object= new Vector(int initialCapacity) // Creates Vector with given initial capacity` `Ex: Vector vec = new Vector(3);`
3. `Vector object= new vector(int initialcapacity, capacityIncrement)`

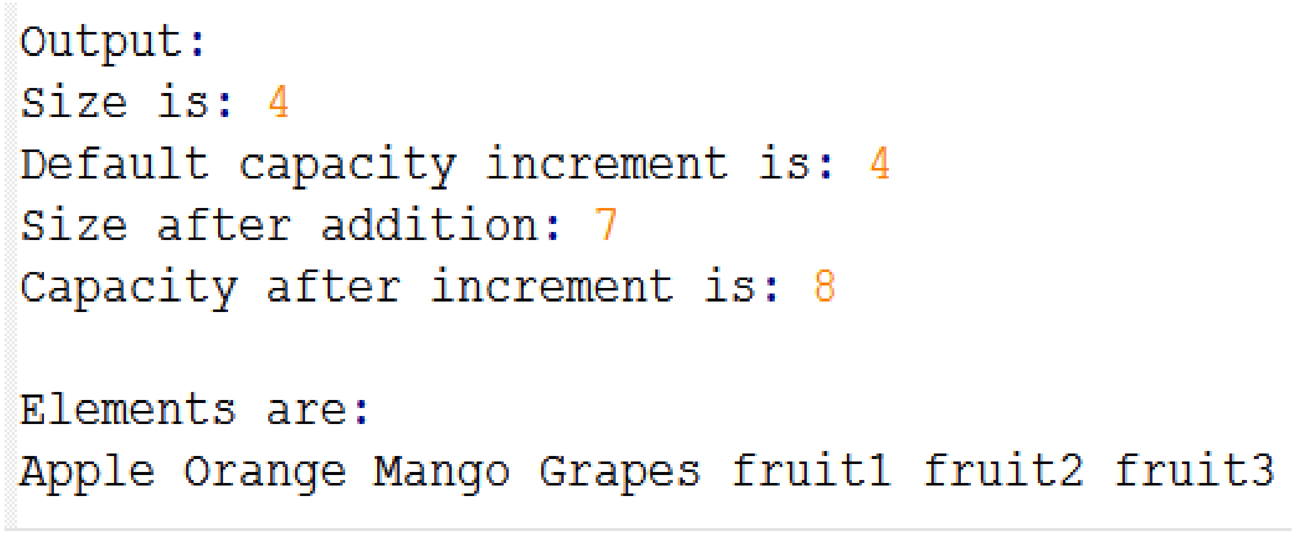
   `Ex: Vector vec= new Vector(4, 6)`

   Creates Vector with initial capacity of 4 and when inserts 4th element, increments its size by 6. This will result into new Vector of size 10.

Vector Example


```java
import java.util.*;
public class VectorExample {

   public static void main(String args[]) {
      /* Vector of initial capacity(size) of 2 */
      Vector<String> vec = new Vector<String>(2);

      /* Adding elements to a vector*/
      vec.addElement("Apple");
      vec.addElement("Orange");
      vec.addElement("Mango");
      vec.addElement("Grapes");

      /* check size and capacityIncrement*/
      System.out.println("Size is: "+vec.size());
      System.out.println("Default capacity increment is: "+vec.capacity());

      vec.addElement("fruit1");
      vec.addElement("fruit2");
      vec.addElement("fruit3");

      /*size and capacityIncrement after two insertions*/
      System.out.println("Size after addition: "+vec.size());
      System.out.println("Capacity after increment is: "+vec.capacity());

      /*Display Vector elements*/
      Enumeration en = vec.elements();
      System.out.println("\nElements are:");

      while(en.hasMoreElements())
         System.out.print(en.nextElement() + " ");
   }
}
```

Vector Example : Output



```
Output:
Size is: 4
Default capacity increment is: 4
Size after addition: 7
Capacity after increment is: 8

Elements are:
Apple Orange Mango Grapes fruit1 fruit2 fruit3
```

ArrayList vs Vector

Differences

1) **Synchronization and Thread-safe:** **Vector is synchronized while ArrayList is not synchronized.** ArrayList is non-synchronized which means multiple threads can work on ArrayList at the same time. For e.g. if one thread is performing an add operation on ArrayList, there can be an another thread performing remove operation on ArrayList at the same time in a multithreaded environment

while Vector is synchronized. 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.

