MODULE 2: The collections and Framework

The collections and Framework

Collections Overview, Recent Changes to Collections, The Collection Interfaces, The Collection Classes, Accessing a collection Via an Iterator, Storing User Defined Classes in Collections, The Random Access Interface, Working With Maps, Comparators, The Collection Algorithms, Why Generic Collections?, The legacy Classes and Interfaces, Parting Thoughts on Collections.

Collections Overview

- ✓ Collections framework was not a part of original Java release. Collections were added to J2SE 1.2. Prior to Java 2.
- ✓ **A collection in java is** a framework that provides architecture to store and manipulate the group of objects.
- ✓ All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.
- ✓ Java Collection simply means a single unit of objects.
- ✓ Java Collection framework provides many **interfaces (Set, List, Queue,Deque etc.)** and **classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).**
- ✓ Framework in java means hierarchy of classes and interfaces.
- ✓ Collections framework is contained in java.util package.

The Collections Framework was designed to meet several goals.

- ✓ First, the frame work had to be **high performance**. The implementations for the fundamental collections (dynamic arrays, linked lists, trees, and hash tables) are highly efficient. You seldom, if ever, need to code one of these "data engines" manually.
- ✓ Second, the framework had to allow different types of collections to work in a similar manner and with a **high degree of interoperability**.
- ✓ Third, extending and /or adapting a collection had to be easy. Toward this end, the entire Collections Framework is built upon a set of standard interfaces.
- ✓ Several standard implementations (such as LinkedList, HashSet, and TreeSet) of these interfaces are provided that you may use as-is.

- ✓ You may also implement your own collection, if you choose.
- ✓ Various special-purpose implementations are created for your convenience, and some partial implementations are provided that make creating your own collection class easier.
- ✓ Finally, mechanisms were added that allow the integration of standard arrays into the Collections Framework.

Recent Changes to Collections

- ✓ Recently, the Collections Framework underwent a fundamental changethat significantly increased its power and streamlined its use.
- ✓ The changes were caused by the
    - addition of generics,
    - autoboxing/unboxing,and
    - the for-each style for loop, by JDK5

Why Collections were made Generic ?

- ✓ Generics added **type safety** to Collection framework.
- ✓ Earlier collections stored **Object class** references which meant any collection could store any type of object.
- ✓ Hence there were chances of storing incompatible types in a collection, which could result in run time mismatch. Hence Generics was introduced through which you can explicitly state the type of object being stored.

Collections and Autoboxing

- ✓ We have studied that Autoboxing converts primitive types into Wrapper class Objects. As collections doesn't store primitive data types(stores only refrences), hence Autoboxing facilitates the storing of primitive data types in collection by boxing it into its wrapper type.

Using for-each loop

- ✓ for-each version of for loop can also be used for traversing each elementof a collection.
- ✓ But this can only be used if we don't want to modify the contents of acollection and we don't want any reverse access.
- ✓ for-each loop can cycle through any collection of object that implementsIterable interface.

