PER SCHOLAS

Introduction to Java Programming

Unit 3 - Java complex collection - ArrayList, list, LinkedList, maps and HashMaps & Treeset

Java Complex Collections

- Students should be able to understand:
  - Arraylist class
  - list, maps and Set
  - HashMaps
  - Treeset

Java Complex Collections

- What is a collection in Java?
  - The Java Collection framework (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures. Although referred to as a framework, it works in a manner of a library
- The JCF aims to:
  - Implement fundamental collections (dynamic arrays, sets, maps, and hashtables) that are highly efficient
  - Allow different types of collections to work in a similar manner and with a high degree of interoperability
  - Extend and/or adapt a collection easily

Java Complex Collections
Arraylist class

- Arraylist class:
  - Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk
- Arraylist constructors:
  - Arraylist(): Builds an empty array list
  - ArrayList(Collection c): this constructor builds an Arraylist that is initialized with the elements of the collection c
  - ArrayList(int capacity): This constructor builds an array list that has the specified initial capacity. The capacity is the size of the underlying array that is used to store the elements. The capacity grows automatically as elements are added to an Arraylist

Java Complex Collections

Arraylist class Example

```
5  public class MainCollections
6  {
7      public static void main(String[] args)
8      {
9          // declaring ArrayList with initial size
10         ArrayList<Integer> arrli = new ArrayList<Integer>();
11         // initial size
12         System.out.println(arrli.size());
13         // Appending the new element at the end of the list
14         for (int i = 1; i <= 10; i++)
15             arrli.add(i);
16         // Printing elements
17         System.out.println(arrli);
18         // Remove element at index 3
19         arrli.remove(3);
20         // Displaying ArrayList after deletion
21         System.out.println(arrli);
22         // Printing elements one by one
23         for (int i = 0; i < arrli.size(); i++)
24             System.out.print(arrli.get(i) + " ");
25     }
26 }
```

Java Complex Collections

Arraylist class Debug - try to fix this program

```
5  public class MainCollections2
6  {
7      public static void main(String[] args)
8      {
9          ArrayList<Integer> arrli = new ArrayList<Integer>();
10         for (float i = 1; i <= 10; i++)
11         {
12             arrli.add(i);
13         }
14         int currentSize = arrli.size();
15         for (int j = currentSize; j <= arrli.size(); j++)
16         {
17             arrli.add(j);
18         }
19         System.out.println(arrli);
20     }
21 }
```

Java Complex Collections
List

- List Interface
  - List is an ordered collection of objects in which duplicate values can be stored. Since List preserves the insertion order it allows positional access and insertion of elements. List Interface is implemented by ArrayList, LinkedList, Vector and Stack classes.
  - Java.util.List is a child interface of Collection.
  - Creating List Objects:
    - List is an interface, we can create instance of List in following ways:

```
 9 public class CollectionsListImplements
10 {
11     public static void main(String[] args)
12     {
13         List<Integer> a = new ArrayList<Integer>();
14         List<Integer> b = new LinkedList<Integer>();
15         List<Integer> c = new Vector<Integer>();
16         List<Integer> d = new Stack<Integer>();
17     }
18 }
```

Java Complex Collections
List Example

```
 9 public class CollectionsListImplements
10 {
11     public static void main(String[] args)
12     {
13         List<Integer> a = new ArrayList<Integer>();
14         for(int i = 0; i < 10; i++) {
15             a.add(i);
16             System.out.println(a);
17         }
18         a.remove(2);
19         a.add(2, 102);
20         for(int x : a) {
21             System.out.print(x + " ");
22         }
23     }
24 }
```

Java Complex Collections
List Example

- Different ways to implement a List:
  - **List<Integer> b = Arrays.asList(3, 5, 7, 9, 10, 13, 15, 18, 23);**
  - **List<Integer> c = new ArrayList<Integer>();**
- Try it:
  - Given two sorted arrays:
  - List<Integer> a = Arrays.asList(3, 4, 6, 7, 9, 12, 15, 17, 23);
  - List<Integer> b = Arrays.asList(3, 5, 7, 9, 10, 13, 15, 18, 23);
  - Add the elements that exist in both “a” and “b” into a third List “c”
  - Constraint: Do not use nested loops

