SLIIT
Discover Your Future

Java Collection Framework

Object Oriented Programming (OOP)

Year 2 – Semester 1

Collections in Java

The **Collections Framework** is a sophisticated hierarchy of interfaces and classes that provide state-of-art technology for managing groups of objects.

Java Reference Book 09: Page 497

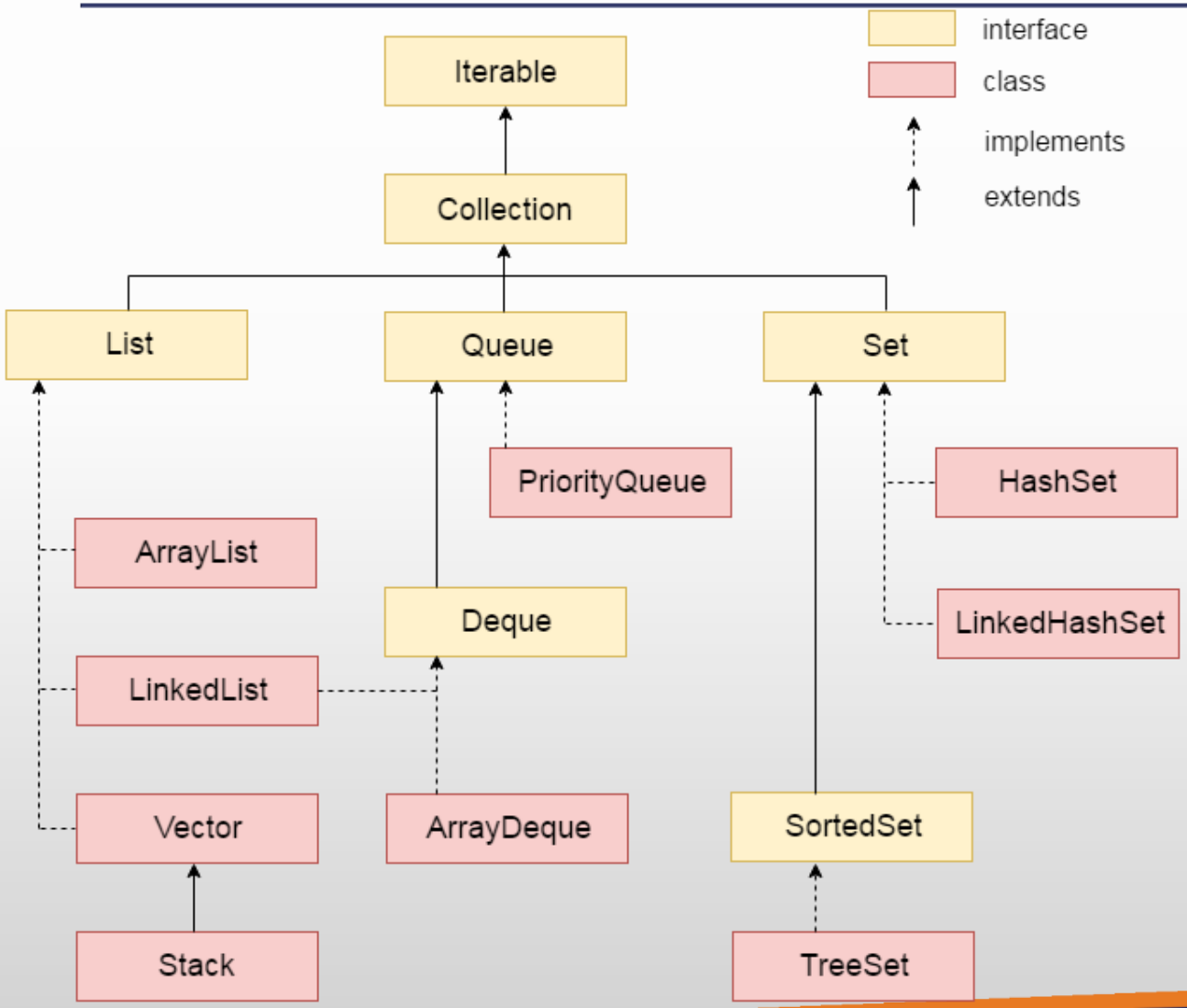
The primary advantages of a collections framework are;

- Reduces programming effort
- Increases performance
- Provides interoperability between unrelated APIs
- Reduces the effort required to learn APIs
- Reduces the effort required to design and implement APIs
- Fosters software reuse

Generics make Collections type safe. Before generics, collections stored Object references; can store any type of object. Thus avoids run-time mismatch errors.

SLIIT
Discover Your Future

Hierarchy of Collections Framework



The **java.util** package contains all the classes and interfaces for Collection framework

SLIIT
Discover Your Future

Collection Interface

- Collection is the foundation upon which the Collection Framework is built on.

