Lecture 10

ArrayList

- The **ArrayList** class is a resizable array, which can be found in the **java.util** package.

- The difference between a built-in array and an **ArrayList** in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an **ArrayList** whenever you want.

```
// import the ArrayList class
import java.util.ArrayList;
// Create an ArrayList object
ArrayList<String> cars = new ArrayList<String>();
```

ArrayList

- The important points about the Java ArrayList class are:
- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized
- Java ArrayList allows random access because the array works on an index basis.
- In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases.
- Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.

Hierarchy of ArrayList class

- As shown in the above diagram, the Java ArrayList class extends AbstractList class which implements the List interface.

- The List interface extends the Collection and Iterable interfaces in hierarchical order.

Add Items

- The **ArrayList** class has many useful methods. For example, to add elements to the **ArrayList**, use the **add()** method

```
// import java.util.ArrayList;
public class Main {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add(“Volvo");
        cars.add(“BMW");
        cars.add(“Ford");
        cars.add(“Mazda");
        System.out.println(cars);
    }
}
```

Access an Item

- To access an element in the **ArrayList**, use the **get()** method and refer to the index number:

```
cars.get(0);
```

Change an Item

- To modify an element, use the **set()** method and refer to the index number.

```
cars.set(0, "Opel");
```

Remove an Item

- To remove an element, use the **remove()** method and refer to the index number.

```
cars.remove(0);
```

- To To remove all the elements in the **ArrayList**, use the **clear()** method:

```
cars.clear();
```

ArrayList Size

- To find out how many elements an ArrayList have, use the **size** method.

```
cars.size();
```

Ways to iterate the elements

- There are various ways to traverse the collection elements:

    1. By Iterator interface.
    2. By for-each loop.
    3. By ListIterator interface.
    4. By for loop.
    5. By forEach() method.
    6. By forEachRemaining() method.

Loop Through an ArrayList

- Loop through the elements of an **ArrayList** with a **for** loop, and use the **size()** method to specify how many times the loop should run.

```
public class Main {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add(“Volvo");
        cars.add(“BMW");
        cars.add(“Ford");
        cars.add(“Mazda");
        for (int i = 0; i < cars.size(); i++) {
            System.out.println(cars.get(i));
        }
        for (String i : cars) {
            System.out.println(i);
        }

    }
}
```

Other Types

- Elements in an ArrayList are actually objects. In the examples above, we created elements (objects) of type "String". Remember that a String in Java is an object (not a primitive type).

- To use other types, such as int, you must specify an equivalent wrapper class: Integer. For other primitive types, use: Boolean for boolean, Character for char, Double for double, etc

Sort an ArrayList

- Another useful class in the **java.util** package is the **Collections** class, which include the **sort()** method for sorting lists alphabetically or numerically:

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add(“Volvo");
        cars.add(“BMW");
        cars.add(“Ford");
        cars.add(“Mazda”);
        Collections.sort(cars);  // Sort cars
        for (int i = 0; i < cars.size(); i++) {
            System.out.println(cars.get(i));
        }
    }
}
```

Java LinkedList

- The **LinkedList** class is almost identical to the **ArrayList.**
- Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

LinkedList

- 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 occur.
- Java LinkedList class can be used as a list, stack or queue.

ArrayList vs LinkedList

- When the rate of addition or removal rate is more than the read scenarios, then go for the LinkedList. On the other hand, when the frequency of the read scenarios is more than the addition or removal rate, then ArrayList takes precedence over LinkedList.

- Since the elements of an ArrayList are stored more compact as compared to a LinkedList; therefore, the ArrayList is more cache-friendly as compared to the LinkedList. Thus, chances for the cache miss are less in an ArrayList as compared to a LinkedList. Generally, it is considered that a LinkedList is poor in cache-locality.

- Memory overhead in the LinkedList is more as compared to the ArrayList. It is because, in a LinkedList, we have two extra links (next and previous) as it is required to store the address of the previous and the next nodes, and these links consume extra space. Such links are not present in an ArrayList.

HashMap

- A **HashMap** store items in "**key/value**" pairs, and you can access them by an index of another type.

```
// import the HashMap class
import java.util.HashMap;
HashMap<String, String> capitalCities;
capitalCities = new HashMap<String, String>();
```

HashMap

- HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key.

- Since Java 5, it is denoted as HashMap<K,V>, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.

HashMap

- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.
- Java HashMap may have one null key and multiple null values.
- Java HashMap is non synchronized.
- Java HashMap maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Method of HashMap

- The **HashMap** class has many useful methods, to add items to it, use the **put()** method.

- To access a value in the **HashMap**, use the **get()** method and refer to its key.

- To remove an item, use the **remove()** method and refer to the key.

- To remove all items, use the **clear()** method.

- To find out how many items there are, use the **size()** method.

Loop Through a HashMap

- Loop through the items of a HashMap with a **for-each** loop.
- **Note:** Use the keySet() method if you only want the keys, and use the values() method if you only want the values

```
// Print keys
for (String i : capitalCities.keySet()) {
    System.out.println(i);
}
// Print values
for (String i : capitalCities.values()) {
    System.out.println(i);
}
// Print keys and values
for (String i : capitalCities.keySet()) {
    System.out.println("key: " + i + " value: " + capitalCities.get(i));
}
```

HashSet

- A HashSet is a collection of items where every item is unique, and it is found in the **java.util** package

```
// Import the HashSet class
import java.util.HashSet;
HashSet<String> cars = new HashSet<String>();
```

HashSet

- HashSet stores the elements by using a mechanism called **hashing.**
- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.
- The initial default capacity of HashSet is 16, and the load factor is 0.75.

HashSet

- A list can contain duplicate elements whereas Set contains unique elements only.
- The HashSet class extends AbstractSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

Method of HashSet

- The **HashSet** class has many useful methods, to add items to it, use the **add()** method.

- To check whether an item exists in a HashSet, use the **contains()** method.

- To remove an item, use the **remove()** method.

- To remove all items, use the **clear()** method

Java Collection