Program 1: Example of traversing collection using for-each

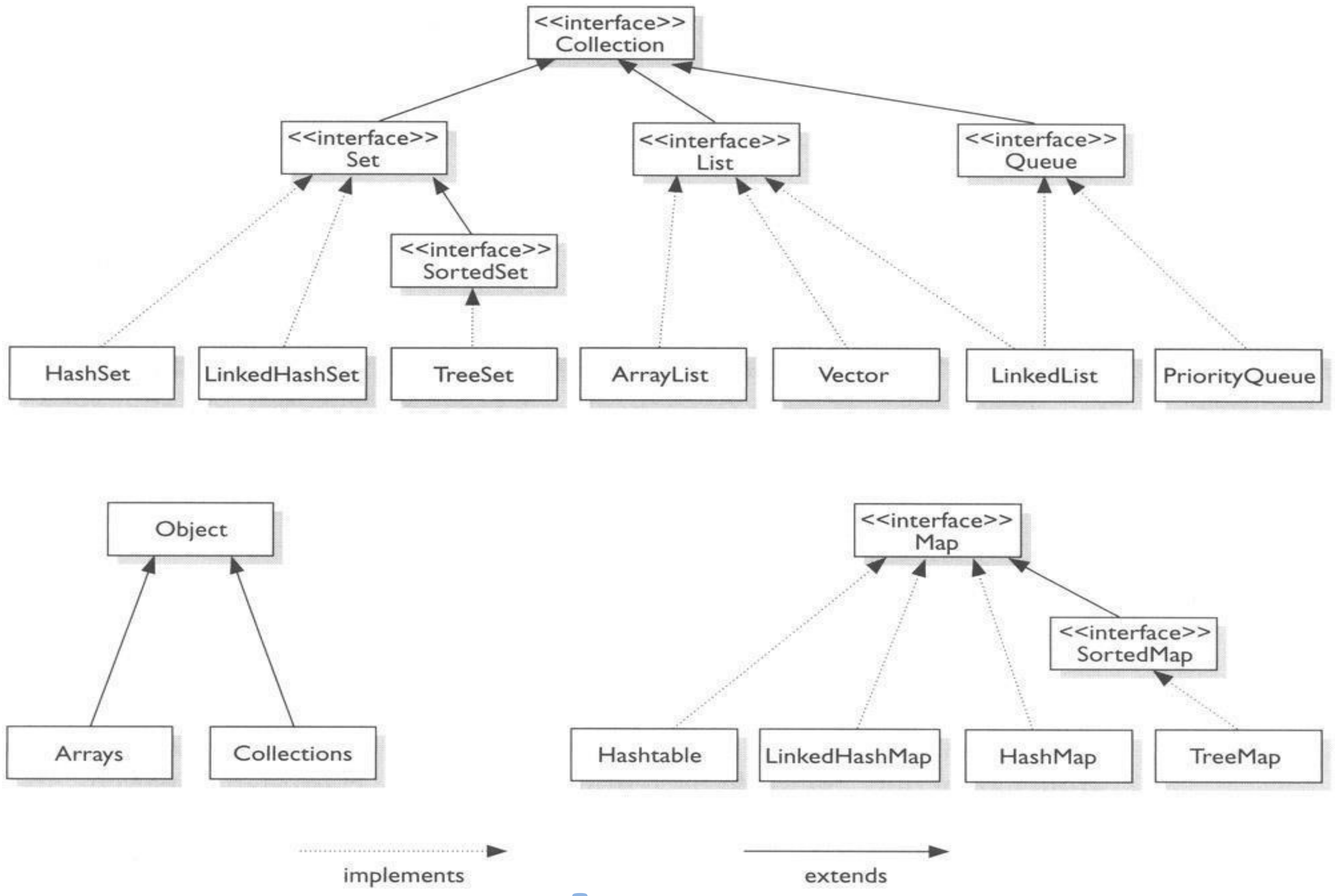
```
import java.util.*; class
ForEachDemo
{
 public static void main(String[] args)
 {
   LinkedList< String> ls = new LinkedList< String>();
  ls.add("a");
  ls.add("b");
  ls.add("c");
  ls.add("d");

  for(String str : ls)
  {
   System.out.print(str+" ");
  }
 }
}
```

Output :a b c d

Collection Interfaces

Interface	**Description**
Collection	Enables you to work with groups of object; it is at the top ofCollection hierarchy
Deque	Extends Queue to handle double ended queue.
List	Extends Collection to handle sequences list of object.
Queue	Extends Collection to handle special kind of list in which element are removed only from the head.
Set	Extends Collection to handle sets, which must contain unique element.
SortedSet	Extends Set to handle sorted set.
NavigableSet	Extends SortedSet to handle retrieval of elements based onclosest-match searches. (Added by Java SE 6.)



Collection Hierarchy

All these Interfaces give several methods which are defined by collectionsclasses which implement these interfaces.

Most commonly thrown Exceptions in Collections Framework

Exception Name	**Description**
UnSupportedOperationException	occurs if a Collection cannot be modified
ClassCastException	occurs when one object is incompatible with another
NullPointerException	occurs when you try to store null object in

	Collection
IllegalArgumentException	thrown if an invalid argument is used
IllegalStateException	thrown if you try to add an element to analready full Collection

Interfaces of Collection Framework

- ✓ The Collections framework has a lot of Interfaces, setting the fundamental nature of various collection classes.

