Core Java

Day 12 Agenda

- Q & A
- Java collection framework
  - List implementations
  - Fail-fast and Fail-safe Iterators
  - Collections
  - Queue implementations
  - Set basics

Q & A

- Natural Ordering
  - Comparision of "current" class object to another class object on certain fields.
  - Natural Ordering is implemented within the class.
  - Standard for implementation = java.lang.Comparable interface
  - Example:

```
class Point implements Comparable<Point> {
    private int x, y;
    // ...
    public int compareTo(Point other) {
        // calculate diff between "this" and "other" point
        return diff;
    }
}
```

```
Arrays.sort(arr);
```

- Comparator
  - External/third-party comparision of two objects of a class.
  - Example:

```
class BookComparator implements Comparator<Book> {
    public int compare(Book b1, Book b2) {
        int diff = b1.getQuantity() - b2.getQuantity();
        return diff;
    }
}
```

```
// "arr" is Book[]
Arrays.sort(arr, new BookComparator());
```

```
// "list" is List<Book>
list.sort(new BookComparator());
```

Java Collection Framework

List interface

- Ordered/sequential collection.
- Implementations: ArrayList, Vector, Stack, LinkedList, etc.
- List can contain duplicate elements.
- List can contain multiple null elements.

- Elements can be accessed sequentially (bi-directional using Iterator) or randomly (index based).
- Abstract methods
    - void add(int index, E element)
    - String toString()
    - E get(int index)
    - E set(int index, E element)
    - int indexOf(Object o)
    - int lastIndexOf(Object o)
    - E remove(int index)
    - boolean addAll(int index, Collection<? extends E> c)
    - ListIterator listIterator()
    - ListIterator listIterator(int index)
    - List subList(int fromIndex, int toIndex)
- To store objects of user-defined types in the list, you must override equals() method for the objects. It is mandetory while searching operations like contains(), indexOf(), lastIndexOf().

Iterator vs Enumeration

- Enumeration
    - Since Java 1.0
    - Methods
        - boolean hasMoreElements()
        - E nextElement()
    - Example

```
Enumeration<E> e = v.elements();
while(e.hasMoreElements()) {
    E ele = e.nextElement();
    System.out.println(ele);
}
```

    - Enumeration behaves similar to fail-safe iterator.
- Iterator
    - Part of collection framework (1.2)
    - Methods
        - boolean hasNext()
        - E next()
        - void remove()
    - Example

```
Iterator<E> e = v.iterator();
while(e.hasNext()) {
    E ele = e.next();
    System.out.println(ele);
}
```

- ListIterator
    - Part of collection framework (1.2)
    - Inherited from Iterator
    - Bi-directional access
    - Methods
        - boolean hasNext()
        - E next()
        - int nextIndex()
        - boolean hasPrevious()
        - E previous()
        - int previousIndex()
        - void remove()
        - void set(E e)
        - void add(E e)

ArrayList class

- Internally ArraysList is dynamically growable array.
- Elements can be traversed using Iterator, ListIterator, or using index.
- Default initial capacity of ArrayList is 10. If it gets filled then its capacity gets increased by half of its existing capacity.
- Primary use
    - Random access is very fast
    - Add/remove at the end of list
- Internals (for experts)
    - https://www.javatpoint.com/internal-working-of-arraylist-in-java

Vector class

- Legacy collection class (since Java 1.0), modified for collection framework (List interface).
- Internally Vector is dynamically growable array.
- Elements can be traversed using Enumeration, Iterator, ListIterator, or using index.
- Default initial capacity of vector is 10. If it gets filled then its capacity gets increased/ by its existing capacity.
- Synchronized collection -- Thread safe but slower performance
- Primary use
    - Random access (in multi-threaded applications)
    - Add/remove at the end of list (in multi-threaded applications)

```
NOTE:
* To perform multiple tasks concurrently within a single process, threads are used (thread based multi-tasking or
multi-threading).
* When multiple threads are accessing same resource at the same time, the race condition may occur. Due to this
undesirable/unexpected results will be produced.
* To avoid this, OS/JVM provides synchronization mechanism. It will provide thread-safe access to the resource
(the other threads will be blocked).
```

Traversal

- Using Iterator

```
Iterator<Integer> itr = list.iterator();
while(itr.hasNext()) {
    Integer i = itr.next();
    System.out.println(i);
}
```

- Using for-each loop

```
for(Integer i:list)
    System.out.println(i);
```

  - Gets converted into Iterator traversal

```
for(Iterator<Integer> itr = list.iterator();itr.hasNext();) {
    Integer i = itr.next();
    System.out.println(i);
}
```

- Traversing List collection

```
for(int i=0; i<list.size(); i++) {
    Integer n = list.get(i);
    System.out.println(n);
}
```

  - Faster for ArrayList/Vector (than Iterator).
  - Much slower for LinkedList.

- Enumeration -- Traversing Vector (Java 1.0)

```
// v is Vector<Integer>
Enumeration<Integer> e = v.elements();
while(e.hasMoreElements()) {
    Integer i = e.nextElement();
    System.out.println(i);
}
```

Fail-fast vs Fail-safe Iterator

- If state of collection is modified (add/remove operation other than iterator methods) while traversing a collection using iterator and iterator methods fails (with ConcurrentModificationException), then iterator is said to be Fail-fast.
  - e.g. Iterators from ArrayList, LinkedList, Vector, ...
- If iterator allows to modify the underlying collection (add/remove operation other than iterator methods) while traversing a collection (NO ConcurrentModificationException), then iterator is said to be Fail-safe.
  - e.g. Iterators from CopyOnWriteArrayList, ...

