Advanced Programming

Week 1

Java Collections

- Introduction
- Java Collection Framework
- Hierarchy of Collection Framework
- Collection interface
- Iterator interface

Tilahun Melak(PhD)

September, 2025

Objectives

At the end of this lecture students will be able to :

- Describe Java Collection Framework
- Identify Hierarchy of Collection Framework
- Identify Collection interfaces and Classes
- Use Iterator Interface

Introduction

- In order to handle group of objects we can use array of objects. If we have a class called Employ with members name and id, if we want to store details of 10 Employees, create an array of object to hold 10 Employ details.

  Employ ob [] = new Employ [10];

- The main limitations on the array implementation like the above example include;
  - We cannot store different class objects into same array.
  - Inserting element at the end of array is easy but at the middle is difficult.
  - After retrieving the elements from the array, in order to process the elements we don't have any methods.
- The solution is Collections

Introduction cntd…

- Collections is sometimes called Containers
- An object that groups multiple elements into a single unit
- Used to:
  - Store data
  - Retrieve data
  - Manipulate data
  - Pass data between methods
- Represent natural groups of items, e.g.:
  - Poker hand → collection of cards
  - Mail folder → collection of letters
  - Telephone directory → mapping of names to phone numbers

Oracle. (n.d.). Introduction to collections. The Java™ Tutorials.

Java Collections Framework

- It is a Pre packaged Implementation
- Unified architecture for representing and manipulating collections.
- Java collection framework comprises of ready to use components.
- In many cases it is no need to create your own collection but there is an option if you want to.
- The collection components are standard implementation that doesn't require us to worry about details of their implementation

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Java Collections Framework cntd..

- All collections frameworks contain:
  - **Interfaces:**
    - are abstract data types that represent collections.
    - allow collections to be manipulated independently of the details of their representation.
  - **Implementations, i.e., Classes:**
    - are the concrete implementations of the collection interfaces.
    - are reusable data structures.

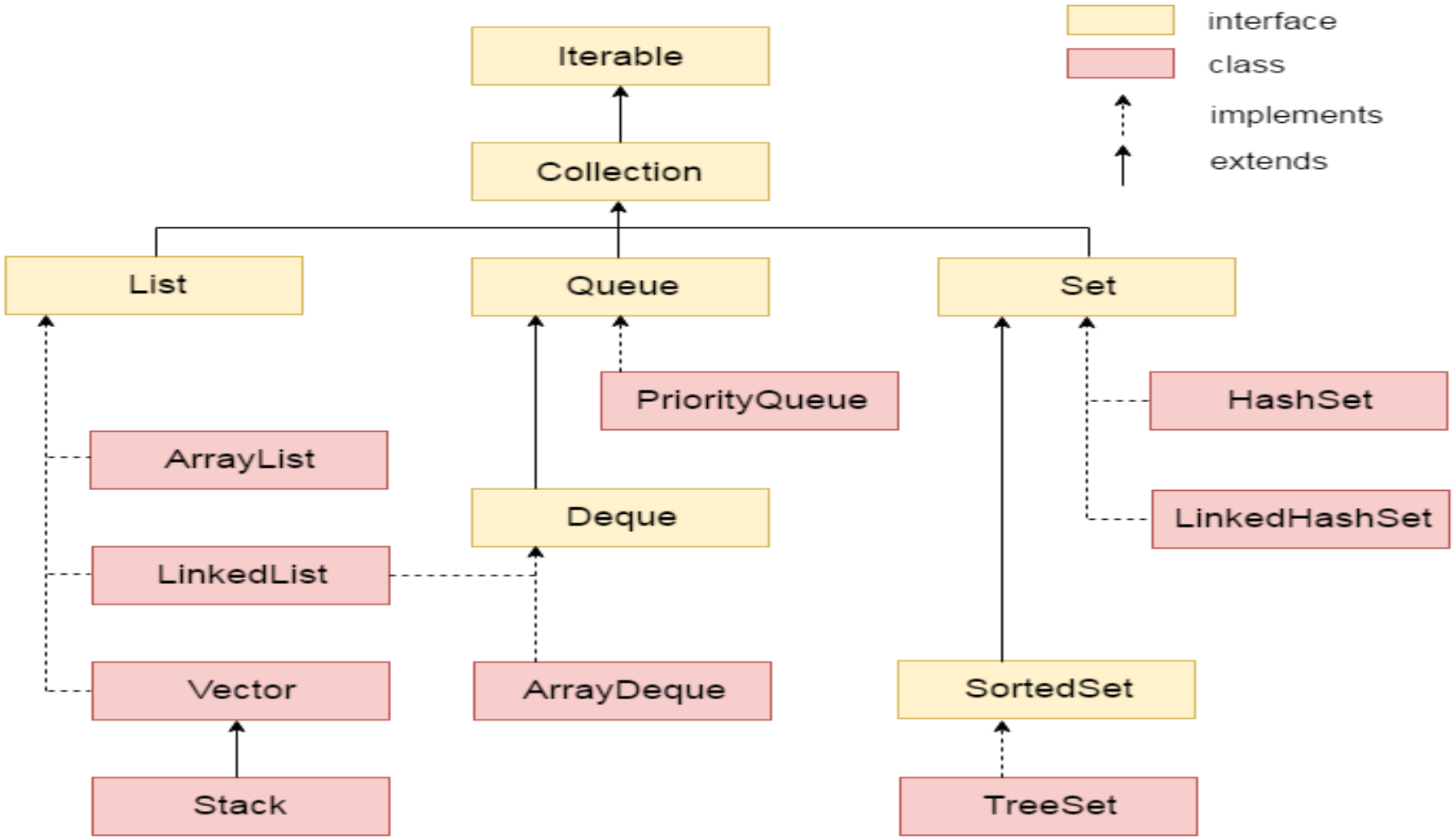
Oracle. (n.d.). *Introduction to collections*. The Java™ Tutorials.

