**Hash Map**:

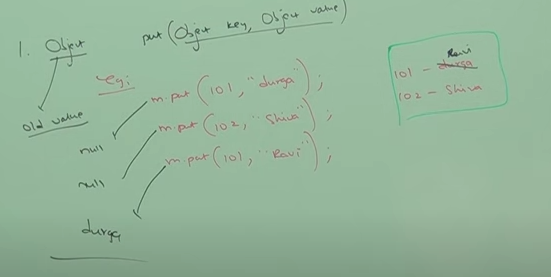
Map is not child interface of collection. If we want to represent a group of objects as key/value pair—map. Duplicate keys are not allowed. Values can be duplicated. Each key value pair is called **Entry**.

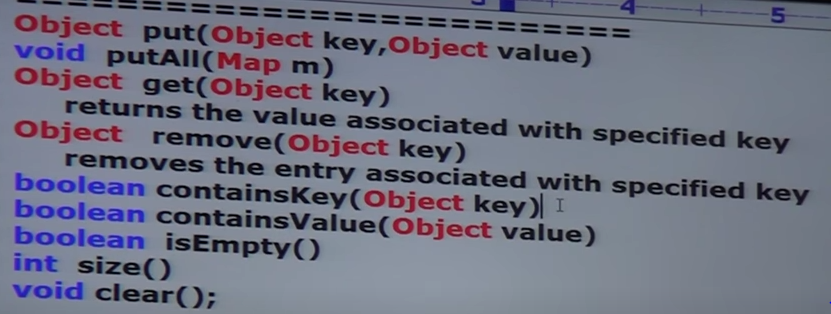
**Map Interface methods:**

1. **Object put (Object key, Object value);**

To add one key value pair to the map. If the key is already present then old value will be replaced will be replaced with new value and returns old value.

Ex:





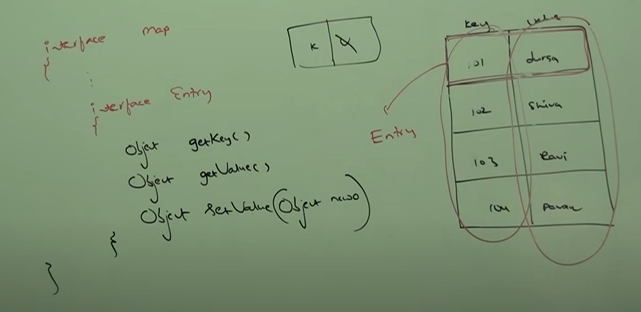
**Collection Views of Map**

1. Set KeySet ()
2. Collection values ()
3. Set entrySet ()

**Entry**:

Map is a group of key value pair. Each key/value pair is called an Entry. Hence, map is considered as a collection of entry objects. Without existing Map object there is no existence of Entry object.

So, Entry interface is defined within Map interface.



Above 3 methods are Entry specific methods and can be applied only in Entry objects

**Hash-Map:**

1. Underlying data structure is Hashtable.
2. Insertion order is not preserved and it is based on hash code of Keys
3. Duplicate Keys==not allowed, Duplicate Values == Allowed
4. Heterogeneous objects are allowed for both key and value.
5. Null is allowed for key (Only once).
6. Null is allowed for values (Any number of times).
7. Hash Map implements Serializable and Cloneable interface but not Random access.
8. Best choice if our frequent operation is search.

HashMap m =new HashMap ();

It creates an empty Hash Map object with default initial capacity 16 and default fill ration 0.75.

HashMap m = new HashMap (int initCapacity);

Creates an empty HashMap object with specified initial capacity and default fill ratio 0.75.

