HashMap
Concurrent HashMap
Hash Table

UNIQUE

SOWJANYA MORISETTY
YASHWANTH RAVULA
MEGHANA BAREDDY
RAYONA MATHEW

HashMap

- CONTENTS:
  - Introduction
  - How HashMap works
  - Usage and Examples
  - HashMap Collision
  - Methods in HashMap
  - Advantages of HashMaP

HashMap

- **HashMap<K, V>** is a part of Java's collection since Java 1.2. This class is found in **java. util** package.
- It provides the basic implementation of the Map interface of Java.
- It stores the data in (Key, Value) pairs, and you can access them by an index of another type (e.g. an Integer).
- One object is used as a key (index) to another object (value). If you try to insert the duplicate key, it will replace the element of the corresponding key.

HashMap<K,V>

Java HashMap contains only unique keys

```
import java.util.*;
public class  ProgrammingLanguages {
 public static void main(String args[]){
   HashMap<Integer,String> map=new HashMap<Integer,String>();
   map.put(1,"JAVA");
   map.put(2,"PYTHON");
   map.put(3,"JAVA8");
   map.put(3,"C");
     System.out.println(name);
         }
 } }
```

- **OUTPUT**

```
1          JAVA
2        PYTHON
3  C
```

HOW HASHMAP WORKS

- It uses a technique called Hashing. It implements the map interface. It stores the data in the pair of Key and Value.
- HashMap contains an array of the nodes. It uses an array and LinkedList data structure internally for storing Key and Value.
- There are four fields in HashMap.

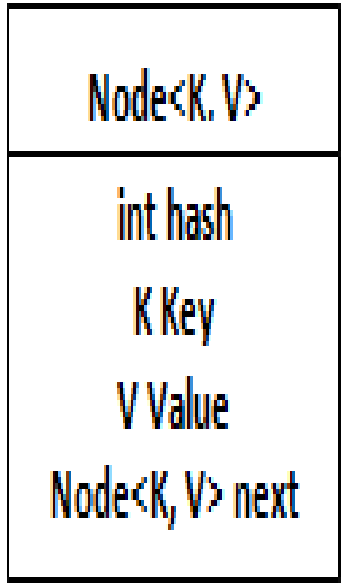


Figure: Representation of a Node

equals()
hashCode()

- **equals():** It checks the equality of two objects. It compares the Key, whether they are equal or not. It can be overridden. If you override the equals() method, then it is mandatory to override the hashCode() method.
- **hashCode():** It returns the memory reference of the object in integer form. The value received from the method is used as the bucket number. The bucket number is the address of the element inside the map.
- **Buckets:** Array of the node is called buckets. Each node has a data structure like a LinkedList.