Synchronized vs Unsynchronized collections

- Synchronized collections are thread-safe and sync checks cause slower execution.
- Legacy collections were synchronized.
  - Vector
  - Stack
  - Hashtable
  - Properties
- Collection classes in collection framework (since 1.2) are non-synchronized (for better performance).
- Collection classes can be converted to synchronized collection using Collections class methods.
  - syncList = Collections.synchronizedList(list)
  - syncSet = Collections.synchronizedSet(set)
  - syncMap = Collections.synchronizedMap(map)

Collections class

- Helper/utility class that provides several static helper methods
- Methods
  - List reverse(List list);
  - List shuffle(List list);
  - void sort(List list, Comparator cmp)
  - E max(Collection list, Comparator cmp);
  - E min(Collection list, Comparator cmp);
  - List synchronizedList(List list);

Collection vs Collections

- Collection interface
  - All methods are public and abstract. They implemented in sub-classes.
  - Since all methods are non-static, must be called on object.

```
Collection<Integer> list = new ArrayList<>();
//List<Integer> list = new ArrayList<>();
//ArrayList<Integer> list = new ArrayList<>();
list.remove(new Integer(12));
```

- Collections class
  - Helper class that contains all static methods.
  - We never create object of "Collections" class.

```
Collections.methodName(...);
```

LinkedList class

- Internally LinkedList is doubly linked list.
- Elements can be traversed using Iterator, ListIterator, or using index.
- Primary use
    - Add/remove elements (anywhere)
    - Less contiguous memory available
- Inherited from List<>, Deque<>.

Queue interface

- Represents utility data structures (like Stack, Queue, ...) data structure.
- Implementations: LinkedList, ArrayDeque, PriorityQueue.
- Can be accessed using iterator, but no random access.
- Methods
    - boolean add(E e) - throw IllegalStateException if full.
    - E remove() - throw NoSuchElementException if empty
    - E element() - throw NoSuchElementException if empty
    - boolean offer(E e) - return false if full.
    - E poll() - returns null if empty
    - E peek() - returns null if empty
- In queue, addition and deletion is done from the different ends (rear and front).

Deque interface

- Represents double ended queue data structure i.e. add/delete can be done from both the ends.
- Two sets of methods
    - Throwing exception on failure: addFirst(), addLast(), removeFirst(), removeLast(), getFirst(), getLast().
    - Returning special value on failure: offerFirst(), offerLast(), pollFirst(), pollLast(), peekFirst(), peekLast().
- Can used as Queue as well as Stack.
- Methods
    - boolean offerFirst(E e)
    - E pollFirst()
    - E peekFirst()

    - boolean offerLast(E e)
    - E pollLast()
    - E peekLast()

ArrayDeque class

- Internally ArrayDeque is dynamically growable array.
- Elements are allocated contiguously in memory.

LinkedList class

- Internally LinkedList is doubly linked list.

PriorityQueue class

- Internally PriorityQueue is a "binary heap" data structure.
- Elements with highest priority is deleted first (NOT FIFO).
- Elements should have natural ordering or need to provide comparator.

Set interface

- Collection of unique elements (NO duplicates allowed).
- Implementations: HashSet, LinkedHashSet, TreeSet.
- Elements can be accessed using an Iterator.
- Abstract methods (same as Collection interface)
    - add() returns false if element is duplicate

HashSet class

- Non-ordered set (elements stored in any order)
- Elements must implement equals() and hashCode()
- Fast execution

LinkedHashSet class

- Ordered set (preserves order of insertion)
- Elements must implement equals() and hashCode()
- Slower than HashSet

TreeSet class

- Sorted navigable set (stores elements in sorted order)
- Elements must implement Comparable or provide Comparator
- Slower than HashSet and LinkedHashSet

Assignment

1. Store book details in a library in a list -- ArrayList.
    - Book details: isbn(string), price(double), authorName(string), quantity(int)
    - Write a menu driven program to
        1. Add new book in List
            - If book not present, then add a new book (hint - contains())
            - If book is present, increment its quantity.
        2. Display all books in forward order using random access
        3. Search a book with given isbn (hint - indexOf())
        4. Delete a book at given index.
        5. Delete a book with given isbn.
        6. Reverse the list (hint - Collections.reverseList())
2. Create a list of strings. Find the string with highest length using Collections.max().
3. Create LinkedList<> of Employee. Perform add, delete, find, sort, edit functionality in a menu driven program. Refer hint below for edit/update functionality:

```
System.out.println("Enter emp id to be modified: ");
int id = sc.nextInt();
Employee key = new Employee();
```

```
key.setId(id);
int index = list.indexOf(key);
if(index == -1)
    System.out.println("Employee not found.");
else {
    Employee oldEmp = list.get(index);
    System.out.println("Employee Found: " + oldEmp);
    System.out.println("Enter new information for the Employee");
    Employee newEmp = new Employee();
    newEmp.accept();
    list.set(index, newEmp);
}
```

4. In which collection classes null is not allowed? Duplicate null is not allowed? Multiple nulls are allowed?

```
//Collection<String> c = new ArrayList<>();
//Collection<String> c = new HashSet<>();
//Collection<String> c = new LinkedHashSet<>();
//Collection<String> c = new TreeSet<>();
c.add("B");
c.add("D");
c.add("A");
c.add("C");
c.add(null);
c.add(null);
c.add(null);
System.out.println(c.toString());
```