The Collection Interface

1. It is at the top of collection hierarchy and must be implemented by anyclass that defines a collection. Its general declaration is,

interface **Collection** < *E* >

2. Following are some of the commonly used ***methods in this interface***.

Methods	**Description**
boolean add(E obj)	Used to add objects to a collection. Returns true if obj was added to the collection. Returns false if obj is already a member of the collection, or if the collection does not allow duplicates.
boolean addAll(Collection C)	Add all elements of collection C to the invoking collection. Returns true if the element were added. Otherwise, returns false.

boolean remove(Object obj)	To remove an object from collection. Returns true if the element was removed. Otherwise, returnsfalse.
boolean removeAll(Collection C)	Removes all element of collection C from the invoking collection. Returns true if the collection's elements were removed. Otherwise, returns false.
boolean contains(Object obj)	To determine whether an object is present in collection or not. Returns true if obj is an element of the invoking collection. Otherwise, returns false.
boolean isEmpty()	Returns true if collection is empty, else returnsfalse.
int size()	Returns number of elements present in collection.
void clear()	Removes total number of elements from the collection.
Object[] toArray()	Returns an array which consists of the invoking collection elements.
boolean retainAll(Collection c)	Deletes all the elements of invoking collectionexcept the specified collection.
Iterator iterator( )	Returns an iterator for the invoking collection.
boolean equals(Object obj)	Returns true if the invoking collection and obj areequal. Otherwise, returns false.
Object[]toArray(Object array[])	Returns an array containing only those collection

	elements whose type matches of the specified array.

The List Interface

1. It extends the **Collection** Interface, and defines storage as sequence ofelements. Following is its general declaration,

   interface **List** < *E* >

2. Allows random access and insertion, based on position.
3. It allows Duplicate elements.
4. Apart from methods of Collection Interface, it adds following methods ofits own.

Methods	**Description**
Object get(int index)	Returns object stored at the specified index
Object set(int index, E obj)	Stores object at the specified index in thecalling collection
int indexOf(Object obj)	Returns index of first occurrence of obj inthe collection
int lastIndexOf(Object obj)	Returns index of last occurrence of obj in the collection
List subList(int start, int end)	Returns a list containing elements betweenstart and end index in the collection

The Set Interface

1. This interface defines a Set. It extends **Collection** interface and doesn'tallow insertion of duplicate elements. It's general declaration is,

   interface **Set**< *E* >

2. It doesn't define any method of its own. It has two sub interfaces, **SortedSet** and **NavigableSet**.
3. **SortedSet** interface extends **Set** interface and arranges added elementsin an ascending order.
4. **NavigabeSet** interface extends **SortedSet** interface, and allows retrievalof elements based on the closest match to a given value or values.

The Queue Interface

1. It extends **collection** interface and defines behaviour of queue, that isfirst-in, first-out. It's general declaration is,

   interface **Queue**< *E* >

2. There are couple of new and interesting methods added by this interface.Some of them are mentioned in below table.

Methods	**Description**
Object poll()	removes element at the head of the queue andreturns **null** if queue is empty

Object remove()	removes element at the head of the queue and throws **NoSuchElementException** if queue is empty
Object peek()	returns the element at the head of the queue without removing it. Returns **null** if queue is empty
Object element()	same as peek(), but throws **NoSuchElementException** if queue is empty
boolean offer(E obj)	Adds object to queue.

The Dequeue Interface

1. It extends **Queue** interface and implements behaviour of a double-endedqueue. Its general declaration is,

   interface **Dequeue** < *E* >

2. Since it implements Queue interface, it has the same methods asmentioned there.
3. Double ended queues can function as simple queues as well as likestandard Stacks.

The Collection classes

- ✓ There are some standard classes that implements Collection interface.Some of the classes provide full implementations that can be used as it is.
- ✓ Others are an abstract class, which provides skeletal implementationsthat can be used as a starting point for creating concrete collections.