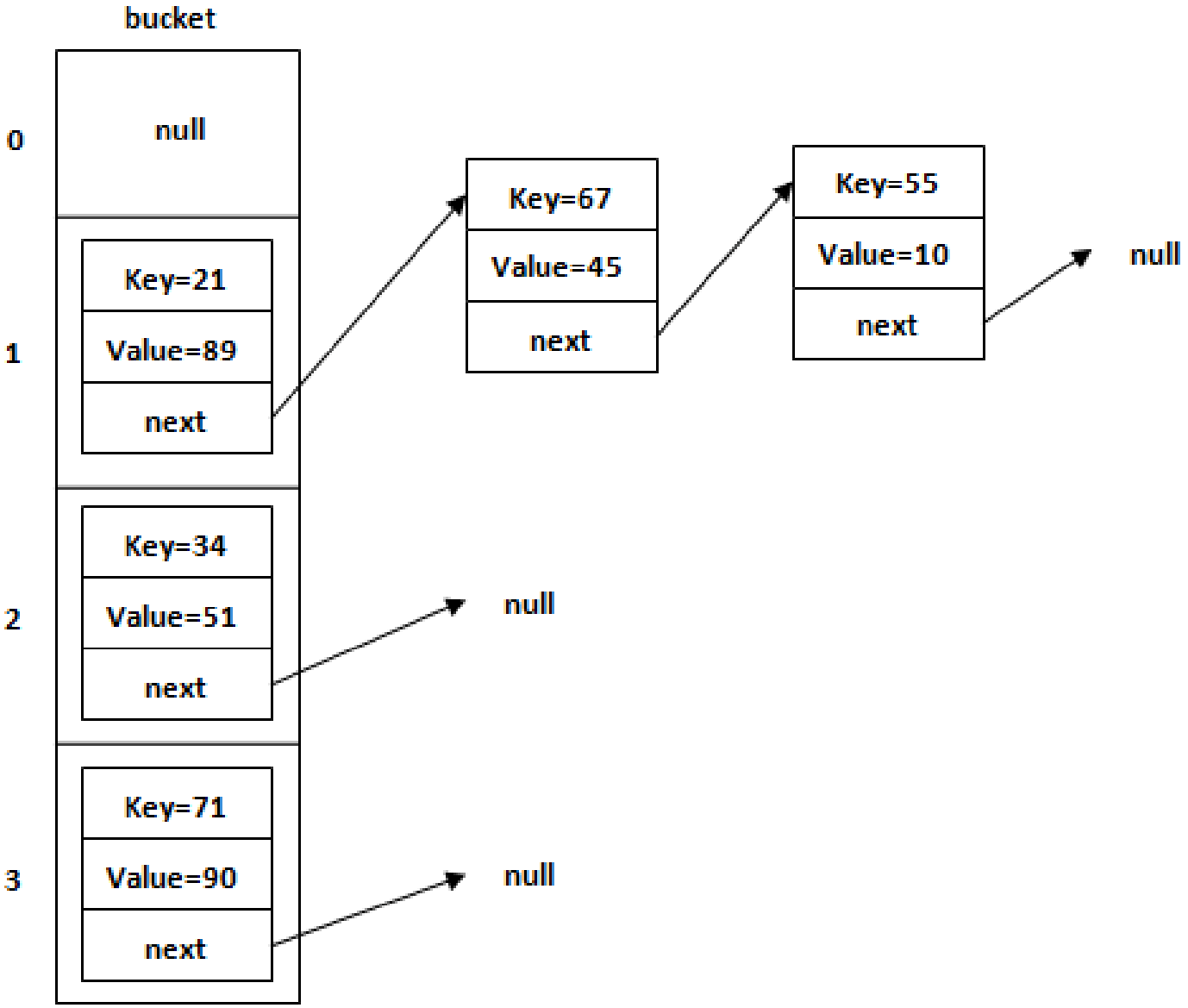


Figure: Allocation of nodes in Bucket

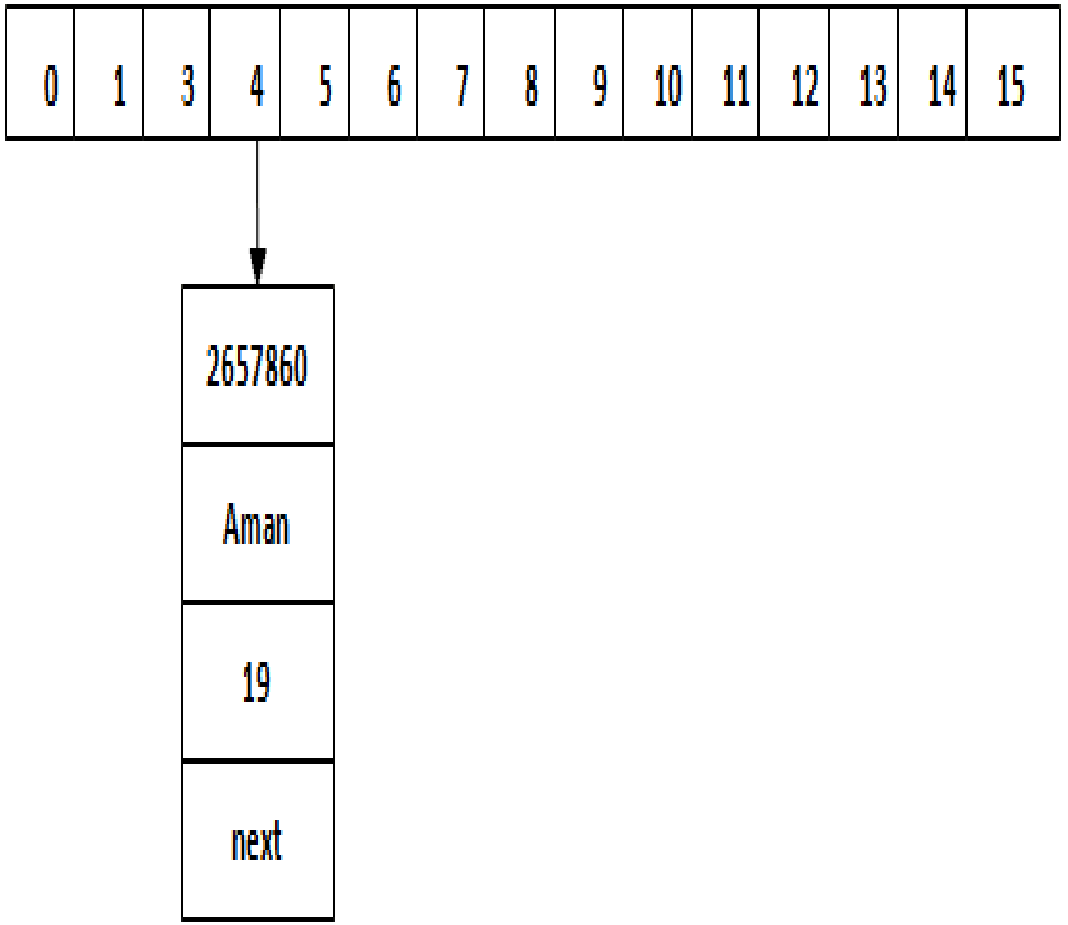
Index

Calculating Index :

Index = hashCode(Key) & (n-1)

EX :-

- Index = 2657860 & (16-1) = 4
- The value 6 is the computed index value where the Key and value will store in HashMap.



HASH COLLISION

- This is the case when the calculated index value is the same for two or more Keys.
- This scenario can occur because according to the *equals* and *hashCode* contract, **two unequal objects in Java can have the same hash code**.
- It can also happen because of the finite size of the underlying array, that is, before resizing. The smaller this array, the higher the chances of collision.

METHODS IN HASHMAP

Clear()

IsEmpty()

Put(object key,object value)

Remove(object key)

get(object key)

void clear() :

It is used to remove all of the mappings from this map.

boolean isEmpty():

It is used to return true if this map contains no key-value mappings.

V put(Object key, Object value):

It is used to insert an entry in the map.

V remove(Object key):

It is used to delete an entry for the specified key.

V get(Object key)

This method returns the object that contains the value associated with the key.

ADVANTAGES OF HASHMAP

- Allows insertion of key value pair.
- HashMap is non synchronized. HashMap cannot be shared between multiple threads without proper synchronization.
- HashMap is a fail-fast iterator.
- Faster access of elements due to hashing technology.

Concurrent HashMap

Concurrent HashMap

CONTENTS:

I. Introduction

II. How Concurrent HashMap works

III. Advantages of Concurrent HashMap

IV. Usage and Examples

V. Limitations and Challenges

Concurrent HashMap

1. Introduction

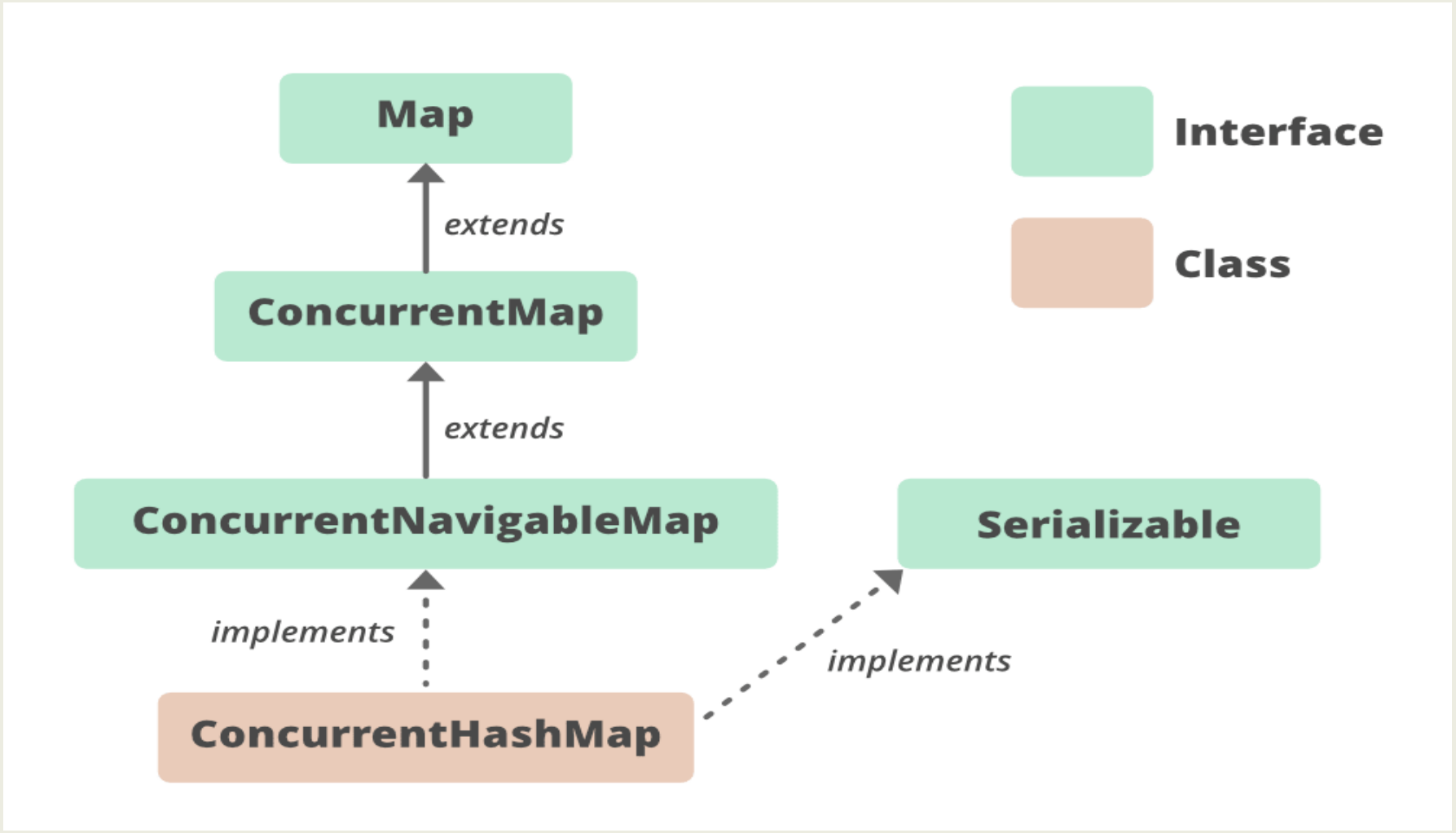
- Concurrent HashMap is an enhancement of HashMap as we know that while dealing with Threads in our application HashMap is not a good choice because performance-wise HashMap is not up to the mark.
- Concurrent HashMap is a thread-safe implementation of the Map interface in Java, which means multiple threads can access it simultaneously without any synchronization issues. It's part of the java.util.concurrent package.
- In a Concurrent HashMap, the map is divided into segments, each of which can be locked independently. This means that multiple threads can read from different segments of the map simultaneously, while only one thread can modify a given segment at any given time. This allows for higher throughput and scalability than a traditional synchronized HashMap, which locks the entire map for each operation.

