Core Java

Day 13 Agenda

- Q & A
- Java collection framework
    - Set implementations
    - Hashtable concepts
    - hashCode() and equals()
    - Map implementations

Java collection framework

Set interface

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

HashSet class

- Non-ordered set (elements stored in any order -- based on hashCode())
- Elements must implement equals() and hashCode()
- Fast execution
- Only one null value is allowed. Additional null values are not added in set.

LinkedHashSet class

- Ordered set (preserves order of insertion)

- Elements must implement equals() and hashCode()
- Slower than HashSet
- Only one null value is allowed. Additional null values are not added in set.

SortedSet interface

- Use natural ordering or Comparator to keep elements in sorted order
- Methods
  - E first()
  - E last()
  - SortedSet headSet(E toElement)
  - SortedSet subSet(E fromElement, E toElement)
  - SortedSet tailSet(E fromElement)

NavigableSet interface

- Sorted set with additional methods for navigation
- Methods
  - E higher(E e)
  - E lower(E e)
  - E pollFirst()
  - E pollLast()
  - NavigableSet descendingSet()
  - Iterator descendingIterator()

TreeSet class

- Sorted navigable set (stores elements in sorted order)
- Elements must implement Comparable or provide Comparator
- Slower than HashSet and LinkedHashSet
- null values are not allowed i.e. cause NullPointerException.

- It is recommended to have consistent implementation for Comparable (Natural ordering) and equals() method i.e. equality and comparison should done on same fields.
- If need to sort on other fields, use Comparator.

```
class Book implememts Comparable<Book> {
    private String isbn;
    private String name;
    // ...
    public int hashCode() {
        return isbn.hashCode();
    }
    public boolean equals(Object obj) {
        if(!(obj instanceof Book))
            return false;
        Book other=(Book)obj;
        if(this.isbn.equals(other.isbn))
            return true;
        return false;
    }
    public int compareTo(Book other) {
        return this.isbn.compareTo(other.isbn);
    }
}
```

```
// Store in sorted order by name
set = new TreeSet<Book>((b1,b2) -> b1.getName().compareTo(b2.getName()));
```

```
// Store in sorted order by isbn (Natural ordering)
set = new TreeSet<Book>();
```

HashTable Data structure

- Hashtable stores data in key-value pairs so that for the given key, value can be searched in fastest possible time.
- Internally hash-table is a table(array), in which each slot(index) has a bucket(collection). Key-value entries are stored in the buckets depending on hash code of the "key".
- Load factor = Number of entries / Number of buckets.
- Examples
    - Key=Name, Value=Phone number
    - Key=pincode, Value=city/area
        - 400027 -- Byculla, Mumbai
        - 411046 -- Katraj, Pune
        - 411052 -- Hinjawadi, Pune
        - 415110 -- Karad, Satara
        - 411037 -- Marketyard, Pune
        - 411002 -- Bajirao Rd, Pune
        - 411007 -- Aundh, Pune
    - Key=Employee, Value=Manager
    - Key=Department, Value=list of Employees

hashCode() method

- Object class has hashCode() method, that returns a unique number for each object (by converting its address into a number).
- To use any hash-based data structure hashCode() and equals() method must be implemented.
- If two distinct objects yield same hashCode(), it is referred as collision. More collisions reduce performance.
- Most common technique is to multiply field values with prime numbers to get uniform distribution and lesser collsions.
- hashCode() overriding rules
    - hash code should be calculated on the fields that decides equality of the object.
    - hashCode() should return same hash code each time unless object state is modified.
    - If two objects are equal (by equals()), then their hash code must be same.
    - If two objects are not equal (by equals()), then their hash code may be same (but reduce performance).

Map interface

- Collection of key-value entries (Duplicate "keys" not allowed).
- Implementations: HashMap, LinkedHashMap, TreeMap, Hashtable, ...
- The data can be accessed as set of keys, collection of values, and/or set of key-value entries.
- Map.Entry<K,V> is nested interface of Map<K,V>.
    - K getKey()
    - V getValue()
    - V setValue(V value)
- Abstract methods

```
* boolean isEmpty()
* int size()
* V put(K key, V value)
* V get(Object key)
* Set<K> keySet()
* Collection<V> values()
* Set<Map.Entry<K,V>> entrySet()
* boolean containsValue(Object value)
* boolean containsKey(Object key)
* V remove(Object key)
* void clear()
* void putAll(Map<? extends K,? extends V> map)
```

- Maps not considered as true collection, because it is not inherited from Collection interface.

