Core Java

Agenda

- Q & A
- Lists
- Queues

Q & A

1. Type erasure
    - Java generics provides type-safety by type-checking done by the compiler.
    - In JVM, all generic type "T" are treated as "Object" references.
    - When compiler compiles Java code all generic type information `<...>` is erased by the compiler.
    - Example:

```
void printBox(Box<Integer> b) {
    // ...
}
void printBox(Box<String> b) {
    // ...
}
```

        - When compiler compiles code, it will remove `<Integer>` and `<String>` and both methods will have exactly same signature, because of which overloading is not possible.
2. Natural ordering
    - How object of a class is compared with another object of that class?

SUNBEAM INFOTECH

- This comparison logic written inside the class in a standard way -- java.lang.Comparable -- compareTo() -- compares current (this) object with other (argument) object and returns difference.
- Natural ordering is always implemented within the class -- by the developer who implemented the class -- Generally it should be consistent with equals().

```
class Point implements Comparable<Point> {
    private int x, y;
    // ...
    public int compareTo(Point other) {
        double dx = this.x - other.x;
        double dy = this.y - other.y;
        return Double.compare(dx * dx, dy * dy);
    }
}
```

```
Point[] arr = new Point[] { ... };
Arrays.sort(arr);
```

3. Comparable vs Comparator

- Comparator is standard of comparing two given objects of the class -- int compare(T a, T b);

```
class PointComparator implements Comparator<Point> {
    public int compare(Point a, Point b) {
        double dx = a.getX() - b.getX();
        double dy = a.getY() - b.getY();
        return - Double.compare(dx * dx, dy * dy);
    }
}
```

SUNBEAM INFOTECH

```
Point[] arr = new Point[] { ... };
Arrays.sort(arr, new PointComparator());
```

- Differences:
  - Comparable standard is implemented within class by the developer of the class; while Comparator standard is implemented outside the class as per requirement.
  - Comparable is standard of comparing "this" object with "other" object; while Comparator is standard of comparing two given objects.
  - java.lang.Comparable vs java.util.Comparator
  - Comparable: int compareTo(T other); Comparator: int compare(T first, T second);

4. Iterable vs Iterator

- interface java.lang.Iterable -- standard for creating an iterator object.

```
interface Iterable {
    Iterator iterator();
        // create object of Iterator (i.e. object of a class inherited from Iterator interface) and
return it.
}
```

- interface java.util.Iterator -- standard for iterating/traversing through collection.

```
interface Iterator {
    boolean hasNext();
        // check if element is present at current position
    T next();
        // returns eleemnt at current position and points to next position
}
```

SUNBEAM INFOTECH

- The iterators are implemented by the collection classes (inside that collection class).

5. for-each loop vs Iterator

```
for(String ele : collection) {
    System.out.println(ele);
}
```

- Internally converted to following code by Java compiler.

```
itr = collection.iterator();
while(itr.hasNext()) {
    ele = itr.next();
    System.out.println(ele);
}
```

- for-each loop can be used only on the classes inherited from Iterable. Otherwise, it will give compiler error.

6. contains() vs equals()

```
class ArrayList ... {
    // ...
    public boolean contains(T key) {
        for(T ele : values) {
            if(key.equals(value))
                return true;
        }
    }
    return false;
}
```

SUNBEAM INFOTECH

7. ConcurrentModificationException
    - Fail-fast iterator vs Fail-safe iterator -- See you tomorrow.

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().

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.

SUNBEAM INFOTECH

- 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).
```

Iterator vs Enumeration

- Enumeration
  - Since Java 1.0
  - Methods
    - boolean hasMoreElements()

SUNBEAM INFOTECH

        - 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()

SUNBEAM INFOTECH

    - 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);
}
```

SUNBEAM INFOTECH

- 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);
}
```

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.

SUNBEAM INFOTECH

- 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(...);
```

SUNBEAM INFOTECH

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

SUNBEAM INFOTECH

- 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.

References

- Linked List and Iterator implementation in Java

SUNBEAM INFOTECH

```java
public class SinglyLinkedList<T> implements Iterable<T> {
    private static class Node<T> {...}

    private Node<T> head;

    public SinglyLinkedList() {...}

    public void add(T val) {...}

    public java.util.Iterator<T> iterator() {
        return new ListItr();
    }

    private class ListItr implements Iterator<T> {
        private Node<T> cur;
        public ListItr() {
            cur = head;
        }
        @Override
        public boolean hasNext() {
            return cur != null;
        }
        @Override
        public T next() {
            T val = cur.data;
            cur = cur.next;
            return val;
        }
    }
}
```

```java
public class Program {
    public static void main(String[] args) {
        SinglyLinkedList<String> list;
        list = new SinglyLinkedList<String>();

        list.add("Bill");
        list.add("Steve");
        list.add("Mark");
        list.add("Elon");

        System.out.println("Traversal using Iterator: ");
        Iterator<String> itr = list.iterator();
        while(itr.hasNext()) {
            String ele = itr.next();
            System.out.println(ele);
        }

        System.out.println("\nTraversal using for-each: ");
        for (String ele : list)
            System.out.println(ele);
    }
}
```

NILESH GHULE

- https://www.linkedin.com/posts/nilesh-g_java-datastructures-linkedlist-activity-7125699934757998593-X2Jn

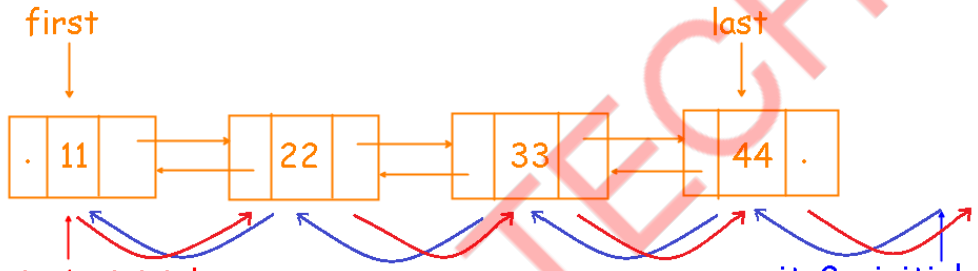
- Forward and Backward Traversal using ListIterator

SUNBEAM INFOTECH

```
List<Integer> list = new LinkedList<>();
list.add(11);
list.add(22);
list.add(33);
list.add(44);

ListIterator<Integer> itr1;
itr1 = list.listIterator();
while(itr1.hasNext()) {
  Integer ele = itr1.next();
  // print ele
}

ListIterator<Integer> itr2;
itr2=list.listIterator(list.size());
while(itr2.hasPrevious()) {
  Integer ele = itr2.previous();
  // print ele
}
```



itr1 - initial position.

itr2 - initial position.

hasNext() checks if an element present at current position

hasPrevious() checks if an element is present at previous position

next() returns element at current position and then start pointing to the next element.

NILESH GHULE

previous() start pointing to the previous element and then returns element at that position.

Assignments

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 - indexOf())
        - If book is present, sum its quantity i.e. new quantity = existing quantity + input quantity
     2. Display all books in forward order using random access
     3. Search a book with given isbn (hint - indexOf())

SUNBEAM INFOTECH

4. Delete a book at given index.
5. Delete a book with given isbn.
6. Delete a book with given name.
7. Sort books by isbn in asc order -- Collections.sort(list);
8. Sort books by price in desc order -- Collections.sort(list, comparator);
9. Reverse the list -- Collections.reverse(list);

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. Create PriorityQueue<> of Employee. Add employees in the queue and ensure that employees are deleted in desc order of their salaries.

SUNBEAM INFOTECH