HashMap m = new HashMap (int initCapacity, float fillRatio);

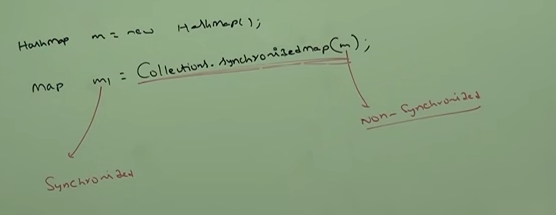
HashMap m = new HashMap (Map m);

**Difference between Hash Map and Hash Table**

|  |  |
| --- | --- |
| **Hash Map** | **Hash Table** |
| Every method present in Hash Map is not Synchronized | Every method in Hash Table is Synchronized |
| Relatively performance is high because threads are not required to wait to operate on HashMap object | Relatively performance is low because threads are required to wait to operate on Hash Table Object. |
| Not Thread safe. Multiple thread can access at a time on Hash Map object | Thread Safe. Only one thread can operate at a time. |
| Null is allowed for both key and value | Null is not allowed for keys and values. Otherwise, we will get Null Pointer exception |
| Introduced in 1.2 version and it is not legacy | Introduced in 1.0 version and it is legacy |

**Synchronized version of Hash Map**

By default, Hash Map is Non- Synchronized. But we can get the Synchronized version of Hash Map by using synchronized Map method of collections class.



**Linked HashMap:**

Child class of Hash Map. It is exactly same as HashMap (including method and constructor) except following differences:

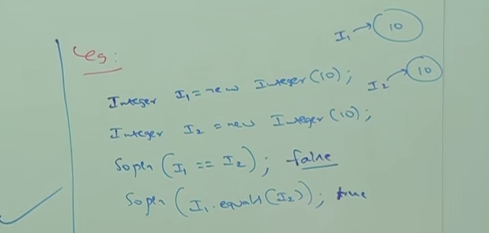
|  |  |
| --- | --- |
| **HashMap** | **Linked HashMap** |
| 1. Underlying data structure is Hash Table | 1. Combination of Linked List + Hash Table (Hybrid Data Structure) |
| 1. Insertion Order is not preserved. It is based on Hash Code of Keys. | 1. Insertion Order is preserved. |
| 1. Introduced in 2.0 version. | 3. In 1.4 version |

Linked HashMap and Linked Hash Set are commonly used for developing cache based application.

**Difference between == and .equals () method**

In general, == operator is meant for reference comparison (address comparison). Whereas, .equals() method meant for content comparison.

Ex:

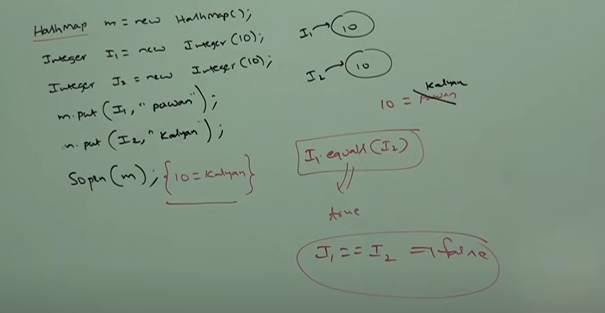


**IdentityHashMap:**

It is exactly same as HashMap (Including methods and constructors) except the following differences:

In case of normal HashMap JVM will use .equals() method to identify duplicate keys, which is meant for content comparison.

In the case of IdentityHashMap JVM will use == operator to identify duplicate keys, which is meant for reference comparison (Address comparison).

Ex: 

I1 and I2 are duplicate keys because I1.equals(I2) returns true.

If we replace HashMap with IdentityHashMap then I1 and I2 are not duplicate keys.Because I1==I2 returns false.