Java Collections Framework cntd..

- **Algorithms:**
  - are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces.
  - are said to be **polymorphic**: that is, the same method can be used on many different implementations of the appropriate collection interface.

Oracle. (n.d.). *Introduction to collections*. The Java™ Tutorials.

Hierarchy of Collection Framework



Adapted from "Collections overview," by Oracle, n.d., in Java Platform, Standard Edition 8 Documentation

Hierarchy of Collection Framework cntd..

- Collection
  - The root of the collection hierarchy.
  - A collection represents a group of objects known as its *elements*.
  - It is parent of all collections framework.
- Set
  - A Set represents a group of elements (objects) arranged just like an array.
  - The set will grow dynamically when the elements are stored into it.
  - A set will not allow duplicate elements.
- Sorted Set:
  - Ordered version of the set interface.

Hierarchy of Collection Framework cntd..

- List
  - Lists are like sets but it is an ordered collection and can contain duplicate elements.
- Queue
  - Queue provides variety of implementation in addition to following the FIFO (First In First Out) principles.
- Map
  - Maps store elements in the form of key value pairs.
  - If the key is provided its corresponding value can be obtained.
  - It does not allow duplicate keys.
- Sorted Map
  - Maintains ascending order of keys.

Collections

- A collection is an object which can store group of other objects.
  - A collection object has a class called Collection class.
  - All the collection classes are available in the package called 'java.util' (util stands for utility).
  - Group of collection classes is called a Collection Framework.
  - A collection object does not store the physical copies of other objects; it stores references of other objects.
  - All the collection classes in java.util package are the implementation classes of different interfaces.

Collections cntd…

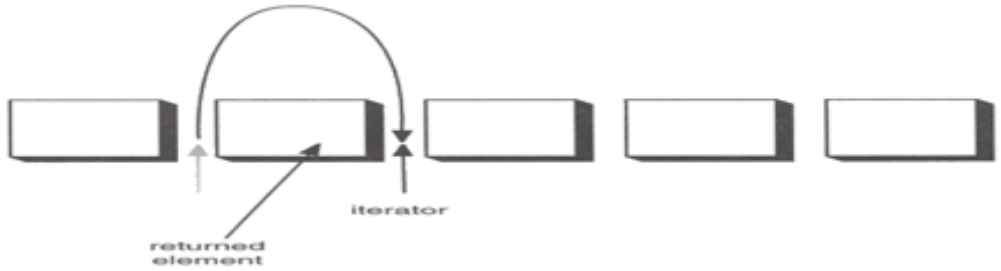
- Some examples of collections are
  - the cards you hold in a card game,
  - your favorite songs stored in your computer,
  - the members of a sports team and
  - the real-estate records in your local registry of deeds (which map book numbers and page numbers to property owners).

