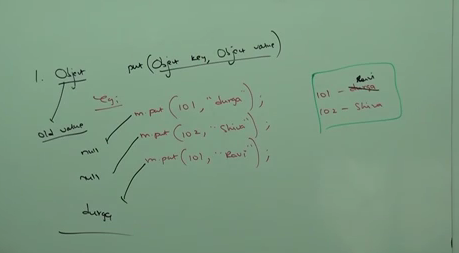
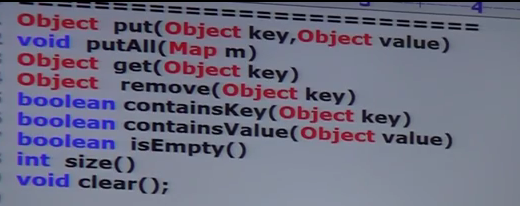


**Map (I):**

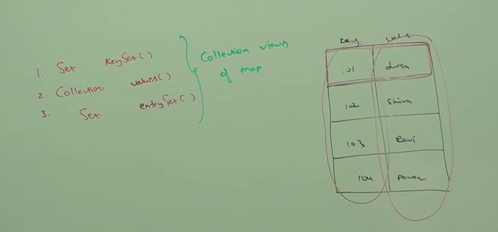
1. Collections talk about a group of individual objects.
2. **Map represent a group of objects as key value pairs.**
3. It is not child interface of Collection interface.
4. Both Collection and Map are different.
5. Each key value pair is called one Entry. So a group of entries is called a map.
6. Duplicate keys are not allowed but values can be duplicated.

**Methods:**

****

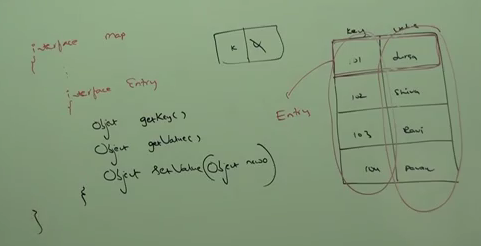
****

**Collection Views of Map : Viewing the map with the help of Collections**

****

**Entry (I):**

Map is considered as a collection of entry objects. Without existing map object, there is no chance of entry object. Hence, entry interface is defined inside map interface.



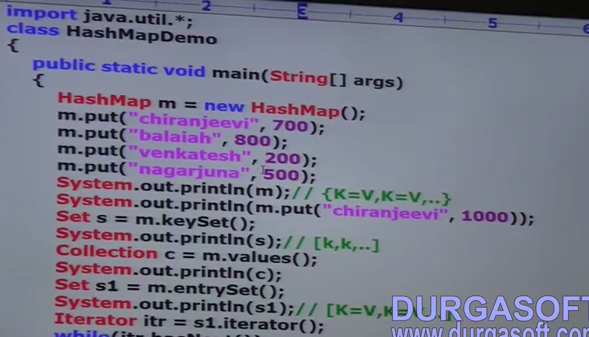
**Hashmap:**

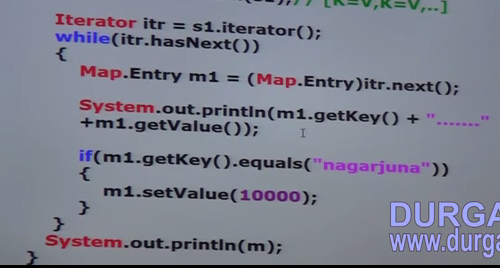
1. The underlying data structure is hash table.
2. Insertion order is not preserved and it based on of hashing. (Hashing is based on keys only not values).
3. Duplicated keys are not allowed but values can be duplicated.
4. Heterogeneous objects are allowed for both key and value.
5. Null key is allowed only once and null value is allowed any no of times
6. Implements serializable, cloneable interfaces but not randomaccess
7. **Best choice if our frequent operation is search**

**Constructors:**

1. Hashmap m=new Hashmap(); -> creates empty hashmap of intital capacity 16 and default fill ratio of 0.75
2. Hashmap m =new Hashmap(int initialcapacity)
3. Hashmap m =new Hashmap(int initialcapacity, float fillratio)
4. Hashmap m =new Hashmap(Map m) -> this is the only constructor differ from Hashset. Hashset takes collection but map is not part of collection interface, it takes map as argument here.