Java Complex Collections
List Common Elements Example

```
10 public class CollectionsListImplements{
11     public static void main(String[] args){
12         List<Integer> a = Arrays.asList(3, 4, 6, 7, 9, 12, 15, 17, 23);
13         List<Integer> b = Arrays.asList(3, 5, 7, 9, 10, 13, 15, 18, 23);
14         List<Integer> c = new ArrayList<Integer>();
15         int A_account = 0, B_account = 0;
16         while(A_account != a.size() && B_account != b.size()) {
17             if(a.get(A_account) < b.get(B_account)) {
18                 A_account++;
19             } else if(b.get(B_account) < a.get(A_account)) {
20                 B_account++;
21             }else {
22                 c.add(a.get(A_account));
23                 A_account++;
24                 B_account++;
25             }
26         }
27         System.out.println(c);
28     }
29 }
```

Java Complex Collections
LinkedList

- In Java, *LinkedList* is a generic class that extends the *AbstractSequentialList* and implements *List*, *Queue*, and *Deque* interfaces. It is a part of the Java Collection API Library. It basically is an implementation of a type of linked list data structure that facilitates the storage of elements. The contents of this storage can grow and shrink at run time as per the requirement. Quite visibly, it is not very different from the other *List* classes, such as *ArrayList*. But, the point of difference is in how the list is maintained at the core.
- The important points about Java LinkedList are:
  - Java LinkedList class can contain duplicate elements.
  - Java LinkedList class maintains insertion order.
  - Java LinkedList class is non synchronized.
  - In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
  - Java LinkedList class can be used as list, stack or queue.

Java Complex Collections
LinkedList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
void addFirst(Object o)	It is used to insert the given element at the beginning of a list.
void addLast(Object o)	It is used to append the given element to the end of a list.
int size()	It is used to return the number of elements in a list
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean contains(Object o)	It is used to return true if the list contains a specified element.
boolean remove(Object o)	It is used to remove the first occurence of the specified element in a list.

Java Complex Collections
LinkedList

Method	Description
Object getFirst()	It is used to return the first element in a list.
Object getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.

Java Complex Collections
LinkedList Constructor

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection c)	It is used to construct a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Java Complex Collections
LinkedList Constructor

```
5  public class LinkedListExample
6  {
7      public static void main(String[] args)
8      { // create an empty linked list
9          LinkedList<String> c1 = new LinkedList<String>();
10         c1.add("Red");
11         c1.add("Green");
12         c1.add("Black");
13         c1.add("White");
14         c1.add("Pink");
15         System.out.println("List of first linked list: " + c1);
16         LinkedList<String> c2 = new LinkedList<String>();
17         c2.add("Red");
18         c2.add("Green");
19         c2.add("Black");
20         c2.add("Pink");
21         System.out.println("List of second linked list: " + c2);
22         // Let join above two list
23         LinkedList<String> a = new LinkedList<String>();
24         a.addAll(c1);
25         a.addAll(c2);
26         System.out.println("New linked list: " + a);
27     }
28 }
```

Java Complex Collections
LinkedList Constructor

- Given two arrays:
  - **LinkedList<Integer> c1 = new LinkedList<Integer>();**
  - **LinkedList<Integer> c2 = new LinkedList<Integer>();**
  - **for(int i = 0; i < 10; i++) {**
  - **if(i != 9) {**
  - **c1.add(i);**
  - **}**
  - **if(i % 2 == 0 || i % 3 == 0) {**
  - **c2.add(i);**
  - **}**
  - **}**
- Sort c1 and c2 into a third LinkedList collection c3 :
  - Expected Output: **0 0 1 2 2 3 3 4 4 5 6 6 7 8 8 9**

Java Complex Collections
HashMap

- HashMap is a container that stores key-value pairs. Each key is associated with one value. Keys in a HashMap must be unique. HashMap is called an associative array or a dictionary in other programming languages. HashMaps take more memory because for each value there is also a key. Deletion and insertion operations take constant time. HashMaps can store null values.
- HashMap is a part of collection in Java since 1.2. It provides the basic implementation of Map interface of Java. It stores the data in (Key,Value) pairs. To access a value you must know its key, otherwise you can't access it. HashMap is known as HashMap because it uses a technique Hashing. Hashing is a technique of converting a large String to small String that represents same String. A shorter value helps in indexing and faster searches. HashSet also uses HashMap internally. It internally uses link list to store key-value pairs. We will know about HashSet in detail in further articles.