- In addition to its concurrency features, ConcurrentHashMap also provides the same functionality as a regular HashMap, including methods for adding, removing, and retrieving key-value pairs, as well as support for iterators and various collection views.
- Overall, ConcurrentHashMap is a useful tool for situations where multiple threads need to access a shared map concurrently, as it provides a high level of performance and scalability while ensuring thread safety.
- In summary, ConcurrentHashMap is an important addition to Java's collections framework, providing a safe and efficient way for multiple threads to access and modify shared data structures. It is widely used in multi-threaded applications where thread-safety is a critical requirement.

II. How ConcurrentHashMap works

- ConcurrentHashMap is a thread-safe implementation of the hash table data structure that allows multiple threads to access and modify its contents concurrently, without requiring explicit locking or synchronization mechanisms.
- The internal structure of ConcurrentHashMap is based on the partitioning of the hash table into segments. Each segment is a separate hash table that operates independently, allowing concurrent access to different segments without affecting each other.
- To insert or update a key-value pair, ConcurrentHashMap first determines the segment that the key belongs to using its hash code. It then acquires a lock only for that specific segment and performs the operation on that segment. This approach allows multiple threads to operate on different segments concurrently, without requiring a global lock or synchronization mechanism.

- Similarly, to read or retrieve a value associated with a key, ConcurrentHashMap first determines the segment that the key belongs to and reads the value from that segment. Since each segment is accessed independently, multiple threads can read values from different segments concurrently without any contention.
- In summary, ConcurrentHashMap achieves high concurrency by partitioning the hash table into separate segments and providing fine-grained locking at the segment level. This approach allows multiple threads to operate concurrently on different segments of the hash table, without blocking each other or causing contention.



Methods in ConcurrentHashMap

Clear()

Contains()

hashcode()

containsKey(Object key)

get(object key)

void clear() :

It is used to remove all of the mappings from this map.

boolean isEmpty():

It is used to return true if this map contains no key-value mappings.

V put(Object key, Object value):

It is used to insert an entry in the map.

V remove(Object key):

It is used to delete an entry for the specified key

V get(Object key)

This method returns the object that contains the value associated with the key.

```
import java.util.concurrent.ConcurrentHashMap;

 public class Main { public static void main(String[] args) {

 ConcurrentHashMap<String, Integer> map = new ConcurrentHashMap<String, Integer>(); // Add some key-value pairs to the map

map.put("apple", 1);

 map.put("banana", 2);

 map.put("cherry", 3);

// Print the current state of the map

System.out.println("Map contains:");

 for (String key : map.keySet()) {

System.out.println(key + " -> " + map.get(key));

 }
```

```
// Add a new key-value pair using putIfAbsent
   map.putIfAbsent("banana", 4
map.putIfAbsent("date", 5);
// Print the updated state of the map
System.out.println("Map contains:");
for (String key : map.keySet())
{ System.out.println(key + " -> " + map.get(key)); }
// Remove a key-value pair using remove
map.remove("cherry", 3);
// Print the final state of the map
System.out.println("Map contains:");
for (String key : map.keySet()) {
System.out.println(key + " -> " + map.get(key));
} } }
    }
}
```

Advantages of ConcurrentHashMap

- Thread-safety: ConcurrentHashMap provides thread-safety without the need for external synchronization. This means that multiple threads can access and modify the map simultaneously without causing any issues like data corruption or race conditions.

- Performance: ConcurrentHashMap is designed to provide high performance even under heavy concurrent access. It uses a technique called lock striping to partition the map into multiple segments, each of which can be locked independently. This reduces contention and improves concurrency.

- Scalability: ConcurrentHashMap is highly scalable, meaning it can handle a large number of concurrent requests without any degradation in performance. This makes it ideal for use in high-throughput applications like web servers or data processing pipelines.

- Memory efficiency: ConcurrentHashMap is memory-efficient because it only locks a portion of the map when a thread needs to access or modify it. This means that other threads can continue to access other parts of the map concurrently, without any delays.