Example;





|  |  |
| --- | --- |
| **Hashmap** | **Hashtable** |
| Mehods in hashmap are not synchronized | Mehods in hashmap are synchronized |
| Multiple threads are allowed to operate, so not thread safe | Multiple threads are not allowed to operate, so It is thread safe |
| High performance | Relatively low performance |
| Null is allowed for key and value | Null is not allowed for both key and value. We get Null pointer exception |
| Not legacy. 1.2v | Legacy. 1.0v |

**To convert hashmap into synchronized version,**

Map m =new Hashmap(); Map m = Collections.synchronizedMap(m);

**LinkedHashmap:**

|  |  |
| --- | --- |
| Hashmap | LinkedHashmap |
| It is implementation class of Map interface | It is child class of hashmap. It is exactly same as hashmap including constructors and methods except the below differences |
| Underlying datastructure is hashtable | Underlying datastructure is a combination of linkedlist and hashtable (hybrid datastructure) |
| Insertion order is not preserved and it is based on hash code of keys | Insertion order is preserved |
| 1.2v | 1.4v |

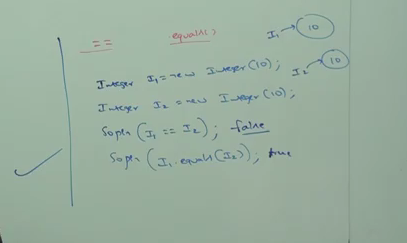
**Just change Hashmap into LinkedHashmap in above example.**

* LinkedHashmap is commonly used for cache based applications

**IdentityHashMap:**

**Diff b/w == operator and equals() on objects:**

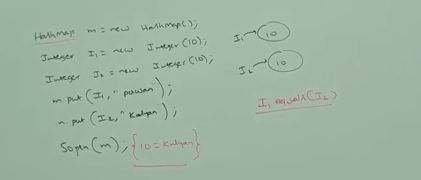
1. == operator is used for reference or address level comparison on objects
2. Equals() is for content based comparison on objects

****

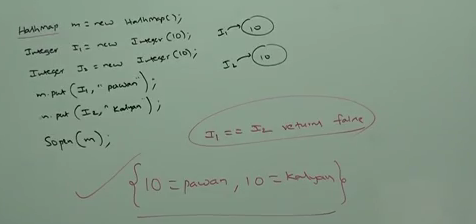
It is exactly same as Hashmap including methods and constructors, except the following differences.

1. In the case of normal hashmap, JVM will use .equals() to identify the duplicate keys, which is meant for content comparision.
2. But in the case of IdentityHashmap, JVM will use ==operator to identify the duplicate keys which is meant for reference(address) comparision.

Example:



Replace hashmap as Identity hashmap in above example,

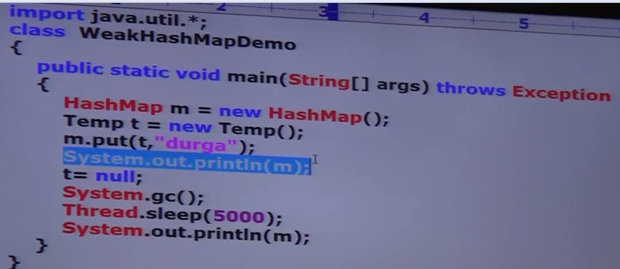


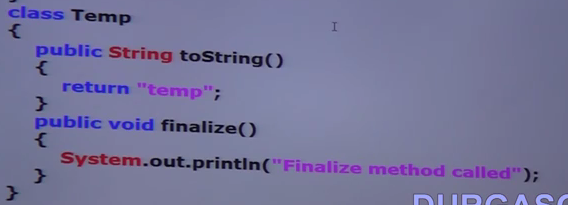
**WeakHashmap:**

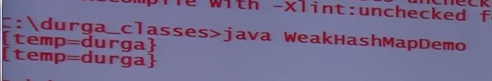
It is exactly same as Hashmap including methods and constructors, except the following differences.

1. In the case of normal hashmap, eventhough object doesn’t have any reference, it is not eligible for GC if it is associated with hashmap. That is hashmap dominates the garbage collector.
2. But in the case of WeakHashmap, if object doesn’t contain any references, it is eligible or GC eventhough the object is associated with weakhashmap. That is garbage collector dominates weakhashmap.

