**Priority Queue – String iteration**

Facts we know:

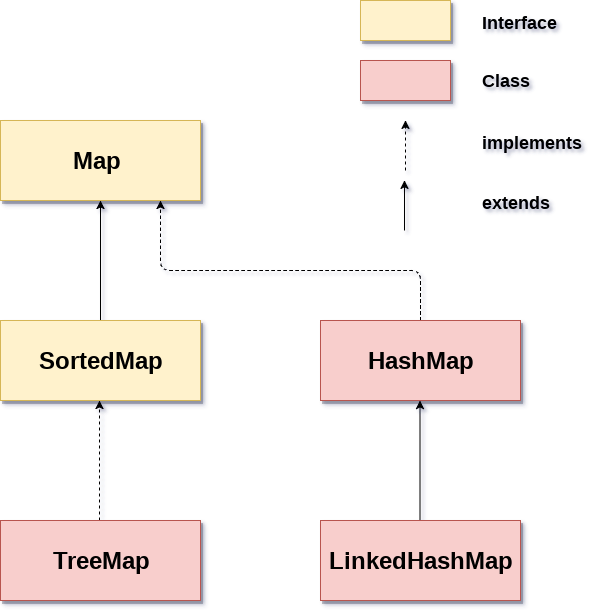
* Queue → FIFO
* Processed according to their priority
* Comparator provided at queue construction time, depending on which constructor is used. Ex: an element with a maximum ASCII value will have the highest priority.
* Add() → The elements are stored based on the priority order which is ascending by default.

Note: We will not get sorted elements by printing PriorityQueue. Iterator for the priority queue does not iterate in any order. And it uses Iterator of the extending colllection - in your case of the PriorityQueue - to travers the collection and create a String from it. And if you check the docs of [PriorityQueue::iterator](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/PriorityQueue.html#iterator()) it returns iterator that returns elements in no particular order . If you want to retrieve elements in their priority order defined by comparator use methods like poll and remove.

**Map Collections**

Facts:

* Value-key pairs
* Useful for search, update, delete.
* 2 interfaces in Java:
  + Map
  + Sorted Map
* 3 classes:
  + HashMap:
    - can have null keys
    - Implementation of Map.
    - Doesn’t hold any order.
    - Uses a technique called hashing. → Converts a large string to a smaller string that represents the same string. Faster search and indexing.
  + LinkedHashMap:
    - can have null keys
    - Extends HashMap
    - Maintains insertion order.
  + TreeMap:
    - cannot have null keys
    - Implementation of Map and SortedMap
    - Holds ascending order
* No duplicate keys but can have duplicate values.
* Cannot be traversed → need to convert it to a set to traverse it.



When to use?

* Dictionaries → key value pair
* Map for error codes and their descriptions.
* Zip codes and their cities.
* Classes and students.

**Hashcode() and equals()**

Facts:

* Methods of object class so every java class gets default
* We can override these methods.
* Important for hash tables.
  + Equals:
    - The default implementation of *equals()* in the *Object* class says that equality is the same as object identity. Ex: Money a = new Money(50); != Money b = new Money(50);
    - To get an accurate result we need to override the method.
    - Requirements:
      * a=a → true
      * x=y → true iff y=x → true
      * If x=y and y=z then x=z
      * x=y always if properties are not changed
    - Inheritance can violate the symmetry property. So instead of inheriting the class, we can have it’s property in the class.
  + Hashcode:
    - Returns the integer hashcode of the object.
    - We should calculate this integer consistent with the equality method. → if we override equals we also need to override the hashcode method.
    - Contract to override:
      * **Internal consistency:** the value of *hashCode()* may only change if a property that is in *equals()* changes.
      * **External consistency**: objects that are equal to each other must return the same hashCode.
      * **Collision:** Different objects may have the same hashcode.
    - General rules for equals and hashcode:
      * If x=y, then x.hashcode() = y.hashcode()
      * If x.hashcode() = y.hashcode(), x doesn't have to be equal to y
    - Writing these methods by hand is complicated so we can let IDE generate them for us.

**HashTables and HashMaps**

Facts:

* Both key-value pairs
* Key is hashed → hashcode
* No insertion order in either of them

Difference between:

| Hash Tables | Hash Maps |
| --- | --- |
| No null values and keys | One null key and multiple null values |
| slow | fast |
| traversed by Enumerator and Iterator. | traversed by Iterator. |
| Synchronized | Not Synchronized |
| – | Generally preferred (For single thread apps) |
| Thread safe | Not thread safe |
|  |  |