The standard collection classes are:

Class	**Description**
AbstractCollection	Implements most of the Collection interface.
AbstractList	Extends AbstractCollection and implements most of theList interface.
AbstractQueue	Extends AbstractCollection and implements parts of theQueue interface.
AbstractSequentialList	Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.
LinkedList	Implements a linked list by extending AbstractSequentialList
ArrayList	Implements a dynamic array by extending AbstractList
ArrayDeque	Implements a dynamic double-ended queue by extending AbstractCollection and implementing the Deque interface(Added by Java SE 6).
AbstractSet	Extends AbstractCollection and implements most of theSet interface.
EnumSet	Extends AbstractSet for use with enum elements.

HashSet	Extends AbstractSet for use with a hash table.
LinkedHashSet	Extends HashSet to allow insertion-order iterations.
PriorityQueue	Extends AbstractQueue to support a priority-based queue.
TreeSet	Implements a set stored in a tree. Extends AbstractSet.

Note:

1. To use any Collection class in your program, you need to import it in your program. It is contained inside java.util package.
2. Whenever you print any Collection class, it gets printed inside the square brackets [ ].

ArrayList class

- ✓ Simple arrays have fixed size i.e it can store fixed number of elements. But, sometimes you may not know beforehand about the number of elements that you are going to store in your array.
- ✓ In such situations, we can use an ArrayList, which is an array whose size can increase or decrease dynamically.

1. ArrayList class extends **AbstractList** class and implements the **List** interface.
2. ArrayList supports dynamic array that can grow as needed. ArrayList hasthree constructors.
   1. **ArrayList**() *//It creates an empty ArrayList*
   2. **ArrayList**(*Collection* C) *//It creates an ArrayList that isinitialized with elements of the Collection C*

3. **ArrayList**(int *capacity*) *//It creates an ArrayList that has thespecified initial capacity*

3. ArrayLists are created with an initial size. When this size is exceeded,the size of the ArrayList increases automatically.
4. It can contain Duplicate elements and it also maintains the insertionorder.
5. Manipulation is slow because a lot of shifting needs to be occurred if anyelement is removed from the array list.
6. ArrayLists are not synchronized.
7. ArrayList allows random access because it works on the index basis.

Program 2: Example of ArrayList

```
import java.util.*
class Test
{
 public static void main(String[] args)
 {
           ArrayList< String> al = new ArrayList< String>();
           al.add("ab");
           al.add("bc");
           al.add("cd");
          system.out.println(al);
 }
}
```

Output :

```
[ab,bc,cd]
```

Program 3: Example of ArrayListOperation

```
public class ArrayListOperation
{

        public static void main(String[] args)
        {
                ArrayList<String> al = new ArrayList<String>(); System.out.println("Initial size of
                al: " + al.size());

                // Add elements to the array list.
                                al.add("C");
                                al.add("A");
                                al.add("E");
                                al.add("B");
                                al.add("D");
                                al.add("F");
                                System.out.println("Contents of al: " + al);


                                al.add(1,"A2");

                                System.out.println(" updated Contents of al:"+al);


                        System.out.println("Size of al after additions:"+al.size());

                                // Display the array list. System.out.println("Contents of al: "
                                + al);

                                // Remove elements from the array list.
                                al.remove("F");
                                al.remove(2);
                        System.out.println("Size of al after deletions: " + al.size());
                        System.out.println("Contents of al: " + al);

                         ArrayList<String> al1=new ArrayList<String>();
                                 al1.add("mahesh");
                                 al1.add("Vijay");
                                 al1.add("pramod");
```

```
                    System.out.println("Contents of al1: " + al1);

          ArrayList<String> al2=new ArrayList<String>();
                              al2.add("pramod");
                              al2.add("naveen");
                              al2.add("goutham");


           al1.addAll(al2);//adding second list in first list
          System.out.println("Contents of al1: " + al1);


                   al1.retainAll(al2);
                  System.out.println("Contents of al1 using reatain all: " + al1);

                   al1.removeAll(al2); System.out.println("Contents of
                  al1: " + al1);

     }

}
```

Output

```
Size of al after deletions: 5 Contents of al:
[C, A2, E, B, D]
Contents of al1: [mahesh, Vijay, pramod]
Contents of al1: [mahesh, Vijay, pramod, pramod, naveen, goutham] Contents of al1
using reatain all: [pramod, pramod, naveen, goutham]Contents of al1: []
```

Getting Array from an ArrayList

- ✓ **toArray() method** is used to get an array containing all the contents ofthe ArrayList.
- ✓ Following are some reasons for why you can need to obtain an array fromyour ArrayList:

- To obtain faster processing for certain operations.
- To pass an array to methods which do not accept Collection as arguments.
- To integrate and use collections with legacy code.