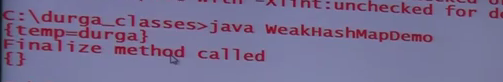
**Example:**

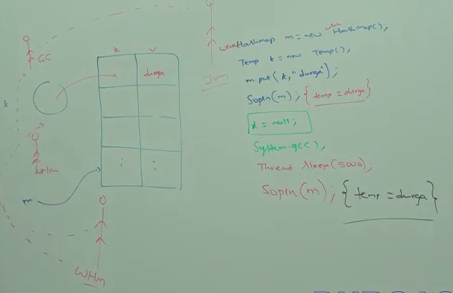
****

****

****

**If we change hashmap into weakhashmap, then the output is:**

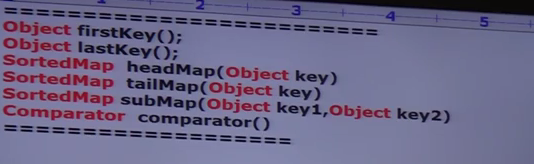
****

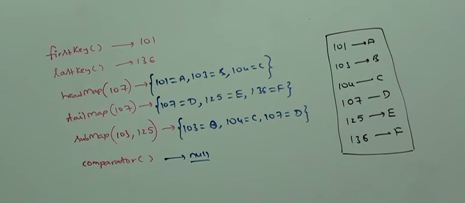
****

**SortedMap (I):**

1. It is the child interface of Map interface
2. If we want to represent a group of key value pairs according to some sorting order of keys, then we should go for SortedMap.
3. Sorting is based on key only but not based on value.

**Methods:**

****

****

**NavigableMap:** - See later in 1.6 enhancement

**TreeMap:**

1. Underlying datastructure is RED-BLACK Tree
2. Insertion order is not preserved and it is based on some sorting order of keys.
3. Duplicate keys are not allowed but values can be duplicated
4. If we are depending on default natural sorting order, then keys should be homogeneous and comparable. Otherwise we will get runtime exception saying classcast exception.
5. If we are defining our own sorting by comparator, then, keys need not be homogeneous and comparable. We can take heterogeneous and non comparable objects also.
6. Whether we are depending on default natural sorting order or customized sorting order, there are no restrictions for values. We can take heterogeneous and non comparable objects also for values.

**Null Acceptance:**

1. For non empty treemap, if we are trying to insert a entry with null key then, we will get runtime exception saying null pointer exception.
2. For empty tree map, as the first entry with null key is allowed. But after inserting that entry, if we are trying to insert any other entry, then we will get runtime exception saying null pointer exception.

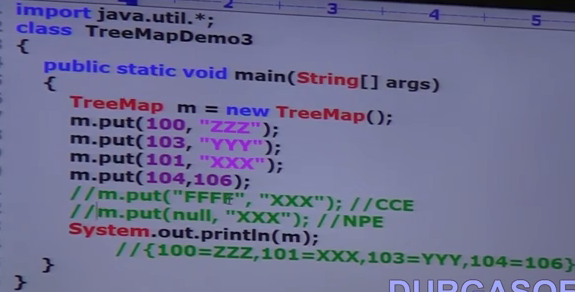
**Note.** The above null acceptance rule is applicable until 1.6 version only. From 1.7 onwards, null is not allowed for key. No restrictions on null values and we can use any no of times.

**Constructors:**

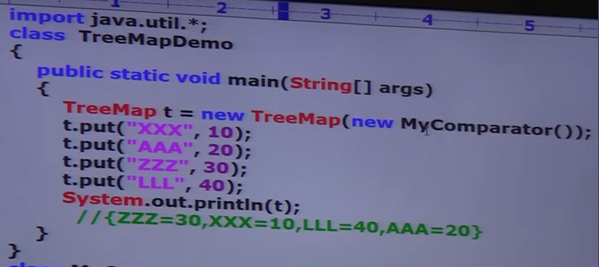
1. **Treemap m=new Treemap() -> create a map with default natural sorting order of keys**
2. **Treemap m=new Treemap(Comparator c) -> for customized sorting of keys**
3. **Treemap m=new Treemap(Map m)**
4. **Treemap m=new Treemap(SortedMap map)**

