Collections

ArrayList

- It is a collection (from Collection interface)
- Implements List Interface
- Resizable dynamically
- Type safe using Generics (diamond notation < >)
- java.util package

ArrayList - Main Methods

- add(Object o)
- add(int index, Object o)
- remove(Object o)
- remove(int index)
- Object get(int index)
- int indexOf(Object o)
- int size()
- boolean contains(Object o)
- clear()

Comparable Interface

- Class whose objects to be sorted must implement this interface. In this, we have to implement compareTo(Object) method.

- Implements:
  - int compareTo(T o)
  - Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Comparator Interface

- Class whose objects to be sorted do not need to implement this interface. Some third class can implement this interface to sort. e.g. StudentSortByNameComparator class can implement Comparator interface to sort collection of student object by name.

- Implements:
  - int compare(T o1, T o2)
  - Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

Comparable vs. Comparator

Parameter	Comparable	Comparator
Sorting logic	Sorting logic must be in same class whose objects are being sorted. Hence this is called natural ordering of objects	Sorting logic is in separate class. Hence we can write different sorting based on different attributes of objects to be sorted. E.g. Sorting using id, name etc.
Implementation	Class whose objects to be sorted must implement this interface. E.g. Student class needs to implement comparable to collection of student object by name	Class whose objects to be sorted do not need to implement this interface. Some other class can implement this interface. E.g.-StudentSortByNameComparator class can implement Comparator interface to sort collection of country object by id
Sorting method	int compareTo(Object o1) This method compares this object with o1 object and returns a integer. Its value has following meaning 1. positive – this object is greater than o1 2. zero – this object equals to o1 3. negative – this object is less than o1	int compare(Object o1,Object o2) This method compares o1 and o2 objects. and returns a integer. Its value has following meaning. 1. positive – o1 is greater than o2 2. zero – o1 equals to o2 3. negative – o1 is less than o1
Calling method	Collections.sort(List) Here objects will be sorted on the basis of CompareTo method	Collections.sort(List, Comparator) Here objects will be sorted on the basis of Compare method in Comparator
Package	Java.lang.Comparable	Java.util.Comparator

Stacks

- Last-in First-out (LIFO)
- The elements are accessed only from the top the the stack
- Main methods:
  - boolean empty() – Tests if the stack is empty.
  - Object peek() – Looks at the object at the top without removing it from the stack.
  - Object pop() – Removes the object at the top of this stack and returns the object.
  - Object push(E item) – Pushes an item onto the top of this stack.
  - int search(Object o) – Returns the 1-based position where an object is on this stack.

HashMap

- Very fast lookups
- Key-value pairs, in which the keys have meaning
- No “order” preservation as we have in ArrayLists or arrays
- null key and null values allowed
- No duplicate keys allowed
- Imagine an ArrayList of 1,000,000 students and a HashMap of 1,000,000 students. If you have a studentNumber, you can get direct access to your student in a HashMap whereas ArrayList would require searching.

HashMap

- There are three main implementations of Map interfaces:
  - HashMap - It makes no guarantees concerning the order of iteration
  - TreeMap - It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.
  - LinkedHashMap - It orders its elements based on the order in which they were inserted into the set (insertion-order)

HashMap Methods

- entrySet() returns a Set mapping of all keys and values
- keySet() returns a Set of keys
- containsKey(Object key)
- containsValue(Object value)
- get(Object key) returns a value when you give it a key
- put(Key k, Value v) a key => value pair into the Map
- remove(Object key)
- size()
- values() returns a Collection of the values

Sets

- No duplicate values permitted
- There are three main implementations of Set interface:
  - HashSet - Stores its elements in a hash table, is the best-performing implementation
  - TreeSet - Stores its elements in a red-black tree, orders its elements based on their values
  - LinkedHashSet – It is implemented as a hash table with a linked list running through it, orders its elements based on the order in which they were inserted into the set (insertion-order)