Collections Interface

- In the collection interface the following fundamental methods are defined:
  - **int size();**
  - **boolean isEmpty();**
  - **boolean contains(Object element);**
  - **boolean add(Object element); // Optional**
  - **boolean remove(Object element); // Optional**
  - **Iterator iterator();**
- The basic behaviors of a collection are defined with these methods are enough to define the basic behavior of a collection
- Iterator is used to navigate through the elements of a collection.

Iterator Interface

- In the Iterator interface the following three fundamental methods are defined:
  - **Object next()**
  - **boolean hasNext()**
  - **void remove()**
- Access to the contents of the collection are realized with the aid of these methods
- An Iterator locates a position within collection
- Each call to next() "reads" an element from the collection
- Then you can use it or remove it



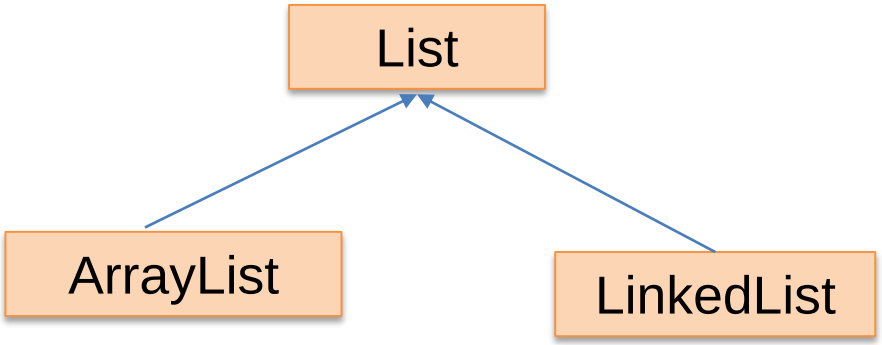
Example

```
public class CollectionDemo  {
    public static void main(String[] args) {
     Collection c;
     c = new ArrayList();
     System.out.println(c.getClass().getName());
     for (int i=1; i <= 10; i++) {
                c.add(i + " * " + i + " = "+i*i);
     }
     Iterator iter = c.iterator();
     while (iter.hasNext())
                System.out.println(iter.next());
    }
}
```

Output:

```
java.util.ArrayList
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
10 * 10 = 100
```

List Interface



- **The List Interface**
  - Adds the notion of **order** to a collection
  - User controls **where elements are inserted**
  - Allows **duplicate elements**
  - Provides **ListIterator** to traverse elements

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

ListIterator Interface

- Extends the Iterator interface
- The following three fundamental methods are defined
  - **void add(Object o)**
  - **boolean hasPrevious()**
  - **Object previous()**
- Defines the basic behavior of an ordered list in addition to the three methods
- A ListIterator knows position within list.

List Implementations

- ArrayList
  - Low cost random access
  - High cost insert and delete
    - Inserting or deleting an element between existing elements of an ArrayList is an inefficient operation. All elements after the newly deleted or inserted element must be updated, which could be an expensive operation in a collection with a large number of elements.
  - Array that resizes if needed

Deitel, P. J., & Deitel, H. M. (2009). *Java™ How to Program* (7th ed.). Pearson Education.

List Implementations cntd…

- LinkedList
  - sequential access
  - low cost insert and delete
    - A LinkedList enables efficient insertion (or removal) of elements in the middle of a collection.
  - high cost random access

Deitel, P. J., & Deitel, H. M. (2009). *Java™ How to Program* (7th ed.). Pearson Education.

