Agenda

- Generic Interfaces
  - Comparable
  - Comparator
- Arrays class
- Collection FrameWork
- List
- Extra Points
- ~~Set~~
- ~~Map~~

Generic Interfaces

- Interface is standard/specification.
- comparable is a predefined interface in java

```
// Comparable is pre-defined interface which was non-generic till Java 1.4

interface Comparable {
int compareTo(Object obj);
}

class Person implements Comparable {
// ...
public int compareTo(Object obj) {
Person other = (Person)obj; // down-casting
// compare "this" with "other" and return difference
}
}

class Program {
  public static void main(String[] args) {
    Person p1 = new Person("James Bond", 50);
    Person p2 = new Person("Ironman", 45);
    int diff = p1.compareTo(p2);
    if(diff == 0)
    System.out.println("Both are same");
    else if(diff > 0)
    System.out.println("p1 is greater than p2");
    else //if(diff < 0)
    System.out.println("p1 is less than p2");
    diff = p2.compareTo("Superman"); // will fail at runtime with
ClassCastException (in down-casting)
}
}
```

- Generic interface has type-safe methods (arguments and/or return-type).

```
// Comparable is pre-defined interface -- generic since Java 5.0
interface Comparable<T> {
int compareTo(T obj);
}

class Person implements Comparable<Person> {
// ...
public int compareTo(Person other) {
// compare "this" with "other" and return difference
}
}

class Program {
  public static void main(String[] args) {
    Person p1 = new Person("James Bond", 50);
    Person p2 = new Person("Ironman", 45);
    int diff = p1.compareTo(p2);
    if(diff == 0)
    System.out.println("Both are same");
    else if(diff > 0)
    System.out.println("p1 is greater than p2");
    else //if(diff < 0)
    System.out.println("p1 is less than p2");
    diff = p2.compareTo("Superman"); // compiler error
}
}
```

Comparable<>

- Standard for comparing the current object to the other object.
- Has single abstract method int compareTo(T other);
- In java.lang package.
- Used by various methods like Arrays.sort(Object[]), ...
- It does the comparision for the natural ordering

Comparator<>

- Standard for comparing two (other) objects.
- Has single abstract method int compare(T obj1, T obj2);
- In java.util package.
- Used by various methods like Arrays.sort(T[], comparator), ...

Arrays

- it is a class is java.util package
- cosists of helper methods for working on Array.
- eg
    - search
    - sort

    - equals
    - toString
- The array must be sorted (as by the sort() method) prior to making the search call.
- If it is not sorted, the results are undefined.

Collection Framework

- Collection framework is Library of reusable data structure classes that is used to develop application.
- Main purpose of collection framework is to manage data/objects in RAM efficiently.
- Collection framework was introduced in Java 1.2 and type-safe implementation is provided in 5.0 (using generics).
- Collection is available in java.util package.
- Java collection framework provides

1. Interfaces -- defines standard methods for the collections.
2. Implementations -- classes that implements various data stuctures.
3. Algorithms -- helper methods like searching, sorting, ...

Collection Hierarchy

- Interfaces: Iterable, Collection, List, Queue, Set, Map, Deque, SortedSet, SortedMap, ...
- Implementations: ArrayList, LinkedList, HashSet, HashMap, ...
- Algorithms: sort(), reverse(), max(), min(), ... -> in Collections class static methods

Collection interface

- Root interface in collection framework interface hierarchy.
- Most of collection classes are inherited from this interface (indirectly).
- Provides most basic/general functionality for any collection
- Abstract methods
    - boolean add(E e)
    - int size()
    - boolean isEmpty()
    - void clear()
    - boolean contains(Object o)
    - boolean remove(Object o)
    - boolean addAll(Collection<? extends E> c)
    - boolean containsAll(Collection<?> c)
    - boolean removeAll(Collection<?> c)
    - boolean retainAll(Collection<?> c)
    - Object[] toArray()
    - Iterator iterator() -- inherited from Iterable
- Default methods
    - default Stream stream()
    - default Stream parallelStream()
    - default boolean removeIf(Predicate<? super E> filter)

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).
- List enables searching in the list
- 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().

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<>.

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(Demo05-> Program02)
    - Random access
    - Add/remove elements (at the end)
- Limitations
    - Slower add/remove in between the collection

    - Uses more contiguous memory
- Inherited from List<>.

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<>.

Stack

- It is inherited from vector class.
- Generally used to have only the stack operations like push, pop and peek opertaions.
- It is recommended to use the Dequeu from the queue collection.
- It is synchronized and hence gives low performanance.

Extra Points

Iterable interface

- To traverse any collection it provides an Iterator.
- Enable use of for-each loop.
- In java.lang package
- Iterable yeilds an iterator
- Methods
    - Iterator iterator()
    - default Spliterator spliterator()
    - default void forEach(Consumer<? super T> action)

Iterator

- Part of collection framework (1.2)
- Methods
    - boolean hasNext()
    - E next()
    - void remove()

Enumeration

Since Java 1.0

- Methods
    - boolean hasMoreElements()
    - E nextElement() (Demo02)

Traversal

1. Using Iterator

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

2. Using for-each loop

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

// gets converted into Iterator traversal internally

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

3. using for loop

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

4. 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);
  }
```

- Enumeration behaves similar to fail-safe iterator.
- Since Java 1.0
- Methods
  - boolean hasMoreElements()
  - E nextElement()
- only useful when traversing with vector

Fail-fast vs Fail-safe Iterator

- If state of collection is modified (add/remove operation other than iterator methods) i.e structural change while traversing a collection using iterator and iterator methods fails (with ConcurrentModificationException), then iterator is said to be Fail-fast.
- The collections from java.util package have fail-fast iterators
- 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.
- The collections from java.util.concurrent package have fail-safe iterators.
- e.g. Iterators from CopyOnWriteArrayList, ...
- If any changes are done in the collection using these iterators then the changes may not be reflected using the same iterator however by creating the new iterator we can get the changes displayed.

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).
- Difference between these methods is first 3 methods throws exception however next 3 methods do not throw exception if operation fails.

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.
- Time Complexity to add and remove is O(1)

LinkedList class

- Internally LinkedList is doubly linked list.
- Time Complexity to add and remove is O(1)

PriorityQueue class

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

Assigenment

- Create a menu driven code which has below menus
- Create an array of Student of size 5.
- Add the dummy records of student in any random order
- create the below menus

1. Add Student
2. Display All Students
3. Sort students in natural ordering (use roll no for it)
4. Sort students in desc order of the name
5. Sort students in desc order of the marks