Java Collection Framework

Mahboob Ali

What is a Data Structure?

- A data structure is a collection of data organized in some fashion.
- The structure not only stores data, but also supports operations for accessing and manipulating the data.

The Collection

- A *collection* is a container object that represents a group of objects, often referred to as *elements*.

- Collections are used to store, retrieve, manipulate and communicate with the aggregated data.

- *Examples*
  - *A poker hand (collection of cards)*
  - *A mail folder (collection of letters)*
  - *A telephone directory (mapping of names to phone numbers)*

Java Collection Framework

Consist on three parts:

- Interfaces
- Implementation classes
- Algorithms

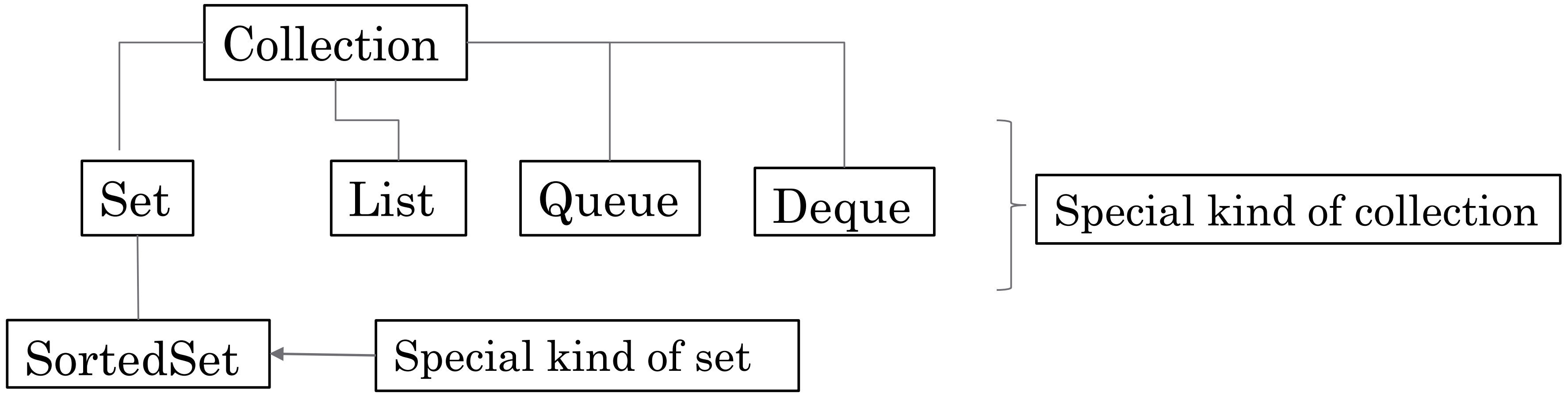
Interfaces

- Interfaces provides the abstract data type to represent collection.

java.util.Collection

The Collection interface is the root interface for manipulating a collection of objects.

- *java.util* contains all the collections framework interfaces.



Syntax

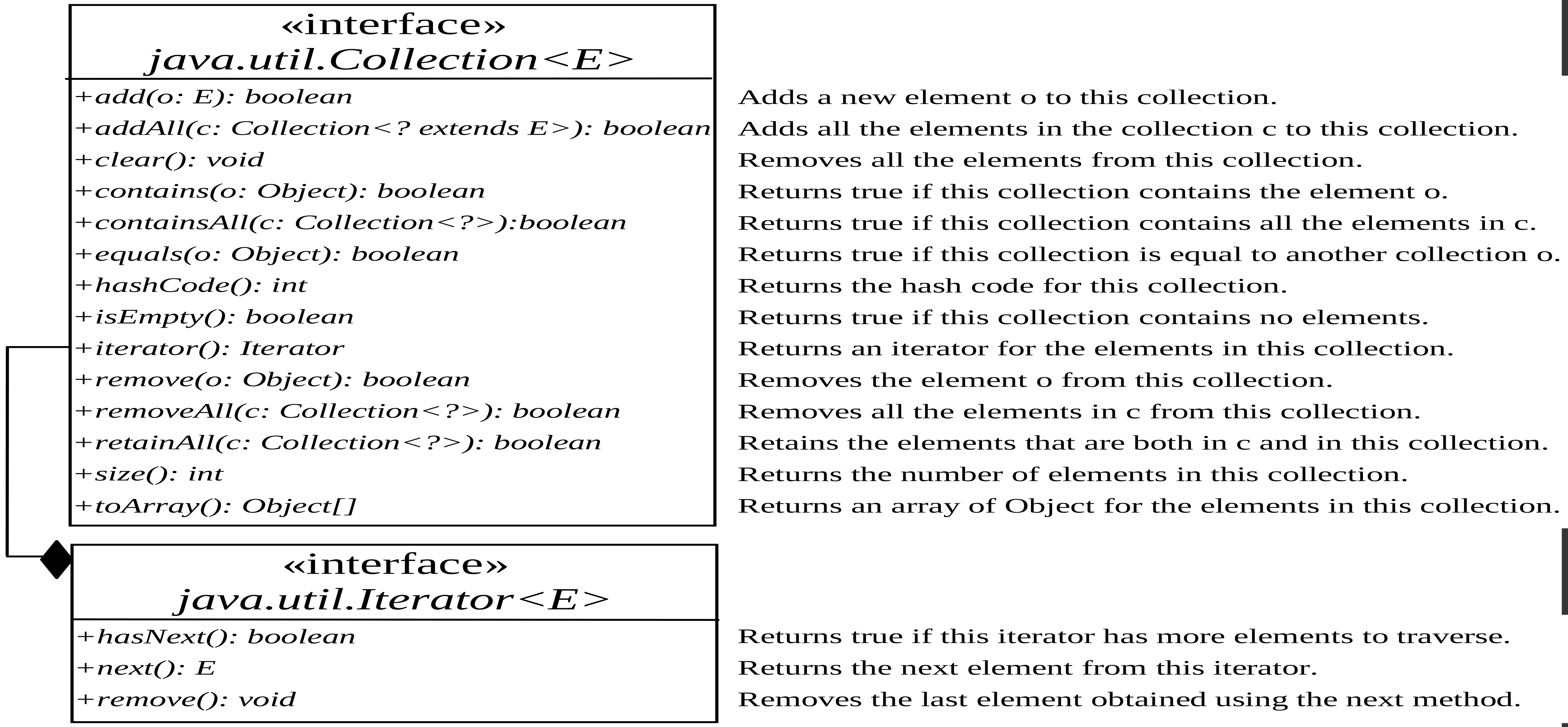
```
public interface Collection<E> extends Iterable<E>{


     // any collection object to be used in foreach
 looks

}
```

