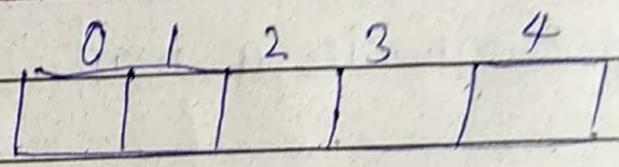
(assumption)

How hash func^n works.

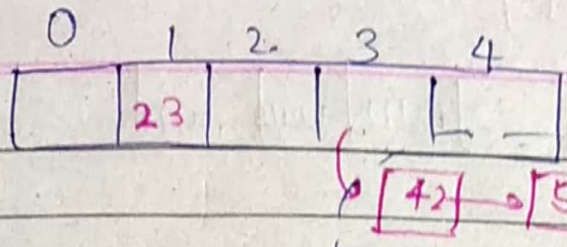


keys	values	hash
CAT	23	216
DOG	42	218
BALL	51	283
GOD	7	218

⇒ 216%5 =
⇒ 218%5 =
⇒ 283%5 =
⇒ 218%5 = 3

~~Public class Stath~~

```
public int hash(String s){
    int hash = 0;
    for(int i=0; i<s.length(); i++){
        hash= hash + s.charAt(i);
    }
    return hash;
}
```



Collision.

(this can be overcome by chaining)