ArrayList Overview

- The ArrayList class extends AbstractList and implements the List interface. ArrayList supports dynamic arrays that can grow as needed.
- Standard Java arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold.
- 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 Overview cntd…

- The ArrayList class supports three constructors.
- The first constructor builds an empty array list: **ArrayList( )**
- The following constructor builds an array list that is initialized with the elements of the collection c.
- **ArrayList(Collection c)**
- The following 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 array list.
- **ArrayList(int capacity)**

ArrayList Methods

- The indexed get and set methods of the List interface are appropriate to use since ArrayLists are backed by an array
  - **Object get(int index)**
  - **Object set(int index, Object element)**
- Indexed add and remove are provided, but can be costly if used frequently
  - **void add(int index, Object element)**
  - **Object remove(int index)**
- May want to resize in one shot if adding many elements
  - **void ensureCapacity(int minCapacity)**

Example: ArrayList

```
import java.util.*;
class ArrayListDemo
{ public static void main(String args[])
  { ArrayList <String> al = new ArrayList<String>();
    al.add (“Africa"); al.add ("North America");
    al.add ("South America"); al.add (“Asia");
    al.add ("Europe");
    al.add (1,"Australia");
    al.add (2,"Antarctica");
    System.out.print("Size of the Array List is: " + al.size ());
    System.out.print("\nRetrieving elements in ArrayList using   Iterator:");
   Iterator it = al.iterator ();
   while (it.hasNext () )
     System.out.print (it.next () + "\t");
  }}
```

LinkedList Overview

- Stores each element in a node
- Each node stores a link to the next and previous nodes
- Insertion and removal are inexpensive
  - just update the links in the surrounding nodes
- Linear traversal is inexpensive
- Random access is expensive
  - Start from beginning or end and traverse each node while counting

LinkedList Methods

- A **List** maintains elements in sequential order, so it should be accessed accordingly.
- The method **listIterator()** provides a **ListIterator**.
- A **ListIterator** is aware of the current position in the list.
- You can use **add()** from **ListIterator** to insert an element at a specific position.
- You can use **remove()** from **ListIterator** to delete an element at a specific position.

Deitel, P. J., & Deitel, H. M. (2009). *Java™ How to Program* (7th ed.). Pearson Education.

LinkedList Methods cntd…

- LinkedList provides several useful methods:
  - voidaddFirst(Object o) – inserts an element at the beginning
  - voidaddLast(Object o) – inserts an element at the end
  - ObjectgetFirst() – retrieves the first element
  - ObjectgetLast() – retrieves the last element
  - ObjectremoveFirst() – removes the first element
  - ObjectremoveLast() – removes the last element

Deitel, P. J., & Deitel, H. M. (2009). *Java™ How to Program* (7th ed.). Pearson Education.

Example: LinkedList

```
import java.util.*;
class LinkedDemo
{ public static void main(String args[])
 { LinkedList <String> ll = new LinkedList<String>();
   ll.add (" Africa");
   ll.add ("North America");
   ll.add ("South America");
   ll.add (“Asia");
   ll.addFirst ("Europe");
   ll.add (1,"Australia");
   ll.add (2,"Antarctica");
   System.out.println("Elements in Linked List is : " + ll);
   System.out.println("Size of the Linked List is : " + ll.size() );}}
```

Exercise 1

Write the output of this program

```
1  import java.util.*;
2      class ListTest{
3          public static void main(String [] args){
4              Integer [] n = {10, 25, 100, 78,  2, 36, 1};
5              ArrayList <Integer> al = new ArrayList <Integer>();
6              for(int i : n)
7                  al.add(i);
8              System.out.printf(al+",");
9          }}
10
```

Exercise 2

You are given with the following program

```
1  import java.util.*;
2  public class LinkedLTest {
3
4      public static void main(String[] args) {
5          List<String> student = new LinkedList<String>();
6          student.add("Chaltu");
7          student.add("Blen");
8          student.add("Benti");
9          student.add("Elnathan");
10         System.out.println(student+",");
11     }
12
13 }
14
```

Exercise 2 cntd…

Based on the above program:

1. Execute the program and write its output.
2. Add students to the first and end of the list.
2. Write a code that adds student named **“Abebe”** to the second position of the list.
3. Create your own list of 10 elements and Merge it to the student list.

Summary

- In todays lecture we have discussed about;
  - Overview of Collection and Its Importance
  - Detail of Collection Framework
  - Array List
  - Linked List
  - Examples
  - Exercises

References

- Oracle. (n.d.). Collections overview. In Java Platform, Standard Edition 8 Documentation. Oracle.
- Deitel, H. M., & Deitel, P. J. (2006). *Java*™ *How to Program* (7th ed.). Prentice Hall.