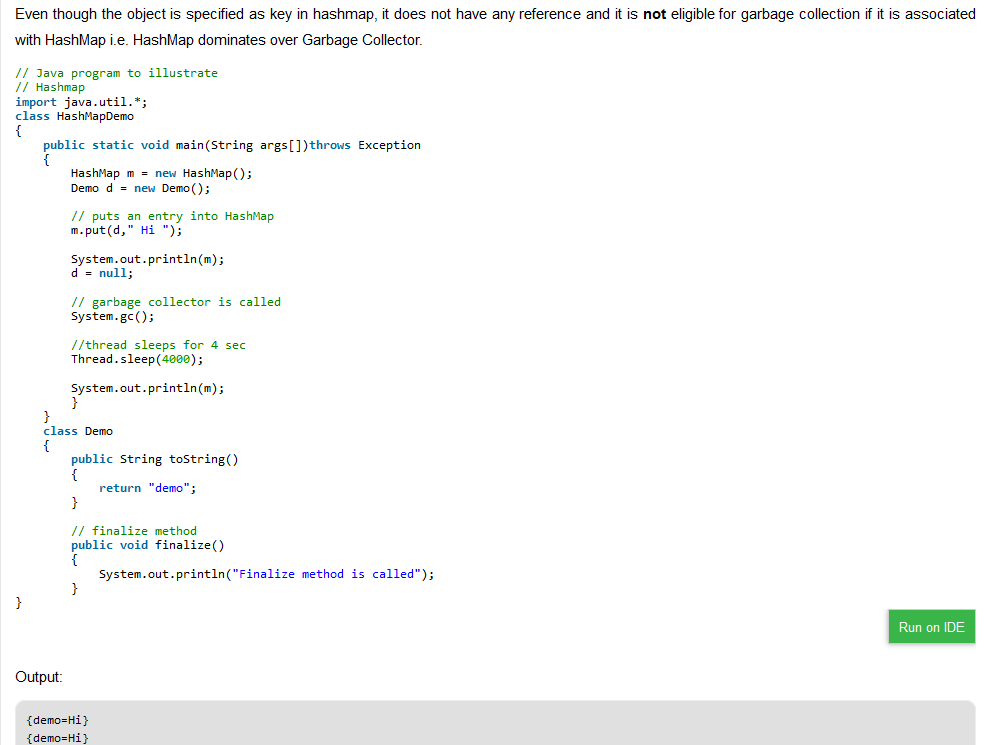
In this case output is: 10==pawan,10==kalyan

**WeakHashMap:**

It is exactly same as HashMap except the following difference.

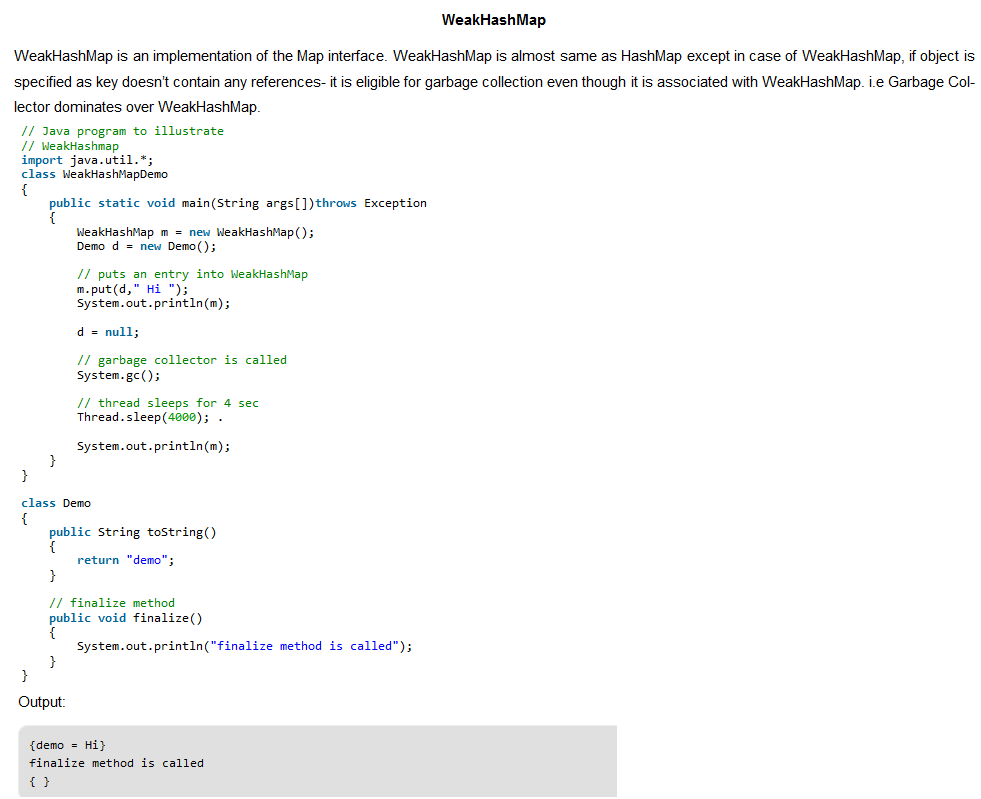
In case of HashMap even though object does not have any reference, it is not eligible for GC if it is associated with HashMap i.e. HashMap dominates garbage collector.