↓↓

In this we make a linkedlist as shown above

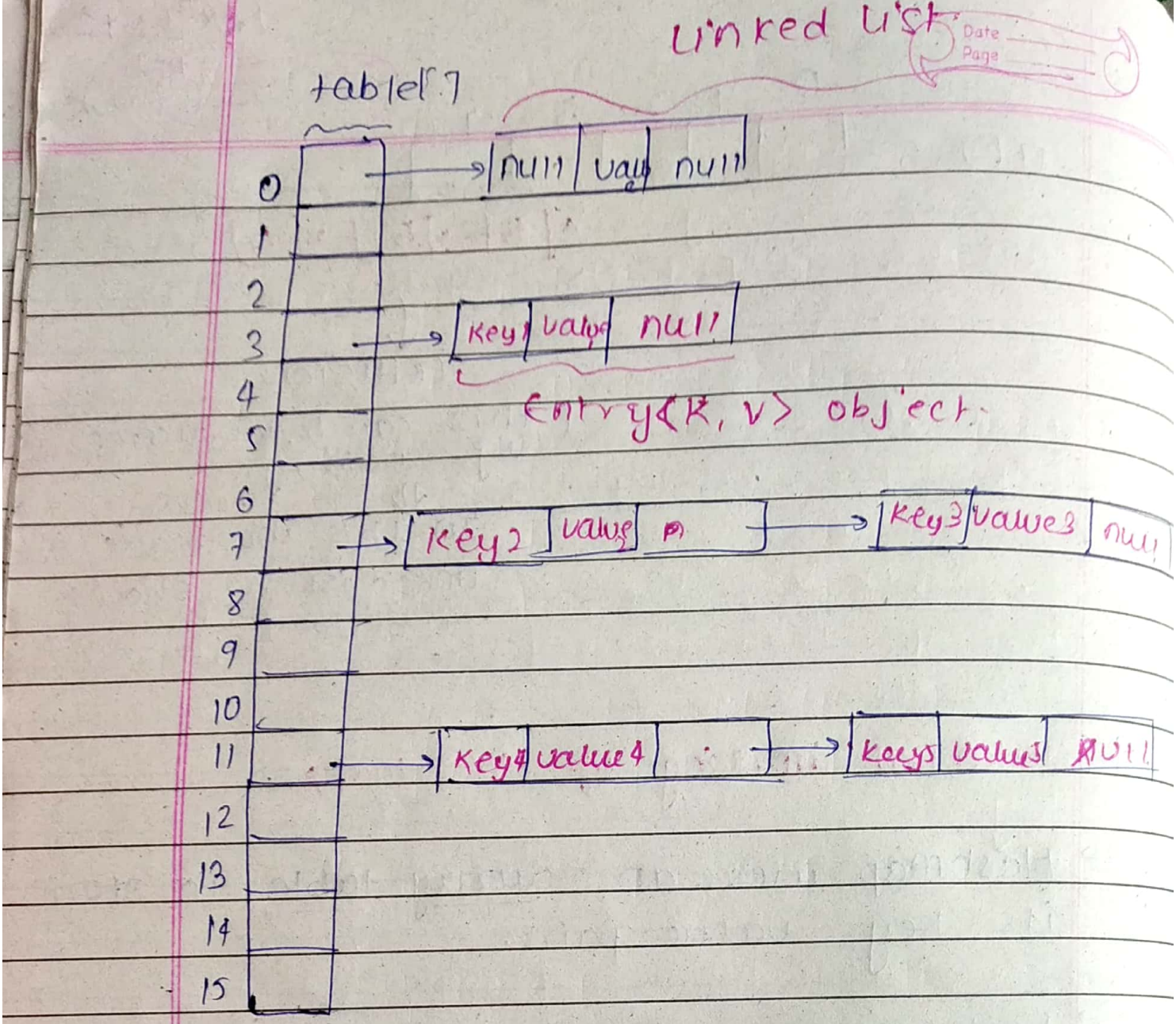
Internal working of HashMap

→ HashMap uses an array table to store its key value pairs.

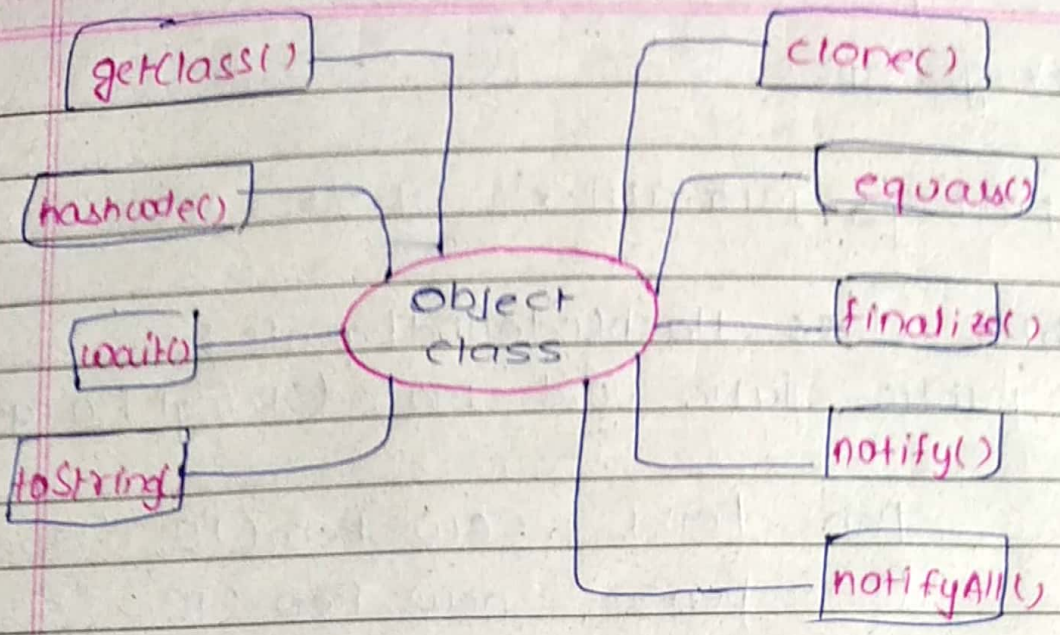
→ Each element of the array holds the head of a linkedlist to avoid collision

→ The hashcode of every key is calculated & the elements are placed in the array using this hash func^{n}

→ The default capacity is kept at 16 & load factor at 0.75.



→ By default java has 0.75 load factor
It means that $\frac{16 \times 75}{100} = 12$.
when we reach 12 elements in array table then size of table must be double i.e. 32 & we have to do rehashing.



hashCode() and equals() methods

→ hashCode() and equals() methods have been defined in Object class which is parent class for java objects.

→ For this reason, all java objects inherit a default implementation of these methods

- Java hashCode()

Object class defined hashCode() method like this

```
pubic int hashCode();
    // TO DO return the hashcode;
}
```

- Java equals(

Object class define equals() method li th

```
public boolean
equals(Object ob
    }
    return(this == obj
}
```