ArrayList vs Vector(2)

2) **Resize**: **Both ArrayList and Vector can grow and shrink dynamically to maintain the optimal use of storage, however the way they resized is different. ArrayList grow by half of its size (50% increase) when resized while Vector doubles the size of itself by default when grows.**

By default ArrayList size is 10. It checks whether it reaches to last element, then it will create a new array, copy old array data to new one and old array will be eligible for garbage collection by JVM.

3) **Performance: Vector is slow as it is thread safe compared to ArrayList.** ArrayList gives better performance as it is non-synchronized. Vector operations gives poor performance as they are thread-safe, the thread which works on Vector gets a lock on it which makes other thread wait till the lock is released.

ArrayList vs Vector(3)

4) Set Increment Size: **ArrayList does not define the increment size . Vector defines the increment size .**

You can find the following method in Vector Class.
public synchronized void setSize(int i) { //some code }. There is no setSize() method or any other method in ArrayList which can manually set the increment size.

Similarities

1. Both Vector and ArrayList use growable array data structure.
2. The iterator and listIterator returned by these classes (Vector and ArrayList) are fail-fast.
3. They both are ordered collection classes as they maintain the elements insertion order.
4. Vector & ArrayList both allows duplicate and null values.
5. They both grows and shrinks automatically when overflow and deletion happens.

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.

Test performance by calculating Time

```
ArrayList arrayList = new ArrayList();

// ArrayList add
long startTime = System.nanoTime();

for (int i = 0; i < 100000; i++) {
  arrayList.add(i);
}
long endTime = System.nanoTime();
long duration = endTime - startTime;
System.out.println("ArrayList add:  " + duration);
```

How to make ArrayList synchronized?

```
//Use Collecions.synzhonizedList method

List list = Collections.synchronizedList(new ArrayList());

...

//If you want to use iterator on the synchronized list, It should be in synchronized block.

synchronized (list) {

        Iterator iterator = list.iterator();

        while (iterator.hasNext())

                ...

        iterator.next();

                ...

}
```

Exercise

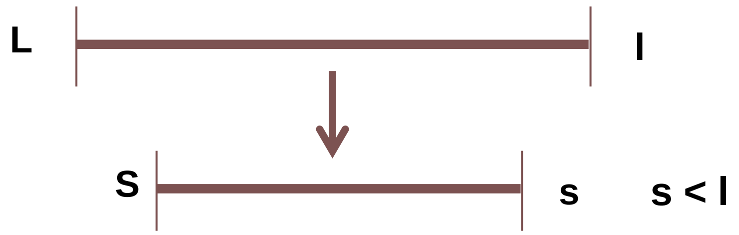
1. Write a program that reads a text file, specified by the first command line argument, into a List. The program should then print random lines from the file, the number of lines printed to be specified by the second command line argument. Write the program so that a correctly-sized collection is allocated all at once, instead of being gradually expanded as the file is read in. Hint: To determine the number of lines in the file, use java.io.File.length to obtain the size of the file, then divide by an assumed size of an average line.
   (Refer document for the solution) Note : Refer document for 2 more exercise questions.
2. Write a program that adds 5 String elements in Vector print them in forward and backward order. **[Hint: for backward order use previous() method of ListIterator.]**

Assignment Questions

1. Array vs ArrayList
2. ArrayList vs Vector
3. How to copy or clone an ArrayList?
4. How to copy array to ArrayList?
5. How to shuffle elemenets in ArrayList?
6. How to read all elemenets in Vector by using iterator?
7. How to copy or clone a Vector?
8. How to add all elemenets of a list to Vector?
9. How to delete all elemenets from Vector?
10. How to find does Vector contains all list elemenets or not?
11. How to copy Vector to Array?
12. How to get sublist from Vector?