- `<E>` tells you that the interface is generic.
- You must specify the type of object when you instantiate the collection.
- Iterable is from java.lang package with one method **iterator().**

Collection Interface



«interface»
java.util.Collection<E>

+add(o: E): boolean	Adds a new element o to this collection.
+addAll(c: Collection<? extends E>): boolean	Adds all the elements in the collection c to this collection.
+clear(): void	Removes all the elements from this collection.
+contains(o: Object): boolean	Returns true if this collection contains the element o.
+containsAll(c: Collection<?>):boolean	Returns true if this collection contains all the elements in c.
+equals(o: Object): boolean	Returns true if this collection is equal to another collection o.
+hashCode(): int	Returns the hash code for this collection.
+isEmpty(): boolean	Returns true if this collection contains no elements.
+iterator(): Iterator	Returns an iterator for the elements in this collection.
+remove(o: Object): boolean	Removes the element o from this collection.
+removeAll(c: Collection<?>): boolean	Removes all the elements in c from this collection.
+retainAll(c: Collection<?>): boolean	Retains the elements that are both in c and in this collection.
+size(): int	Returns the number of elements in this collection.
+toArray(): Object[]	Returns an array of Object for the elements in this collection.

«interface»
java.util.Iterator<E>

+hasNext(): boolean	Returns true if this iterator has more elements to traverse.
+next(): E	Returns the next element from this iterator.
+remove(): void	Removes the last element obtained using the next method.

Collection interface classification

- Basic operations
- Bulk operations
- Array operations

Basic operations

```
public interface Collection<E> extends Iterable<E>{
      boolean add(E element); //optional
      boolean remove(Object element); //optional
      boolean contains(Object element);
      int size(); // size of collection
      boolean isEmpty(); //collection empty/ not
      Iterator<E> iterator();
}
```

Bulk operations

```
public interface Collection<E> extends Iterable<E>{

 boolean addAll(Collection<? Extends E>c); //optional

 boolean removeAll(Collection<? Extends E>c); //optional

 boolean containsAll(Collection<?>c);

 boolean retainAll(Collection<?>c); //optional

 void clear(); //optional

}
```

Array operations

```
public interface Collection<E> extends Iterable<E>{
 Object[] toArray();
 <T> T[] toArray(T[] a);
 //e.g., String[] a = c.toArray(new String[0]);
}
```

Both Types has to be same

- Both methods return an array containing all elements in the collections.

Implementation Classes

- Collections in Java provides core implementation classes for collections.
- Use them to create different types of collections in java program.
- ArrayList, LinkedList, HashMap, TreeMap, HashSet, TreeSet.
- Extend them to create your own custom collection class.

Algorithms

- Algorithms are useful methods to provide some common functionalities, for example searching, sorting and shuffling.

Collection Example (Basic)

```
import java.util.*;

public class TestCollection {
  public static void main(String[] args) {
    ArrayList<String> collection1 = new ArrayList<>();
    collection1.add("New York");
    collection1.add("Atlanta");
    collection1.add("Dallas");
    collection1.add("Madison");

    System.out.println("A list of cities in collection1:");
    System.out.println(collection1);

    System.out.println("\nIs Dallas in collection1? "
      + collection1.contains("Dallas"));
```

```
A list of cities in collection1:
[New York, Atlanta, Dallas, Madison]
```

```
Is Dallas in collection1? true
```

```
collection1.remove("Dallas");
    System.out.println("\n" + collection1.size() +
       " cities are in collection1 now");

    Collection<String> collection2 = new ArrayList<>();
    collection2.add("Seattle");
    collection2.add("Portland");
    collection2.add("Los Angles");
    collection2.add("Atlanta");

    System.out.println("\nA list of cities in collection2:");
    System.out.println(collection2);

    ArrayList<String> c1 = (ArrayList<String>)(collection1.clone());
    c1.addAll(collection2);
    System.out.println("\nCities in collection1 or collection2: ");
    System.out.println(c1);

    c1 = (ArrayList<String>)(collection1.clone());
    c1.retainAll(collection2);
    System.out.print("\nCities in collection1 and collection2: ");
    System.out.println(c1);

    c1 = (ArrayList<String>)(collection1.clone());
    c1.removeAll(collection2);
    System.out.print("\nCities in collection1, but not in 2: ");
    System.out.println(c1);
  }
```

```
3 cities are in collection1 now
```

```
A list of cities in collection2:
[Seattle, Portland, Los Angeles, Atlanta]
```

```
Cities in collection1 or collection2:
[New York, Atlanta, Madison, Seattle, Portland, Los Angeles, Atlanta]
```

```
Cities in collection1 and collection2: [Atlanta]
```

```
Cities in collection1, but not in 2: [New York, Madison]
```