Method	Description
public boolean add(Object element)	is used to insert an element in this collection.
public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
public boolean remove(Object element)	is used to delete an element from this collection.
public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
public int size()	return the total number of elements in the collection.
public void clear()	removes the total no of element from the collection.
public boolean contains(Object element)	is used to search an element.
public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
public Iterator iterator()	returns an iterator.
public Object[] toArray()	converts collection into array.
public boolean isEmpty()	checks if collection is empty.
public boolean equals(Object element)	matches two collection.
public int hashCode()	returns the hashcode number for collection.

List Interface

- Java.util.List is a child interface of Collection.
- 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.

if you want you can remove or add in between elements

- List Interface is implemented by ArrayList, LinkedList, Vector and Stack classes.

Queue Interface

- The java.util.Queue is a subtype of java.util.Collection interface.
- It is an ordered list of objects with its use limited to **inserting elements at the end of list and deleting elements from the start of list. It follows First In First Out (FIFO) principle.**
- Queue Interface is implemented by PriorityQueue class

Problem

Think of a School management application. You have a list of students stored in Database. To display those names on the user interface, You have to make a call to the database and store these elements and populate those list elements on UI.

What is the best approach you are suggesting?

Answer – An Array?

```
String arr[]=new String[5];
arr[0]="Anne";
arr[1]="Peter";
arr[2]="Shenan";
arr[3]="Pete";
arr[4]="Diana";
```

Problems?

- Arrays are of fixed length. You can not change the size of the arrays once they are created.
- You can not accommodate an extra element in an array after they are created.
- Memory is allocated to an array during it's creation only, much before the actual elements are added to it.

Answer – **ArrayList Class**

- ArrayList class extends the AbstractList class and implements the List interface.
- ArrayList support dynamic arrays (can increase and decrease size dynamically).
- Elements can be inserted at or deleted from a particular position
- ArrayList class has many methods to manipulate the stored objects
- If generics are not used, ArrayList can hold any type of objects.

Example : ArrayList

Ref: Java Complete Reference Pg: 512

```
import java.util.*;

class ArrayListDemo {
   public static void main(String args[]) {
     // Create an array list.
     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");
     al.add(1, "A2");

     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);
   }
```

ArrayListDemo.java

Obtaining Array from an ArrayList

Reasons for converting array to an ArrayList

- To obtain faster processing time for certain operations
- To pass an array to a method that is not overloaded to accept a collection
- To integrated collection-based code with legacy code that don't understand collections

ArrayList got Last Come first out method

Example : ArrayList to Array

Ref: Java Complete Reference Pg: 514

```
import java.util.*;

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);

     int sum = 0;

      // Sum the array.
         for(int i =0 ; i< ia.length ; i++)
                sum += i;

   System.out.println("Sum is: " + sum);
  }
}
```

ArrayListToArray.java

Example : forEach Loop for Iterating Over Collections

```
import java.util.ArrayList;

public class ForEachExample {

    public static void main(String[] args) {
        ArrayList<String> items = new ArrayList<>();
        items.add("A");
        items.add("B");
        items.add("C");
        items.add("D");
        items.add("E");

        for (String item : items) {
            System.out.println(item);
        }
    }

}
```

ForEachExample.java

Problem

Think of a scenario where plates stacked over one another in a canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time.

What is the best approach you are suggesting?

Answer – An ArrayList?

Problems?

- In ArrayList , Elements can be inserted at or deleted from a particular position
- What will happen if we remove one middle plate from the stack?

Answer – **Stack Class**

- **Stack** is a subclass of **Vector**
- Stacks are dynamic data structures that follow the **Last In First Out (LIFO)** principle

SLIIT
Discover Your Future

Exercise 2

Problem

Think of a scenario of a Bank line where people who come first will done his transaction first.

What is the best approach you are suggesting?

Answer – **Queue Interface**

- The **Queue** interface extends **Collection**
- Defines queue data structure which is normally **First-In-First-Out**
- Elements are added from one end and elements are deleted from another end
- Most common classes are the PriorityQueue and LinkedList in Java
- In Priority Queue, elements are removed from one end, but elements are added according to the order defined by the supplied comparator.

SLIIT
Discover Your Future

Exercise 3

Problem

Think of an application that require a unique User ID as a input

What is the best approach you are suggesting?