In case of WeakHashMap if the object does not contain any references, it is eligible for GC even though object associated with WeakHashMap i.e. Garbage collector dominates weakHashMap.



In the above example Demo object is not eligible for GC. Because it is associated with HashMap.

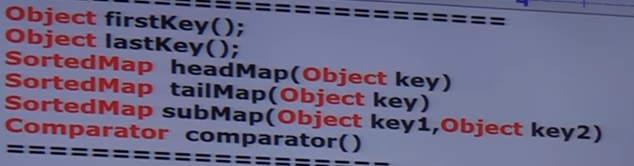
In case of WeakHashMap Demo Object eligible for GC.

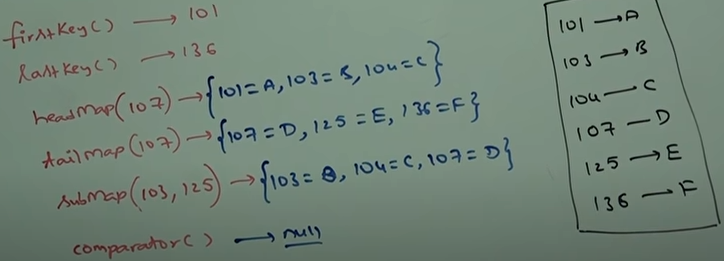


**SortedMap:**

Child Interface of Map. Group of key-value pairs will be sorted based on keys.

Methods present in SortedMap:





**TreeMap:**

* Underlying data structure is RED-BLACK Tree.
* Insertion order is not preserved.
* Based on sorting order of keys
* Duplicate Keys are not allowed. Values can be duplicated.
* If we are depending on default natural sorting order then keys should be homogeneous and comparable. Otherwise, we will get runtime exception saying: ClassCastException.
* If we are defining own sorting by comparator then keys need not be homogeneous and comparable. We can take heterogenous and non-comparable object also.
* Whether we are depending on default natural sorting order or customized sorting order. There are no restrictions for values. We can take heterogeneous non-comparable object also.

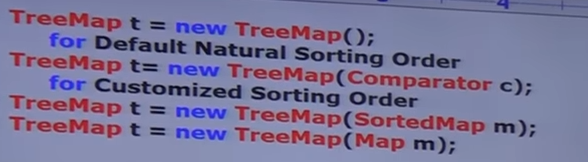
**Null Acceptance:**

1. For non- empty TreeMap, if we are trying to make an entry with null key🡪We will get NPE
2. For empty TreeMap 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 🡪NPE

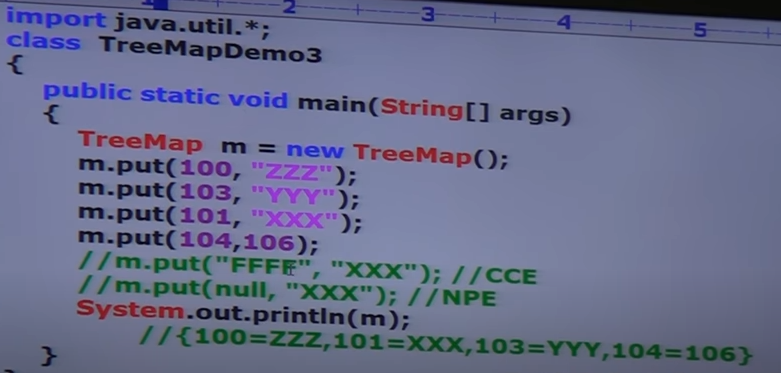
The above null acceptance rule applicable until 1.6 version only. From 1.7 onwards null is not allowed for key.