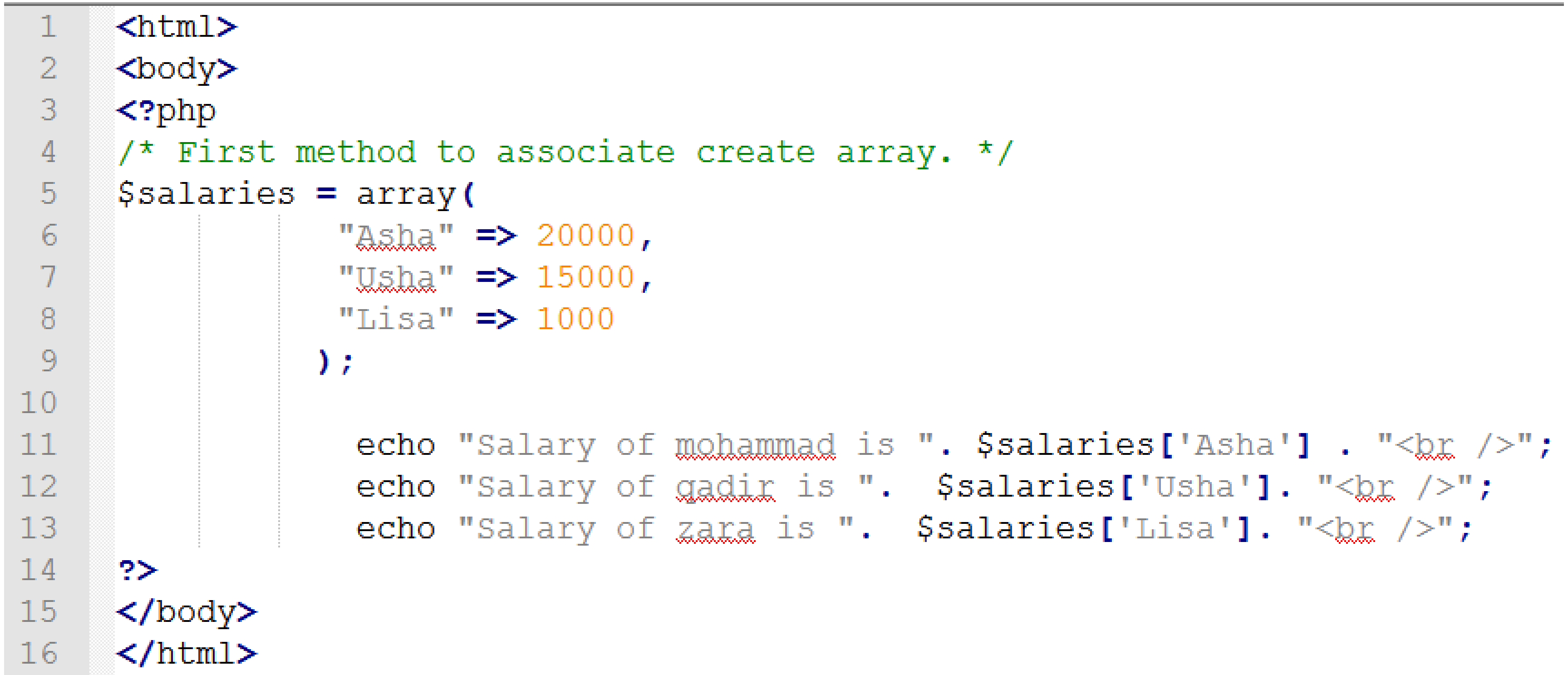
Hashing

- Hashing is a process of taking a big volume into small volume of data and to have strong association between keys and values



- Short list is created from a long list in such way that it work as a long list was.
- Here short list will only contain sufficient information- not all info

Hashing and Associative Array are similar



```
<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array(
           "Asha" => 20000,
           "Usha" => 15000,
           "Lisa" => 1000
          );

            echo "Salary of mohammad is ". $salaries['Asha'] . "<br />";
            echo "Salary of qadir is ".  $salaries['Usha']. "<br />";
            echo "Salary of zara is ".  $salaries['Lisa']. "<br />";
?>
</body>
</html>
```