- Iteration safety: ConcurrentHashMap provides safe and consistent iteration over the entries in the map. This means that if the map is modified while an iteration is in progress, the iteration will either reflect the state of the map before or after the modification, but not an inconsistent state.

Overall, ConcurrentHashMap is a highly efficient and thread-safe implementation of the Map interface in Java, which makes it an excellent choice for use in concurrent applications.

Usage and Examples

Here are some examples of when and how you might use a concurrent hashmap:

1. **Multithreaded applications:** When developing a multithreaded application, you may need to store and access data in a hashmap. Using a regular hashmap in this scenario could lead to data corruption or race conditions, as multiple threads may try to access or modify the hashmap at the same time. A concurrent hashmap can help prevent these issues.

2. **High-concurrency scenarios:** In high-concurrency scenarios, where many threads are accessing and modifying the hashmap frequently, a concurrent hashmap can provide better performance than a regular hashmap. This is because it allows multiple threads to read and write to different parts of the hashmap concurrently, reducing contention and improving throughput.

Limitations and Challenges

While concurrent hashmap offers thread-safe access to data and can improve performance in high-concurrency scenarios, it also has some limitations and challenges. Here are some of them:

1.**Increased memory usage:** Concurrent hashmap uses additional memory to maintain its thread-safe behavior, which can be a concern if memory usage is already a bottleneck in the application.

2.**Performance overhead:** Because concurrent hashmap is designed to be thread-safe, it may have additional performance overhead compared to a regular hashmap. This can impact performance in low-concurrency scenarios where the extra safety measures are not necessary.

3.Scalability: While concurrent hashmap can improve performance in high-concurrency scenarios, it may not scale well to extremely large datasets or very high numbers of concurrent threads.

4.**Concurrent modification:** While concurrent hashmap provides thread-safe access to data, it does not protect against concurrent modification of individual objects within the hashmap. This can lead to race conditions or data inconsistencies if not properly handled by the application.

5.**Complex implementation:** The implementation of concurrent hashmap can be more complex than a regular hashmap, which can make it more difficult to debug and maintain.

6.**Deadlocks:** Concurrent hashmap can be susceptible to deadlocks if not used properly. Deadlocks occur when two or more threads are waiting for each other to release a resource, causing a deadlock in the system.

Hashtable

Contents:

I. Introduction

II. Features of hashtable

III. Declaration

IV. Working of hashtable

V. Constructors

VI. Methods

Introduction

- A Hashtable is an array of a list. Each list is known as a bucket.
- The Hashtable class implements a hash table, which maps keys to values.
- The java.util.Hashtable class is a class in Java that provides a key-value data structure.

Features of hashtable

- Java Hashtable class contains unique elements.
- Hashtable class doesn't allow null key or value.
- It does not accept duplicate keys.
- Each index is known as a bucket/slot.

Features of hashtable(cont.)

- Java Hashtable class is synchronized.
- Enumerator in Hashtable is not fail-fast.
- It is thread-safe i.e, At a time only one thread is allowed to operate the Hashtable’s object.

