Collections in Java

- **Collections in java** is a framework that provides an 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).

Collections in Java

What is Collection in java:

- Collection represents a single unit of objects i.e. a group.

What is Collection framework:

- Collection framework represents a unified architecture for storing and manipulating group of objects. It has:
  - Interfaces and its implementations i.e. classes
  - Algorithm

Iterator interface

Iterator interface

- Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

- There are only three methods in the Iterator interface. They are:
- **public boolean hasNext()** It returns true if iterator has more elements.
- **public Object next()** It returns the element and moves the cursor pointer to the next element.
- **public void remove()** It removes the last elements returned by the iterator.

Java ArrayList class

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

Methods of Java ArrayList

- void add(int index, Object element)
- boolean addAll(Collection c)
- void clear()

ArrayList<String> list=**new** ArrayList<String>();//creating new generic arraylist

Java ArrayList class

```
import java.util.*;
class TestCollection1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add(“Imran");//Adding object in arraylist
  list.add(“Ahsan");
  list.add(“Ayesha");
  list.add(“Ali");
  //Traversing list through Iterator
  Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }  } }
```

Two ways to iterate the elements of collection in java

There are two ways to traverse collection elements:

- By Iterator interface.
- By for-each loop.
- In the above example, we have seen traversing ArrayList by Iterator. Let's see the example to traverse ArrayList elements using for-each loop.

Two ways to iterate the elements of collection in java

```
import java.util.*;
class TestCollection2{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
  list.add(“Imran");//Adding object in arraylist
  list.add(“Ahsan");
  list.add(“Ayesha");
  list.add(“Ali");
  for(String obj:list)
    System.out.println(obj);
 }
}
```

Example of addAll(Collection c) method

```
import java.util.*;
class TestCollection4{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
  list.add(“Imran");//Adding object in arraylist
  list.add(“Ahsan");
  ArrayList<String> list2=new ArrayList<String>();
  list2.add(“Ayesha");
  list2.add(“Ali");
  list.addAll(list2);//adding second list in first list
  Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  } } }
```

Example of removeAll() method

```
import java.util.*;
class TestCollection5{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
   list.add(“Imran");//Adding object in arraylist
  list.add(“Ahsan");
  ArrayList<String> list2=new ArrayList<String>();
  list2.add(“Ayesha");
  list2.add(“Ali");
  list.removeAll(list2);
  System.out.println("iterating the elements after removing the elements of list2...");
  Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }  } }
```

Example of Sorting the elements of List that contains string objects

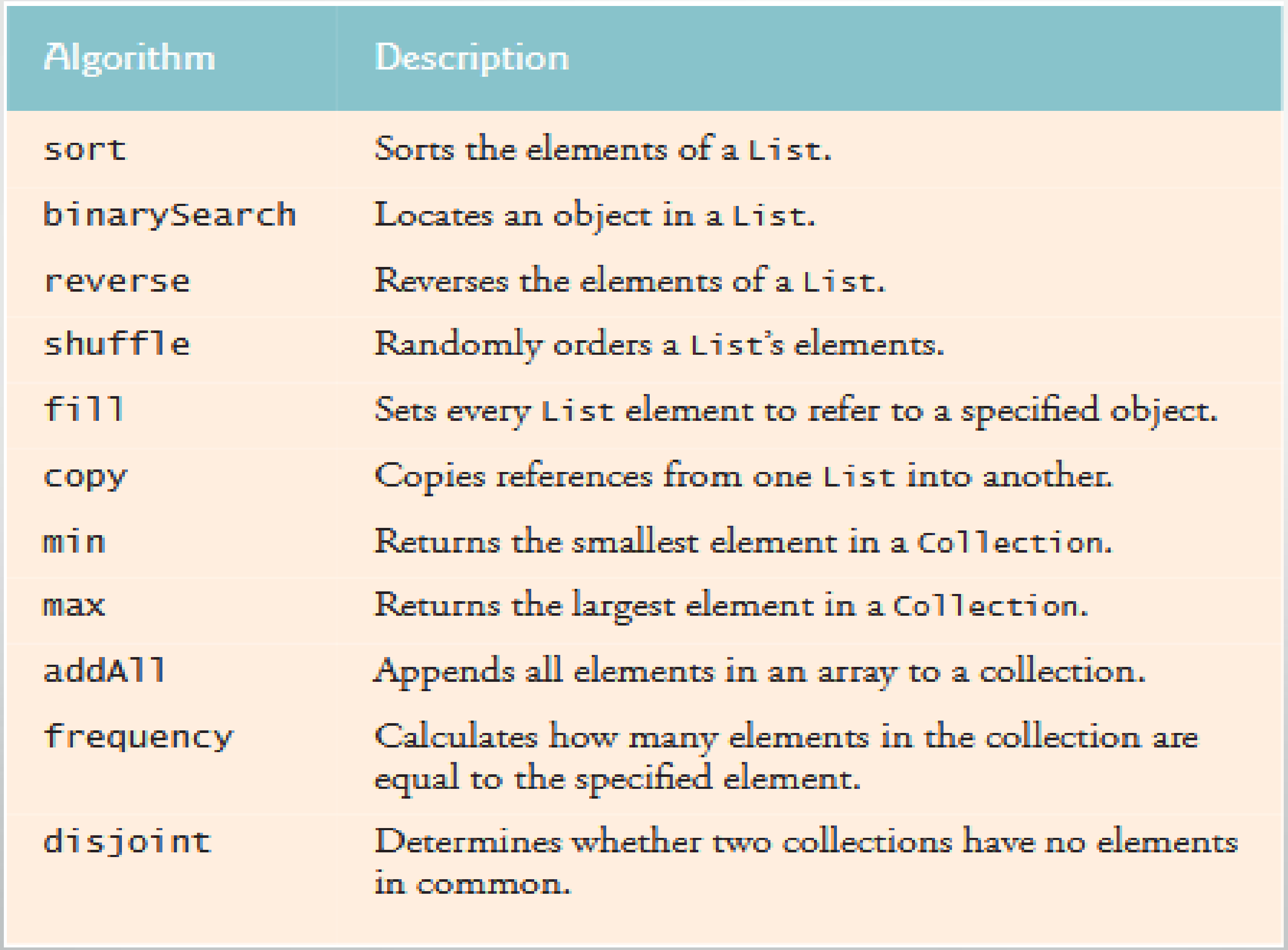
- **import** java.util.*;
- **class** TestSort1{
- **public static void** main(String args[]){
- ArrayList<String> list=**new** ArrayList<String>();
- list.add(“Imran");//Adding object in arraylist
- list.add(“Ahsan");
- list.add(“Ayesha");
- list.add(“Ali");
- Collections.sort(list);
- **//Collections**.sort(list, **Collections**.reverseOrder());
- Iterator itr=list.iterator();
- **while**(itr.hasNext()){