The Set Interface

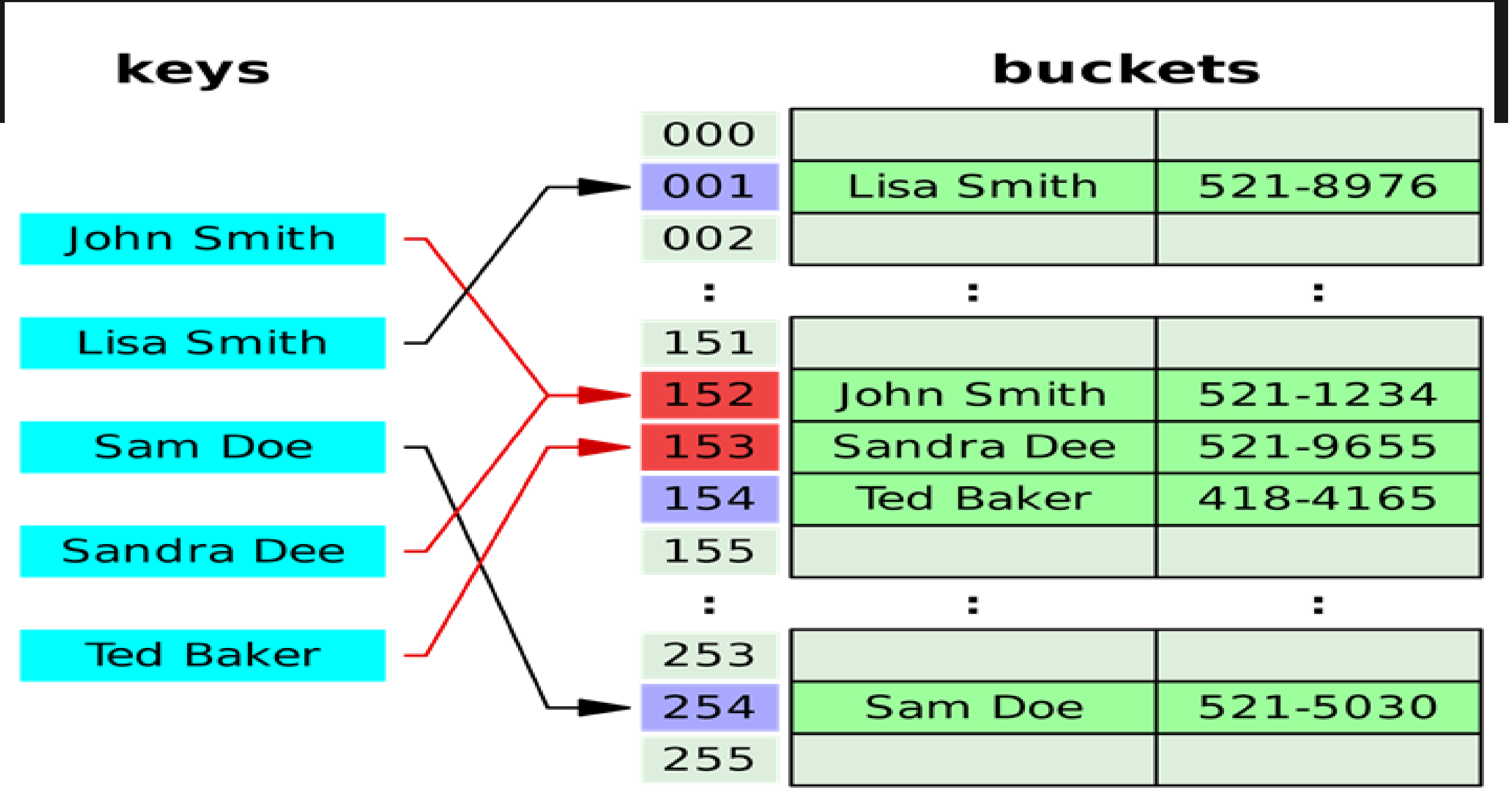
What is unique feature of Set?

❑ **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

References

- http://docs.oracle.com/
- http://www.tutorialspoint.com/
- http://javarevisited.blogspot.in/
- Oracle Certified Associate, Java SE 7 Programmer Study Guide by Richard M. Reese
- Java The Complete Reference by Herbert Schildt