Answer -Set Interface

- The java.util.Set interface is a subtype of Collection interface.
- A **Set is a Collection that cannot contain duplicate elements**. It models the mathematical set abstraction.
- Set interface is implemented by SortedSet interface and Hashset and LinkedHashSet classes.
- SortedSet interface declares the behavior of a set sorted in ascending order.
- TreeSet class implements this interface and TreeSet class are stored in ascending order.
- HashSet class stores the elements by using a mechanism called **hashing** and contains unique elements only

Problem

Think of a scenario where you ordering a 3-topping pizza.
Whether you say ‘Pepperoni, mushrooms, and onions, please’
or ‘Yoo hoo! Make that mushrooms, pepperoni, and onions’

What is the best approach you are suggesting?

Answer - **HashSet Class**

- HashSet extends AbstractSet and implements the Set interface.
- It creates a collection that uses a hash table for storage.
- No duplication and unordered
- 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. The transformation of the key into its hash code is performed automatically

Example : HashSet

Ref: Java Complete Reference Pg: 517

```
import java.util.HashSet;

public class HashSetDemo {

   public static void main(String args[]) {
// Create a hash set.
      HashSet<String> hs = new HashSet<String>();

      // Add elements to the hash set.
      hs.add("Beta");
      hs.add("Alpha");
      hs.add("Eta");
      hs.add("Gamma");
      hs.add("Epsilon");
      hs.add("Omega");

      System.out.println(hs);
   }
}
```

HashSetDemo.java

SLIIT
Discover Your Future

Exercise 4 – Queston 1

Problem

Think of a scenario where you store student details including ID and the marks.

What is the best approach you are suggesting?

Answer - **TreeSet Class**

- **TreeSet** extends **AbstractSet** and implements the **NavigableSet** interface

- Objects are stored in sorted, ascending order

SLIIT
Discover Your Future

Exercise 4 – Question 2

Problem

Think of a cache, where in if new data comes in we overwrite the existing record using the key. So basically the cache would be used to store the most recent state.

What is the best approach you are suggesting?

Answer - **Maps**

- A map is an object that stores associations between keys and values, or key/value pairs. Given a key, you can find its value.
- Both keys and values are objects. The keys must be unique, but the values may be duplicated.
- Some maps can accept a null key and null values, others cannot.
- Map is useful if you have to search, update or delete elements on the basis of key
- **Maps are not part of the Collections Framework**

The Map Classes

Several classes provide implementations of the map interfaces. The classes that can be used for maps are summarized here:

Class	Description
AbstractMap	Implements most of the **Map** interface.
EnumMap	Extends **AbstractMap** for use with **enum** keys.
HashMap	Extends **AbstractMap** to use a hash table.
TreeMap	Extends **AbstractMap** to use a tree.
WeakHashMap	Extends **AbstractMap** to use a hash table with weak keys.
LinkedHashMap	Extends **HashMap** to allow insertion-order iterations.
IdentityHashMap	Extends **AbstractMap** and uses reference equality when comparing documents.

Ref: Java Complete Reference Pg: 537

HashMap Class

- The HashMap class extends AbstractMap and implements the Map interface. It uses a hash table to store the map.

- HashMap is a generic class that has this declaration:

  class HashMap<K, V>

  K specifies the type of keys, and V specifies the type of values.

Example : HashMap

Ref: Java Complete Reference Pg: 537

```
public class HashMapDemo {
   public static void main(String args[]) {
      // Create a hash map.
      HashMap<String, Double> hm = new HashMap<String, Double>();

      // Put elements to the map
      hm.put("John Doe", new Double(3434.34));
      hm.put("Tom Smith", new Double(123.22));
      hm.put("Jane Baker", new Double(1378.00));
      hm.put("Tod Hall", new Double(99.22));
      hm.put("Ralph Smith", new Double(-19.08));

      // Get a set of the entries.
      Set<Map.Entry<String, Double>> set = hm.entrySet();

      // Display the set.
      for (Map.Entry<String, Double> me : set) {
         System.out.print(me.getKey() + ": ");
         System.out.println(me.getValue());
      }

      // Deposit 1000 into John Doe's account.
      double balance = hm.get("John Doe");
      hm.put("John Doe", balance + 1000);

      System.out.println("John Doe's new balance: " + hm.get("John Doe"));
   } }
```

HashMapDemo.java

Hashmap

- Values in hash map are not ordered or not sorted
- Remove duplicates in the key

Exercise 5

LinkedHashMap

- HashMap with additional feature that it maintains **insertion order**
- contains values based on the key
- Remove duplicates in the key
- When Display, the order is not guaranteed

Exercise 6

TreeMap

- contains values based on the key
- contains only unique elements
- maintains ascending orders
- provides an efficient means of storing key-value pairs in sorted order

SLIIT
Discover Your Future

Exercise 7

SLIIT
Discover Your Future

Reference

- Java Complete Reference – 9th Edition
- Java T- Point:
  https://www.javatpoint.com/collections-in-java
- Java Collections Tutorial - Jakob Jenkov
  http://tutorials.jenkov.com/java-collections/index.html