Core Java

Day13 Agenda

- Lists

Java Collection Framework

List interface

- Ordered/sequential collection.
- List can contain duplicate elements.
- List can contain multiple null elements.
- Elements can be accessed sequentially (bi-directional using Iterator) or randomly (index based).
- List enable searching (in the list)
- Implementations: ArrayList, Vector, Stack, LinkedList, etc.
- Abstract methods
    - void add(int index, E element)
    - E get(int index)
    - E set(int index, E element)
    - E remove(int index)
    - boolean addAll(int index, Collection<? extends E> c)
    - int indexOf(Object o)
    - int lastIndexOf(Object o)
    - String toString()
    - 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 conatins(), 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)

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);
}
```

- 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);
}
```

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
- Limitations:
    - Slower random access
- Inherited from List<>, Deque<>.

ArrayList class

- Internally ArrayList is dynamic array (can grow or shrink dynamically).
- When ArrayList capacity is full, it grows by half of its size.
- Elements can be traversed using Iterator, ListIterator, or using index.
- Primary use
    - Random access
    - Add/remove elements (at the end)
- Limitations
    - Slower add/remove in between the collection
    - Uses more contiguous memory
- Inherited from List<>.

Vector class

- Internally Vector is dynamic array (can grow or shrink dynamically).
- Vector is a legacy collection (since Java 1.0) that is modified to fit List interface.
- Vector is synchronized (thread-safe) and hence slower.
- When Vector capacity is full, it doubles its size.

- Elements can be traversed using Enumeration, Iterator, ListIterator, or using index.
- Primary use
    - Random access
    - Add/remove elements (at the end)
- Limitations
    - Slower add/remove in between the collection
    - Uses more contiguous memory
    - Synchronization slow down performance in single threaded environment
- Inherited from List<>.

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(...);
```