But for values we can use null any number of times. There is no restriction whether it is 1.6 or 1.7 version.

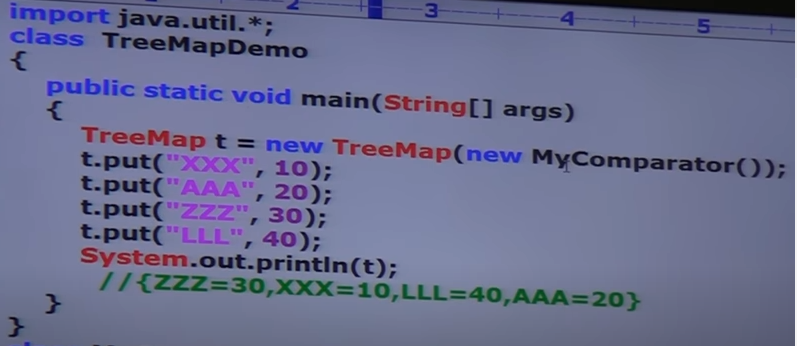
**Constructors in TreeMap**

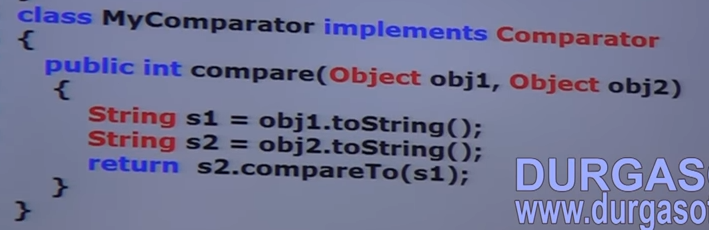


**Demo Program for default sorting:**



**Demo Program for Customized Sorting:**





**Hashtable**

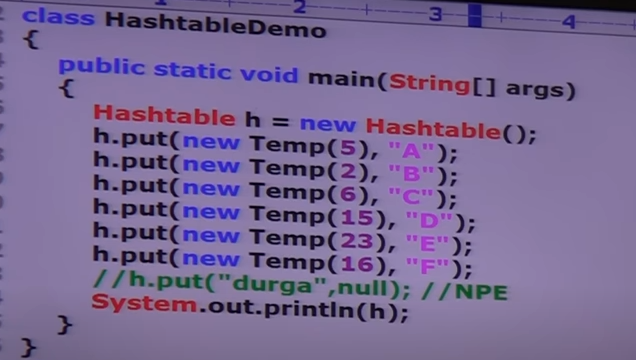
1. Underlying data structure for Hashtable is Hashtable.
2. Insertion order is not preserved and it is based on **hashCode of keys.**
3. Duplicate Keys are not allowed and values can be duplicated.
4. Heterogeneous objects are allowed for both keys and values.
5. Null is not allowed for both key and value. Otherwise, we will get runtime exception saying:

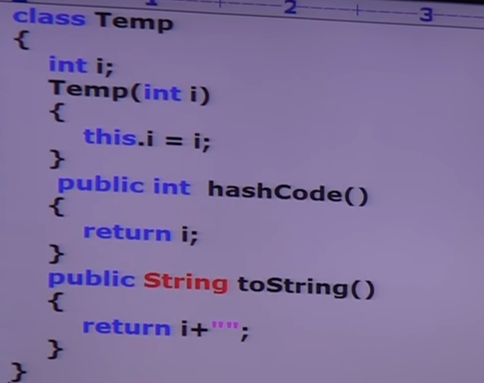
NullPointerException

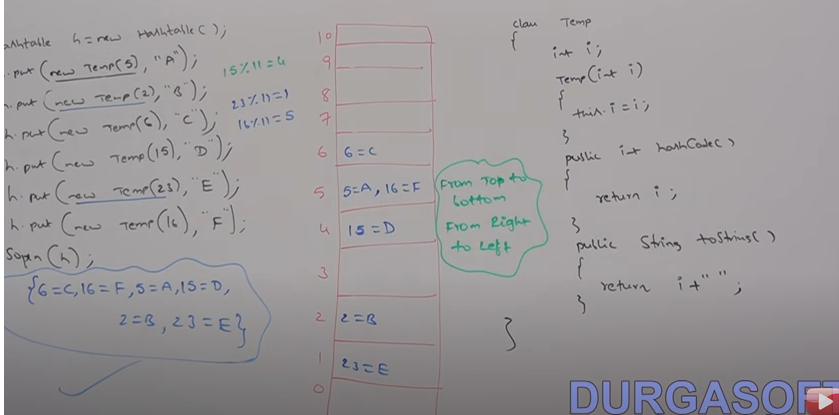
1. Implements Serializable and Cloneable interfaces. But **not Random Access.**
2. Every method in present in Hashtable is synchronized. Hence,Hashtable object is thread safe.
3. Hashtable is the best choice if our frequent operation is search operation.

**Constructors:**

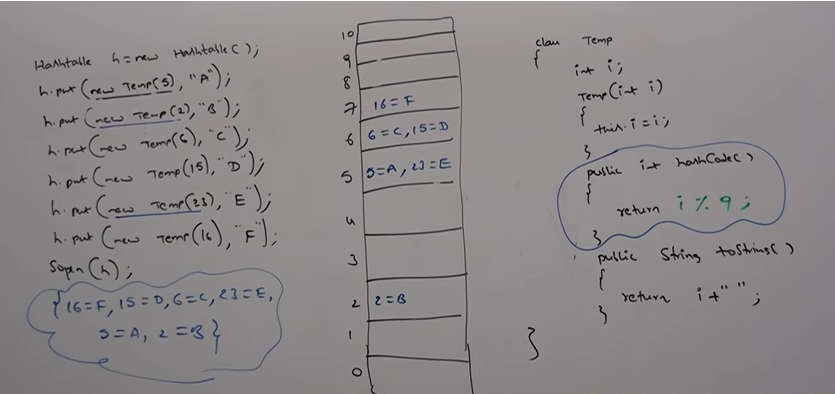
1. Hashtable h = new Hashtable ()🡪Creates an empty Hashtable object with default initial capacity 11 and default field ration 0.75.
2. Hashtable h = new Hashtable (int initCapacity)🡪With user defined initial capacity
3. Hashtable h = new Hashtable (int initCapacity, float fillratio)
4. Hashtable h = new Hashtable (Map m);



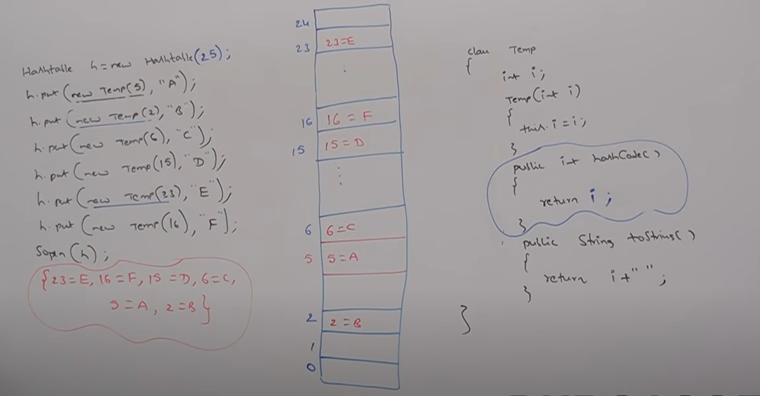




If we change hashCode method of Temp as i%9



If we configure initial capacity as 25



**Properties:**

In our program if anything which changes frequently (like username, password, url, host of db server) or not recommended to hard code in java program (If there is any change, to reflect that change recompilation, rebuild and redeployment of application is required). Sometimes server restart also required. It creates a big business impact to client.