HashMap class

- Non-ordered map (entries stored in any order -- as per hash code of key)
- Keys must implement equals() and hashCode()
- Fast execution
- Mostly used Map implementation

LinkedHashMap class

- Ordered map (preserves order of insertion)
- Keys must implement equals() and hashCode()
- Slower than HashSet
- Since Java 1.4

TreeMap class

- Sorted navigable map (stores entries in sorted order of key)
- Keys must implement Comparable or provide Comparator
- Slower than HashMap and LinkedHashMap
- Internally based on Red-Black tree.
- Doesn't allow null key (allows null value though).

Hashtable class

- Similar to HashMap class.
- Legacy collection class (since Java 1.0), modified for collection framework (Map interface).
- Synchronized collection -- Thread safe but slower performance
- Inherited from java.util.Dictionary abstract class (it is Obsolete).

Example

```Java
class Distance {
    private int feet, inches;
    // ...
    public int hashCode() {
        int hash = Objects.hash(this.feet, this.inches);
        return hash;
    }
    public boolean equals(Object obj) {
        if(obj == null)
            return false;
```

```
        if(this == obj)
            return true;
        if(obj instanceof Distance) {
           if(Objects.equals(this.feet, other.feet) && Objects.equals(this.inches, other.inches))
                return true;
        }
        return false;
    }
    /*
    public int hashCode() {
        int hash = 31 * this.feet + this.inches;
        return hash;
    }
    public boolean equals(Object obj) {
        if(obj == null)
            return false;
        if(this == obj)
            return true;
        if(obj instanceof Distance) {
            Distance other = (Distance)obj;
            if(this.feet == other.feet && this.inches == other.inches)
                return true;
        }
        return false;
    }
    */
}
class Person {
    private String name;
    private int age;
    private Distance height;
    // ...
}
```
```Java
class Main {
```

```
    public static void main(String[] args) {
        Map<Distance,Person> map = new HashMap<>();
        Person p1 = new Person("Nilesh", 40, new Distance(5,7));
        map.put(p1.getHeight(), p1);
        Person p2 = new Person("Rahul", 43, new Distance(5,6));
        map.put(p2.getHeight(), p2);
        // ...
        Person p = new Person();
        p.accept();
        map.put(p.getHeight(), p);
    }
}
```

Assignments

1. Store few books (hardcoded values with the Book class in previous assignment) in a HashSet and display them using iterator. If any book with duplicate isbn is added, what will happen? Books are stored in which order? Solve duplicate ISBN problem.
2. In above assignment use LinkedHashSet instead of HashSet. If any book with duplicate isbn is added, what will happen? Books are stored in which order?
3. In above assignment use TreeSet instead of LinkedHashSet. Use natural ordering for the Book. If any book with duplicate isbn is added, what will happen? Books are stored in which order?
4. Use TreeSet to store all books in descending order of price. Natural ordering for the Book should be isbn (do not change it). Display them using iterator() and descendingIterator().
5. Store Books in HashMap<> so that for given isbn, book can be searched in fastest possible time. Do we need to write equals() and hashCode() in Book class? Hint:

```
// declare map: key=isbn, value=Book object
Map<String,Book> map = new HashMap<>();

// case 1: insert in map
Book b = new Book();
// accept book from user
map.put(b.getIsbn(), b);
```

```
// case 2: find in map
String isbn = sc.next();
Book f = map.get(isbn);
```

6. Store Students in LinkedHashMap<> so that, for given roll, Student can be searched in fastest possible time. Do we need to write equals() and hashCode() in Student class? Follow menu-driven approach. Hint:

```
class Student {
    // ...
}
```

```
Map<Integer, Student> map = new HashMap<>();
```

```
s = new Student();
acceptStudent(s); // implement method in Main class
map.put(s.getRoll(), s);
```

```
roll = sc.nextInt();
s = map.get(roll);
```