  System.out.println(itr.next());
- } } }

Collections Algorithms

- The collections framework provides several high-performance algorithms for manipulating collection elements.
- Algorithms sort , binarySearch , reverse , shuffle , fill and copy operate on Lists. Algorithms min , max , addAll, frequency and disjoint operate on Collections.

Collections Algorithms



Algorithm	Description
`sort`	Sorts the elements of a `List`.
`binarySearch`	Locates an object in a `List`.
`reverse`	Reverses the elements of a `List`.
`shuffle`	Randomly orders a `List`'s elements.
`fill`	Sets every `List` element to refer to a specified object.
`copy`	Copies references from one `List` into another.
`min`	Returns the smallest element in a `Collection`.
`max`	Returns the largest element in a `Collection`.
`addAll`	Appends all elements in an array to a collection.
`frequency`	Calculates how many elements in the collection are equal to the specified element.
`disjoint`	Determines whether two collections have no elements in common.

Algorithm sort

- Algorithm sort sorts the elements of a List
  - Sorting in Ascending Order
  - Sorting in Descending Order

Example of Sorting in Ascending Order

```java
import java.util.*;
class TestCollection5{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
  System.out.println("Unsorted array elements:");
  list.add("Hearts");
  list.add("Diamonds");
  list.add("Clubs");
  list.add("Spades");
  System.out.println(list);
  System.out.printf("Sorted array elements:");
  Collections.sort(list);
  System.out.println(list);
  } }
```

Example of Sorting in Descending Order

```
import java.util.*;
class TestCollection5{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
  System.out.println("Unsorted array elements:");
  list.add("Hearts");
  list.add("Diamonds");
  list.add("Clubs");
  list.add("Spades");
  System.out.println(list);
  System.out.printf("Sorted array elements in reverse Order:");
 Collections.sort( list, Collections.reverseOrder() ); //using sorting algorithm
 Collections.reverse(list); //using Collection algorithm
  System.out.println(list);
   } }
```

Algorithm shuffle

- Algorithm shuffle randomly orders a List’s elements.

Example of shuffle

- **import** java.util.*;
- **class** TestCollection5{
- **public static void** main(String args[]){
- ArrayList<String> list=**new** ArrayList<String>();
- System.out.println("Unsorted array elements:");
- list.add("Hearts");
- list.add("Diamonds");
- list.add("Clubs");
- list.add("Spades");
- System.out.println(list);
- System.out.printf("Shuffle array elements:");
- Collections.shuffle(list);
- System.out.println(list);

} }

Algorithm fill

- Algorithm fill overwrites elements in a List with a specified value.
- The fill operation is useful for reinitializing a List.