Program 4: Example of Getting Array from an ArrayListpublic

```
class ArrayListToArray {

        public static void main(String[] args) {
                // Create an array list.
                ArrayList<Integer> al = new ArrayList<Integer>();

                // Add elements to the array list.
                al.add(1);
                al.add(2);
                al.add(3);
                al.add(4);
                System.out.println("Contents of al: " + al);

                // Get the array.
                Integer ia[] = new Integer[al.size()];ia =
                al.toArray(ia);

                // toArray( ) is called and it obtains an array of Integers.

                int sum = 0;

                // Sum the array.
                for(int i : ia)
                        sum += i; System.out.println("Sum
                is: " + sum);

        }
}
```

Output

```
Contents of al: [1, 2, 3, 4]

Sum is: 10
```

Storing User-Defined classes

In the above mentioned example we are storing only string object in ArrayList collection. But You can store any type of object, including object of class that you create in Collection classes.

Program 5: Example of storing User-Defined object

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


public class ArrayListPgm
{

	public static void main(String[] args)
	{
		// TODO Auto-generated method stub

		Student s1=new Student(101,"arun",23); Student
		  s2=new Student(102,"arjun",21); Student s3=new
		  Student(103,"amogh",25);

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

		  System.out.println(al);
```

```
                        //Getting Iterator:iterate the elements of collectionIterator
                        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);
                        }
          }
}
```

Output

```
[Student@49ac272, Student@4c53ccba, Student@11a5ee7c]
101  arun 23
102  arjun 21
103  amogh 25
```

LinkedList class

1. LinkedList class extends **AbstractSequentialList** and implements **List**,**Deque** and **Queue** inteface.
2. LinkedList has two constructors.
   1. LinkedList() *//It creates an empty LinkedList*
   2. LinkedList(Collection C) *//It creates a LinkedList that isinitialized with elements of the Collection c*
3. It can be used as List, stack or Queue as it implements all the relatedinterfaces.
4. They are dynamic in nature i.e it allocates memory when required. Therefore insertion and deletion operations can be easily implemented.
5. It can contain duplicate elements and it is not synchronized.
6. Reverse Traversing is difficult in linked list.
7. In LinkedList, manipulation is fast because no shifting needs to beoccurred.

Program 6: Example of LinkedList class

```
public class LinkedListPgm
{
	public static void main(String[] args)
	{
		// Create a linked list.
		LinkedList<String> ll = new LinkedList<String>();

		// Add elements to the linked list.
		ll.add("F");
		ll.add("B");
		ll.add("D");
		ll.add("E");
		ll.add("C");
		System.out.println(" contents of ll: " + ll);

		ll.addLast("Z");
		ll.addFirst("A");
		ll.add(1, "A2");

		System.out.println("Original contents of ll: " + ll);

		// Remove elements from the linked list.
		ll.remove("F");
		ll.remove(2);

		System.out.println("Contents of ll after deletion: " + ll);

		// Remove first and last elements.


		ll.removeFirst();
		ll.removeLast();

		System.out.println("ll after deleting first and last: " + ll);

		// Get and set a value.
		String val = ll.get(2);
		System.out.println(val);
```

```
              ll.set(3, val + " ");
          }          System.out.println("ll after change: " + ll);

}
```

Output:

```
contents of ll: [F, B, D, E, C]
Original contents of ll: [A, A2, F, B, D, E, C, Z]Contents of ll
after deletion: [A, A2, D, E, C, Z] ll after deleting first and
last: [A2, D, E, C]
E
ll after change: [A2, D, E, E ]
```

Difference between ArrayList and Linked List

- ✓ **ArrayList** and **LinkedList** are the Collection classes, and both of them implements the List interface.
- ✓ The ArrayList class creates the list which is internally stored in a dynamic array that grows or shrinks in size as the elements are added or deleted from it.
- ✓ LinkedList also creates the list which is internally stored in a DoublyLinked List.
- ✓ Both the classes are used to store the elements in the list, but the major difference between both the classes is that ArrayList allows random access to the elements in the list as it operates on an **index-based** data structure.
- ✓ On the other hand, the LinkedList does not allow random access as it does not have indexes to access elements directly, it has to traverse the listto retrieve or access an element from the list.

Some more differences:

- ✓ ArrayList extends AbstarctList class whereas LinkedList extends AbstractSequentialList.
- ✓ AbstractList implements List interface, thus it can behave as a list only whereas LinkedList implements List, Deque and Queue interface, thus it can behave as a Queue and List both.
- ✓ In a list, access to elements is faster in ArrayList as random access isalso possible. Access to LinkedList elements is slower as it follows sequential access only.
- ✓ In a list, manipulation of elements is slower in ArrayList whereas it is faster in LinkedList.

HashSet class

1. HashSet extends **AbstractSet** class and implements the **Set** interface.
2. HashSet has three constructors.
    1. HashSet() *//This creates an empty HashSet*
    2. HashSet(Collection C) *//This creates a HashSet that is initialized with the elements of the Collection C*
    3. HashSet(int capacity) *//This creates a HashSet that has the specified initial capacity*
3. It creates a collection that uses hash table for storage. Hash table stores information by using a mechanism called **hashing**.
4. In hashing, the informational content of a key is used to determine a unique value, called its hash code. The hash code is then used as the index at which the data associated with the key is stored.

5. HashSet does not maintain any order of elements.
6. HashSet contains only unique elements.

Program 7: Example of HashSet class

```
import java.util.*;
public class HashSetDemo
{
	public static void main(String[] args)
	{
		// TODO Auto-generated method stub HashSet<
		 String> hs = new HashSet< String>();hs.add("B");
		  hs.add("A");
		  hs.add("D");
		  hs.add("E");
		  hs.add("C");
		  hs.add("F");
		  hs.add("F");
		  System.out.println(hs);

		  HashSet< Integer> hs1 = new HashSet< Integer>();
		  hs1.add(10);
		  hs1.add(677);
		  hs1.add(5);
		  hs1.add(4);
		  hs1.add(4);

		  System.out.println(hs1);

	}

}
```

Output:

```
[D, E, F, A, B, C]
[4, 5, 677, 10]
```

LinkedHashSet Class

1. LinkedHashSet class extends **HashSet** class
2. LinkedHashSet maintains a linked list of entries in the set.
3. LinkedHashSet stores elements in the order in which elements areinserted i.e it maintains the insertion order.

Program 8: Example of LinkedHashSet class

```
import java.util.*;
public class LinkedHashSetDemo
{
        public static void main(String[] args)
        {
         LinkedHashSet< String> hs = new  LinkedHashSet< String>();hs.add("B");
                        hs.add("A");
                        hs.add("D");
                        hs.add("E");
                        hs.add("C");
                        hs.add("F");
                        System.out.println(hs);
  LinkedHashSet< Integer> hs1 = new LinkedHashSet< Integer>();
                        hs1.add(10);
                        hs1.add(677);
                        hs1.add(5);
                        hs1.add(4);
                        hs1.add(4);
                       System.out.println(hs1);


        }
}
```

Output :

```
[B, A, D, E, C, F]
[10, 677, 5, 4]
```

Questions

1. What is collection in java? Mention some of the recent changes to Collections.
2. List and give brief description on Java Collection Interfaces and Classes.
3. Implement a java ArrayList collection with add(), remove(), addAll(), reatainAll(), removeAll() and write the output of all the cases.
4. Write a java program to Obtaining an Array from an ArrayList.
5. Implement a java LinkedList collection with add(), addFirst(), addLast(),add and remove elements with the specified Index, get and set methods.
6. Explain with an example program HashSet , LinkedHashSet class andStoring User-Defined Classes in Collections.