Declaration

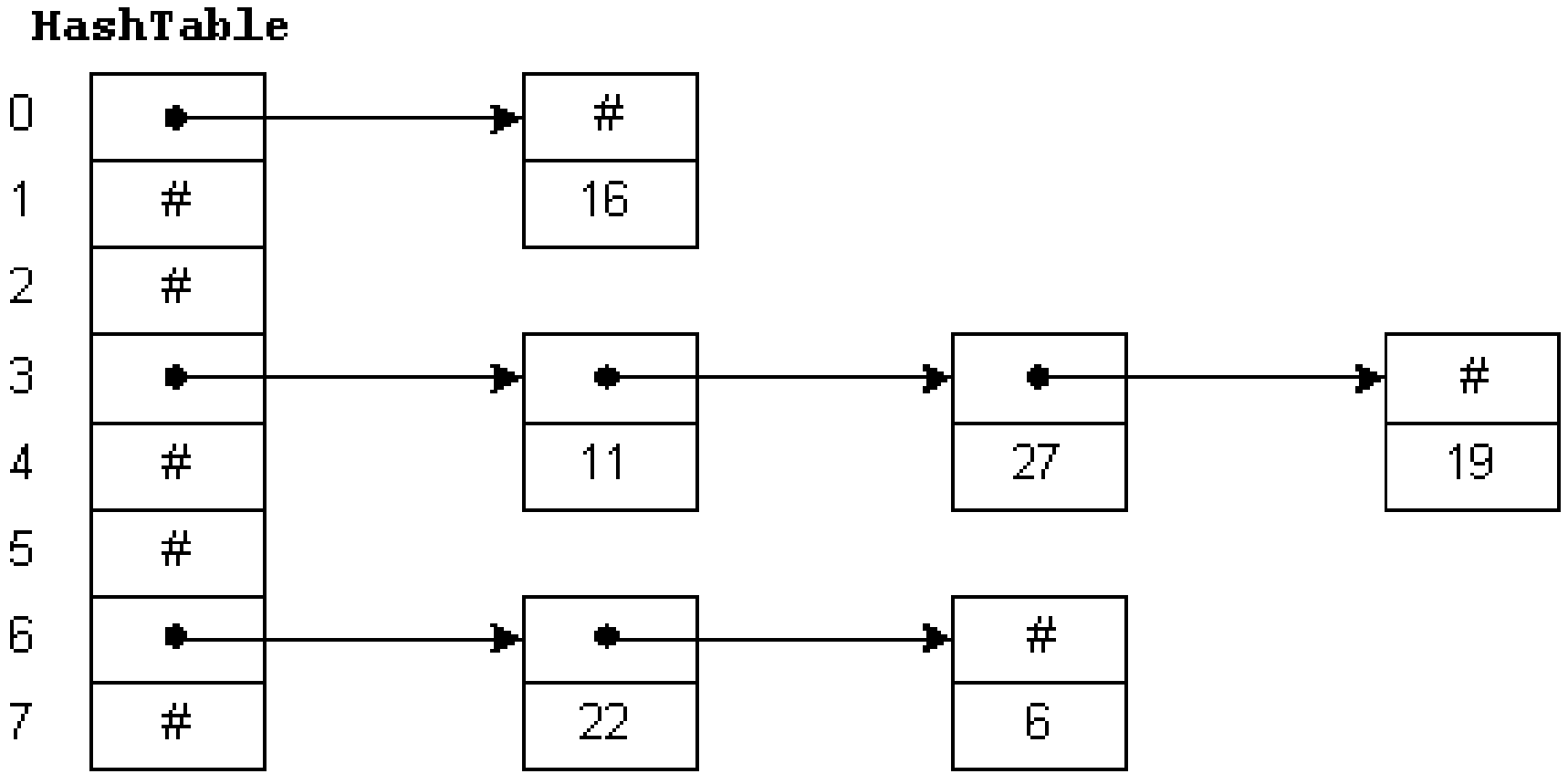
Hashtable<k,v> tablename=**new** Hashtable<k,v>();

K : It is the type of keys maintained by this map.

V : It is the type of mapped values.

Eg : Hashtable<Integer,String> hm=**new** Hashtable<Integer,String>();

Working of Hashtable



Working of Hashtable(cont.)

- Hashtable internally contains buckets in which it store the key/value pairs.

- The Hashtable uses the key's hashcode to determine to which bucket the key/value pair should map.

- The function to get bucket location from Key's hashcode is called hash function.

Constructors

- Hashtable()
- Hashtable(int size)
- Hashtable(int size, float fillration)
- Hashtable(Map m)

Methods

- void clear()
- boolean contains(Object value)
- boolean containsKey(Object key)
- boolean isEmpty()
- Void rehash()

Methods(cont.)

- Object get(Object key)
- Object put(Object key, Object value)
- Object remove(Object key)
- int size()

Thank you