Agenda

- Hashtable
- Map
- hashcode()
- String Tokenizer
- Garbage Collector
- Date/Calender/LocalDate
- ~~Java 8 Interfaces~~
- ~~Enum~~

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).
- Load factor = Number of entries / Number of buckets.
- Multiple keys can compete for the same slot which can cause the collision
- To avoid the collision two techniques are used

1. Open Adderessing
2. Seperate Chaining

- In Seperate Chaning mechanism to avoid the collision Key-value entries are stored in the same bucket depending on hash code of the "key".
- In java we have readymade/ built-in hashtables

1. HashMap
2. LinkedHashMap
3. TreeMap
4. HashTable (Legacy)
5. Properties (Legacy)

- Here we neeed to calculate the hash value of the key using hash function(Override hashcode method).
- The slot in the table is calculated internaly by slot = key.hashcode()%size
- Examples
    - Key=pincode, Value=city/area
    - 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).

Similarity between Set and Map

- Set in internally using map implementation where it have all the values as null.
- In set the the elements are stored as keys and the corresponsing values are null.
- HashSet = HashMap<K,null>
- LinkedHashSet = LinkedHashMap<K,null>
- TreeSet = TreeMap<K,null>
- in set duplicate elements are not allowed, in map duplicate keys are not allowed
- For HashSet,Hashmap, LinkedHashSet, LinkedHashMap duplication is based on equals() and hashcode() of key
- For TreeSet and TreeMap the duplication is based on comparable of K or Comparator of K given in constructor

String Tokenizer

- Used to break a string into multiple tokens - like split() method.
- Methods of java.util.StringTokenizer
  - boolean hasMoreTokens()
  - String nextToken()
  - String nextToken(String delim)

Garbage Collector

- Garbage collection is automatic memory management by JVM.
- If a Java object is unreachable (i.e. not accessible through any reference), then it is automatically released by the garbage collector.
- An object become eligible for GC in one of the following cases:

```
// 1. Nullify the reference.
MyClass obj = new MyClass();
```

```
obj = null;

//2. Reassign the reference.
MyClass obj = new MyClass();
obj = new MyClass();

//3. Object created locally in method.
void method() {
MyClass obj = new MyClass();
// ...
}

- GC is a background thread in JVM that runs periodically and reclaim memory of
unreferenced objects.
- Before object is destroyed, its finalize() method is invoked (if present).
- One should override this method if object holds any resource to be released
explicitly e.g. file close, database connection, etc.

```JAVA
class Test {
    Scanner sc = new Scanner(System.in);

    @Override
    protected void finalize() throws Throwable {
        sc.close();
    }
}

public class Program{

    public static void main(String[] args) {
        Test t1 = new Test();
        t1 = null;
        System.gc();// request GC
    }
}
```


- GC can be requested (not forced) by one of the following.
    1. System.gc();
    2. Runtime.getRuntime().gc();
- GC is of two types i.e. Minor and Major.
    1. Minor GC: Unreferenced objects from young generation are reclaimed. Objects not reclaimed here are moved to old/permanent generation.
    2. Major GC: Unreferenced objects from all generations are reclaimed. This is unefficient (slower process).
- JVM GC internally use Mark and Compact algorithm.
```

- GC Internals: https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html