We can create a property file and store the key value pair. Load the property file in java and use it. Advantages: If there is a change in property file, to reflect that change just redeployment is enough. It won’t create any business impact.

We can use java properties object to hold properties which are coming from properties file.

In normal Map (HashMap, Hashtable…) key and value can be ay type. But in case of properties,

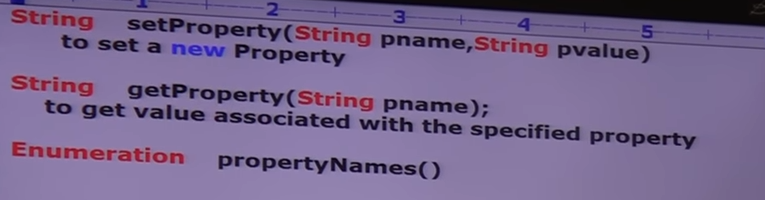
Key and Value should be String type.

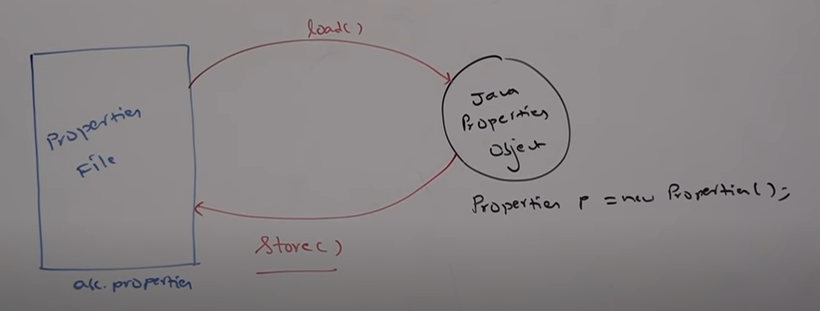
**Methods in Properties:**

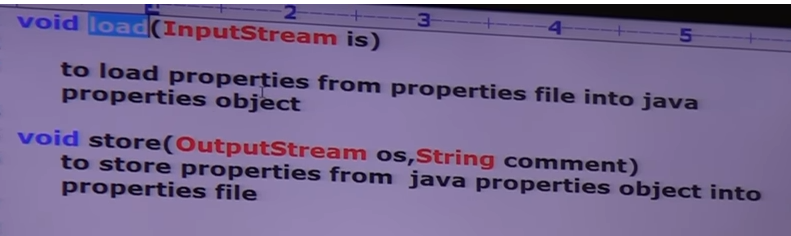
setProperty🡪 if the specified property already exists then the old value will be replaced with new value and returns old value.

getProperty🡪If property not available it returns null.

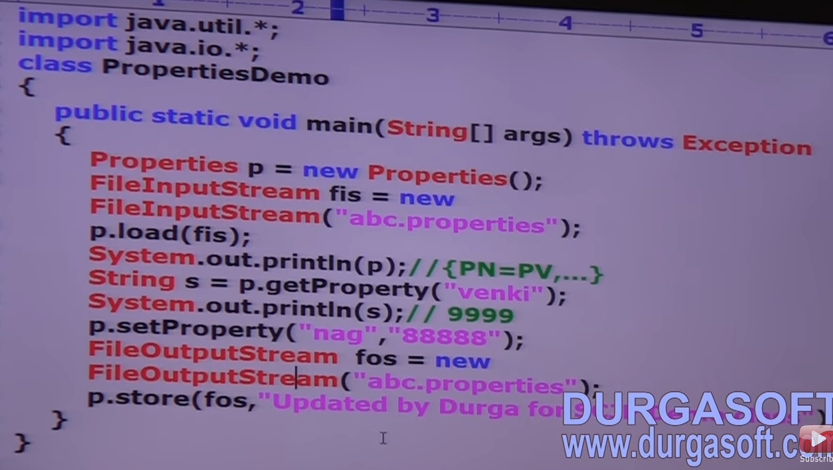
propertyNames()🡪Returns all property present in property object







Ex:



**Queue:**