Example of fill

- **import** java.util.*;
- **class** TestCollection5{
- **public static void** main(String args[]){
- ArrayList<String> list=**new** ArrayList<String>();
- System.out.println("Unsorted array elements:");
- list.add("Hearts");
- list.add("Diamonds");
- list.add("Clubs");
- list.add("Spades");
- System.out.println(list);
- System.out.printf("Fill array elements with some value:");
- Collections.fill(list,"test");
- System.out.println(list);

} }

Algorithm copy

- Algorithm copy takes two arguments—a destination List and a source List. Each source List element is copied to the destination List.

Example of copy

```
import java.util.*;
class TestCollection5{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
  System.out.println("Unsorted array elements:");
  list.add("Hearts");
  list.add("Diamonds");
  list.add("Clubs");
  list.add("Spades");
  System.out.println(list);
  System.out.printf(“Copy list into list1:");
  ArrayList< String > list1 = new ArrayList<String>();
  list1.add("H");
  list1.add("D");
  list1.add("C");
  list1.add("S");
  Collections.copy( list1,list);  System.out.println(list);
   } }
```

Min and max

- Algorithms min and max each operate on any Collection. Algorithm min returns the smallest element in a Collection , and algorithm max returns the largest element in a Collection.

Example of Min and max

- **import** java.util.*;
- **class** TestCollection5{
- **public static void** main(String args[]){
- ArrayList< String > list1 = new ArrayList<String>();
- list1.add("H");
- list1.add("D");
- list1.add("C");
- list1.add("S");
- //Collections.copy(list1,list);
- System.out.println(Collections.min(list1));
- System.out.println(Collections.max(list1));
- } }

frequency

- Algorithm frequency takes two arguments—a Collection to be searched and an Object to be searched for in the collection. Method frequency returns the number of times that the second argument appears in the collection.

Example of frequency

- **import** java.util.*;
- **class** TestCollection5{
- **public static void** main(String args[]){
- ArrayList< String > list1 = new ArrayList<String>();
- list1.add("H");
- list1.add("D");
- list1.add("H");
- list1.add("S");
- int frequency = Collections.frequency(list1,“H");
- System.out.println(frequency);
- } }

disjoint

- Algorithm disjoint takes two Collections and returns true if they have no elements in common.

Example of disjoint

- **import** java.util.*;
- **class** TestCollection5{
- **public static void** main(String args[]){
- ArrayList<String> list=**new** ArrayList<String>();
- System.out.println("Unsorted array elements:");
- list.add("Hearts");
- list.add("Diamonds");
- list.add("Clubs");
- list.add("Spades");
- System.out.println(list);
- ArrayList< String > list1 = new ArrayList<String>();
- list1.add("H");
- list1.add("D");
- list1.add("C");
- list1.add("S");
- boolean disjoint = Collections.disjoint(list, list1);
- } }

Sorting in Collection

We can sort the elements of:

- String objects
- Wrapper class objects
- User-defined class objects

Example of Sorting the elements of List that contains Wrapper class objects

- **import** java.util.*;
- **class** TestSort2{
- **public static void** main(String args[]){
- ArrayList al=**new** ArrayList();
- al.add(201);
- al.add(101);
- al.add(230);//internally will be converted into objects as Integer.valueOf(230)
- al.add(“Imran”);
- Iterator itr=al.iterator();
- **while**(itr.hasNext()){
- System.out.println(itr.next());

} } }

User-defined class objects in Java ArrayList

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

```
import java.util.*;
public class TestCollection3{
 public static void main(String args[]){
  Student s1=new Student(101,“Imran",23);
  Student s2=new Student(102,“Ahsan",21);
  Student s3=new Student(103,“Ayesha",25);
  ArrayList<Student> list=new ArrayList<Student>();
  list.add(s1);//adding Student class object
  list.add(s2);
  list.add(s3);
  Iterator itr=list.iterator();    //Getting Iterator
  while(itr.hasNext()){
   Student st=(Student)itr.next();
   System.out.println(st.rollno+" "+st.name+" "+st.age);
  }  }  }
```

Java ArrayList Example: Book

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String aut
hor, String publisher, int quantity) {
   this.id = id;
   this.name = name;
   this.author = author;
   this.publisher = publisher;
   this.quantity = quantity;
} }
```

```
public class ArrayListExample {
public static void main(String[] args) {
 ArrayList<Book> list=new ArrayList<Book>();
  Book b1=new Book(101,"Let us C","Yashwant Kanetkar
","BPB",8);
   Book b2=new Book(102,"Data Communications & Net
working","Forouzan","Mc Graw Hill",4);
   Book b3=new Book(103,"Operating System","Galvin","
Wiley",6);
   list.add(b1);
   list.add(b2);
   list.add(b3);
   for(Book b:list){
      System.out.println(b.id+" "+b.name+" "+b.author+" "
+b.publisher+" "+b.quantity);
   } } }
```

End.