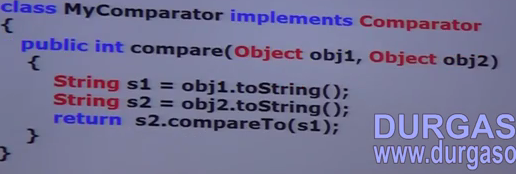
**Example:**

**//Default Natural Sorting order pgm:**

****

**//Customized Sorting order pgm for predefined class (String):**

****

****

**Legacy classes:**

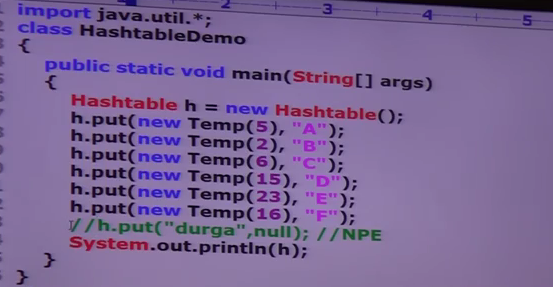
**Hashtable:**

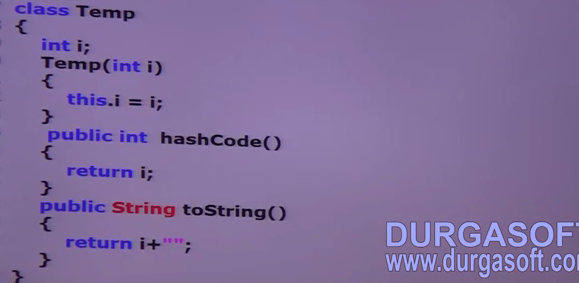
1. Underlying data structure is hashtable only
2. Insertion order is not preserved and it is based on hashcode of keys
3. Duplicate keys are not allowed but values can be duplicated
4. Heterogeneous objects are allowed for both key & value
5. Null is not applicable for both key and value
6. Implements Serializable, cloneable but not random access
7. Thread safe – all methods are synchronized
8. Good for search based operation

**Constructors:**

1. **Hashtable h=new Hashtable() -> creates empty hashtable with a initial capacity of 11 and fill ratio of 0.75**
2. **Hashtable h=new Hashtable(int initialcapacity)**
3. **Hashtable h=new Hashtable(int initialcapacity, float fillratio)**
4. **Hashtable h=new Hashtable(Map m)**

**Example:**

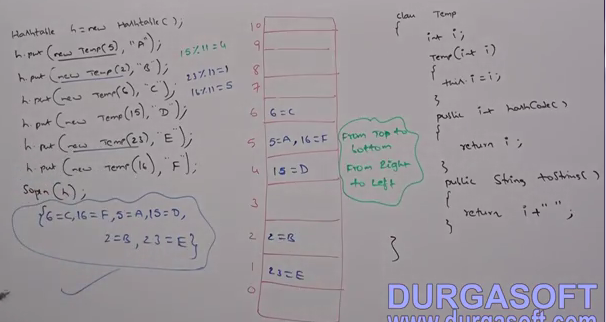
****

****

**While printing the hashtable, how it will be printed?**

From top to Bottom and Right to left from the hashtable, which highlight in green color in below snap

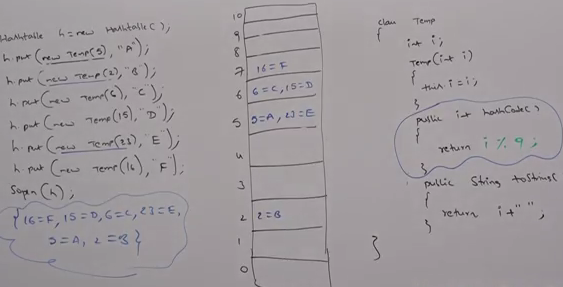
**How it is stored:**

****

**If we change hashcode() of Temp class as**

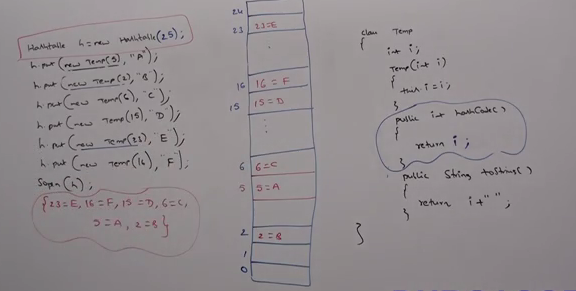
**Public int hashCode(){ return i% 9;}**

**Then the storage is like as follows. If there is change in hashcode, then there is change in output or storage.**

****

**If we change initial capacity of hashmap, then the storage is also differs.**

**Hashmap m =new Hashmap(25);**

****

**Properties:**

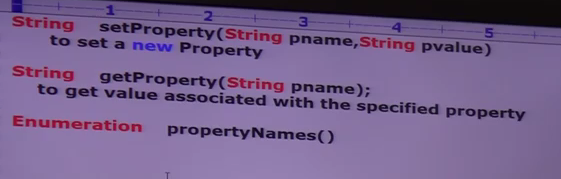
In our program if any changes frequently used like db username, password, is not recommended to hard code in java program.