Java Complex Collections
HashMap Definition

- **public class HashMap extends AbstractMap implements Map, Cloneable, Serializable**
- HashMap is kept in java.util package. As you can see in above definition of HashMap, it extends an abstract class AbstractMap which also provides an incomplete implementation of Map interface. As you can see it also implements Cloneable and Serializable interface.
- K and V in above definition represents for Key and Value respectively.
- HashMap don't allow duplicate keys, but allows duplicate values. That means A single key can't contain more than 1 value but more than 1 key can contain a single value. HashMap allows null key also but only once and multiple null values. This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time. It is roughly similar to HashTable but is unsynchronized.

Java Complex Collections
HashMap

```
4  public class haspMapExample{
5      public static void main(String[] args){
6          HashMap<String, Integer> map = new HashMap<String, Integer>();
7          print(map);
8          map.put("vishal", 10);
9          map.put("sachin", 30);
10         map.put("vaibhav", 20);
11         System.out.println("Size of map is:- " + map.size());
12         print(map);
13         if (map.containsKey("vishal")) {
14             Integer a = map.get("vishal");
15             System.out.println("value for key \"vishal\" is:- " + a);
16         }
17         map.clear();
18         print(map);
19     }
20
21     public static void print(HashMap<String, Integer> map) {
22         if (map.isEmpty()){
23             System.out.println("map is empty");
24         }
25         else{
26             System.out.println(map);
27         }
28         map.put("hhh", 4444);
29     }
30 }
```

Java Complex Collections

HashMap - Loop through map

```
4  public class haspMapExample{
5      public static void main(String[] args){
6          HashMap<String, Integer> map = new HashMap<String, Integer>();
7          map.put("vishal", 10); map.put("sachin", 30);
8          map.put("vaibhav", 20);
9          System.out.println("Size of map is:- " + map.size());
10         //iterating thru the map by values
11         for(Integer num : map.values()) {
12             System.out.println(num);
13         }
14         //iterating thru the map by keys
15         for(String key : map.keySet()) {
16             System.out.println(key);
17         }
18         //A taste of Lambda, iterating thru the map by keys and values
19         map.forEach((keys, values) ->{//sshhh...
20             System.out.println(keys + " = " + values);
21         });
22     }
23 }
```

Java Complex Collections
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.
- A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null value (HashMap) but some do not (TreeMap).

Method	Description
Object put(Object key, Object value)	It is used to insert an entry in this map.
void putAll(Map map)	It is used to insert the specified map in this map.
Object remove(Object key)	It is used to delete an entry for the specified key.
Object get(Object key)	It is used to return the value for the specified key.

Java Complex Collections
Map

Method	Description
boolean containsKey(Object key)	It is used to search the specified key from this map.
Set keySet()	It is used to return the Set view containing all the keys.
Set entrySet()	It is used to return the Set view containing all the keys and values.

Java Complex Collections
Map Use

```
4  public class MapExamples{
5      public static void main(String[] args){
6          Map<String, String> m1 = new HashMap<String, String>();
7          m1.put("Zara", "8");
8          m1.put("Mahnaz", "31");
9          m1.put("Ayan", "12");
10         m1.put("Daisy", "14");
11         System.out.print("\nMap Elements: ");
12         System.out.print(m1);
13         for(Map.Entry<String, String> i : m1.entrySet()) {
14             System.out.println(i.getKey() + " = " + i.getValue());
15         }
16     }
17 }
```

Java Complex Collections
Map Exercise

- Write a program using map that would act as a Cart system, where you can add an item to the Cart and if the same item is added twice then only the quantity is incremented.
  - Create a class Item with the following member properties: String item_name, Double item_price and Integer item_quatity
  - Create a class MainPoint which has the main method. This class will maintain the map that represents the Cart.
    - Create a function in MainPoint named add. This Function takes in a Item object and would add it to the cart, if the item already exists in the cart then the quantity is increased. Function add returns true when is done.

Java Complex Collections
HashSet

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called hashing.
- HashSet contains unique elements only.

Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(Collection c)	It is used to initialize the hash set by using the elements of the collection c.
HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.

Java Complex Collections
HashSet

Method	Description
void clear()	It is used to remove all of the elements from this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean add(Object o)	It is used to adds the specified element to this set if it is not already present.
boolean isEmpty()	It is used to return true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
Object clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
Iterator iterator()	It is used to return an iterator over the elements in this set.
int size()	It is used to return the number of elements in this set.