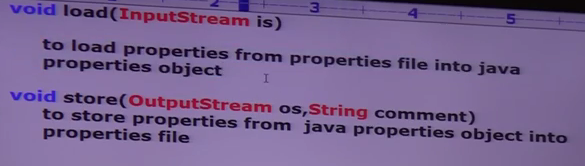
**In normal map like Hashmap, Hashtable, Treemap, key and value can be any type. But in the case of Properties, key and value should be String type.**

**Constructors:**

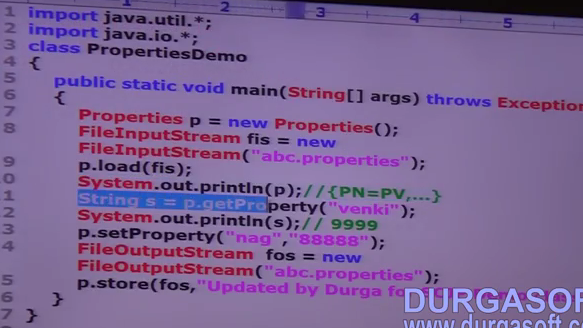
**Properties p=new Properties()**

**Methods:**

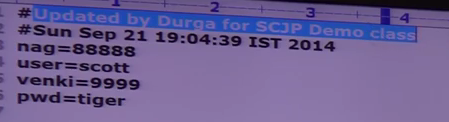




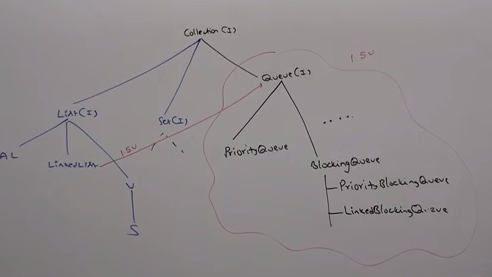
Example:



Property File:



**1.5 Enhancement – QUEUE (I):**



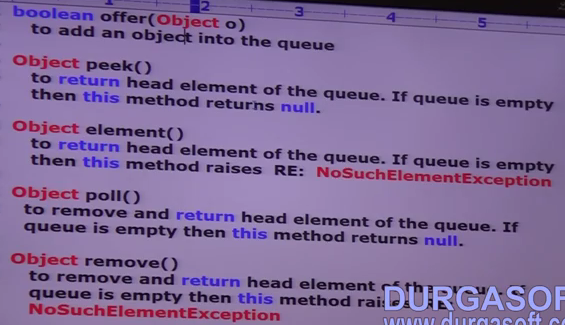
1. It is a child interface of Collection Interface
2. If we want to represent a group of individual objects prior to processing, then we should go for Queue.

Ex: Before sending messages, all mobile numbers we have to store in some data structure. In which order we added mobile nos, in the same order only message should be delivered. For this FIFO requirement, Queue is the best choice.

Usually Queue follows, First In First Out order but based on our requirement, we can implement our own priority requirement.

From 1.5 onwards, LinkedList class also implements Queue Interface. LinkedList based implementation of queue also follows FIFO.

**Methods:**

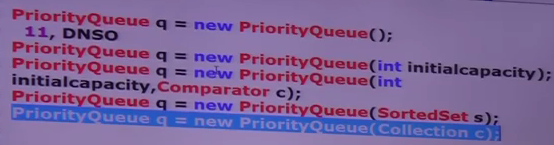


**PriorityQueue:**

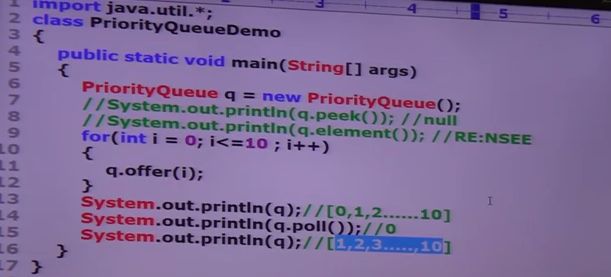
1. Represent a Group of individual objects prior to processing according to some prioriy.
2. The priority can be either default natural sorting order or customized sorting order defined by comparator.
3. Insertion order is not preserved and it is based on some priority.
4. Duplicate objects are not allowed.
5. If we are depending default natural sorting order, the objects should be homogeneous and comparable otherwise we we will RE : ClassCast Exception.
6. If we defining Customzied Sorting order by Comparator, then the objects need not be homogeneous and comparable.
7. Null is not allowed even as the first element.

**Constructors:**

1. PriorityQueue p =new PriorityQueue() -> Creates a Empty Priority Queue with default initial capacity of 11 and all objects will be inserted according to default sorting order.



Example (DSO): -Default Sorting Order

****

**Note: Some operating system or platforms wont provide proper support for Thread Priorities and Priority Queues. The sop in above pgm, we should 1 to 10 after removal head element. But it wont get because not providing support by platform.**

Example (CSO): -Customized Sorting Order