Java Complex Collections
HashSet

```
3  public class HashSetExamples
4  {
5      public static void main(String args[]){
6          //Creating HashSet and adding elements
7          HashSet<String> set=new HashSet<String>();
8          set.add("Ravi");
9          set.add("Vijay");
10         set.add("Ravi");
11         set.add("Ajay");
12         //Traversing elements
13         for(String s : set) {
14             System.out.println(s);
15         }
16        }
17 }
```

Java Complex Collections
HashSet

```
 5 public class HasgSetExample2Merge {
 6     public static void main(String[] args) {
 7         // Create a new HashSet of String objects
 8         HashSet<String> setOfStrs = new HashSet<>();
 9         // Adding elements in HashSet
10         setOfStrs.add("hello");
11         setOfStrs.add("abc");
12         setOfStrs.add("time");
13         setOfStrs.add("Hi");
14         System.out.println("setOfStrs = " + setOfStrs);
15         // Convert the String array into a Collection i.e. List
16         List<String> arrList = Arrays.asList("abc", "def", "ghi", "jkl");
17         // Merge the HashSet and List
18         boolean bResult = setOfStrs.addAll(arrList);
19         if(bResult)
20         {
21             System.out.println("Merging of Set & ArrayList Successfull");
22         }   //notice that every element is sorted
23         System.out.println("setOfStrs = " + setOfStrs);
24     }
25 }
```

Java Complex Collections
TreeSet

- TreeSet is similar to HashSet except that it sorts the elements in the ascending order while HashSet doesn't maintain any order. TreeSet allows null element but like HashSet it doesn't allow. Like most of the other collection classes this class is also not synchronized.
- The important points about Java TreeSet class are:
  - Contains unique elements only like HashSet.
  - Access and retrieval times are quiet fast.
  - Maintains ascending order.

Java Complex Collections
TreeSet

Constructor	Description
TreeSet()	It is used to construct an empty tree set that will be sorted in an ascending order according to the natural order of the tree set.
TreeSet(Collection c)	It is used to build a new tree set that contains the elements of the collection c.
TreeSet(Comparator comp)	It is used to construct an empty tree set that will be sorted according to given comparator.
TreeSet(SortedSet ss)	It is used to build a TreeSet that contains the elements of the given SortedSet.

Java Complex Collections
TreeSet

Method	Description
boolean addAll(Collection c)	It is used to add all of the elements in the specified collection to this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean isEmpty()	It is used to return true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void add(Object o)	It is used to add the specified element to this set if it is not already present.
void clear()	It is used to remove all of the elements from this set.
Object clone()	It is used to return a shallow copy of this TreeSet instance.
Object first()	It is used to return the first (lowest) element currently in this sorted set.

Java Complex Collections
TreeSet

Method	Description
Object last()	It is used to return the last (highest) element currently in this sorted set.
int size()	It is used to return the number of elements in this set.

Java Complex Collections
TreeSet

```
3  public class TreeSetEx1
4  {
5      public static void main(String args[]){
6          //Creating HashSet and adding elements
7          TreeSet<String> set=new TreeSet<String>();
8          set.add("Ravi");
9          set.add("Vijay");
10         set.add("Ravi");
11         set.add("Ajay");
12         //Traversing elements
13         for(String s : set) {
14             System.out.println(s);
15         }
16        }
17 }
```

Java Complex Collections
Set

- A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.
- The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
- Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

Java Complex Collections
Set

```
5  public class SetsEx1 {
6      public static void main(String args[]) {
7          int count[] = {34, 22,10,60,30,22};
8          Set<Integer> set = new HashSet<Integer>();
9          try {
10             for(int i = 0; i < 5; i++) {
11                 set.add(count[i]);
12             }
13             System.out.println(set);
14             TreeSet<Integer> sortedSet = new TreeSet<Integer>(set);
15             System.out.println("The sorted list is:");
16             System.out.println(sortedSet);
17             System.out.println("The First element of the Treeset is: "+ (Integer)sortedSet.first());
18             System.out.println("The last element of the Treeset is: "+ (Integer)sortedSet.last());
19         }
20         catch(Exception e) {}
21     }
22 }
```