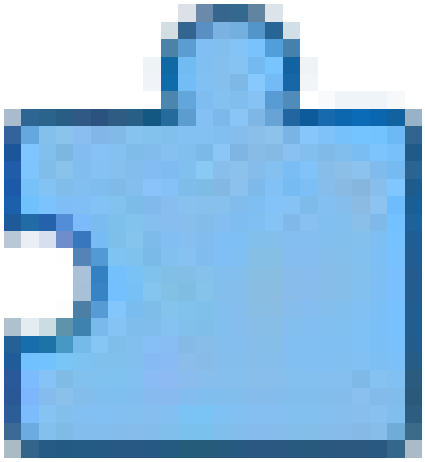


Java Map Interface

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

Java Map Hierarchy

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap. The hierarchy of Java Map is given below:

A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

A Map can't be traversed, so you need to convert it into Set using keySet() or entrySet() method.

Class	Description
HashMap	HashMap is the implementation of Map, but it doesn't maintain any order.
LinkedHashMap	LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order.
TreeMap	TreeMap is the implementation of Map and SortedMap. It maintains ascending order.

Useful methods of Map interface

Method	Description
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.

V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
Set keySet()	It returns the Set view containing all the keys.
Set<Map.Entry<K,V>> entrySet()	It returns the Set view containing all the keys and values.
void clear()	It is used to reset the map.
V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null.
V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V get(Object key)	This method returns the object that contains the

	value associated with the key.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
int hashCode()	It returns the hash code value for the Map
boolean isEmpty()	This method returns true if the map is empty; returns false if it contains at least one key.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection values()	It returns a collection view of the values contained in the map.
int size()	This method returns the number of entries in the map.

Map.Entry Interface

Entry is the subinterface of Map. So we will be accessed it by Map.Entry name. It returns a collection-view of the map, whose elements are of this class. It provides methods to get key and value.

Methods of Map.Entry interface

Method	Description
K getKey()	It is used to obtain a key.
V getValue()	It is used to obtain value.

int hashCode()	It is used to obtain hashCode.
V setValue(V value)	It is used to replace the value corresponding to this entry with the specified value.
boolean equals(Object o)	It is used to compare the specified object with the other existing objects.
static <K extends Comparable<? super K>,V> Comparator<Map.Entry<K,V>> comparingByKey()	It returns a comparator that compare the objects in natural order on key.
static <K,V> Comparator<Map.Entry<K,V>> comparingByKey(Comparator<? super K> cmp)	It returns a comparator that compare the objects by key using the given Comparator.
static <K,V extends Comparable<? super V>> Comparator<Map.Entry<K,V>> comparingByValue()	It returns a comparator that compare the objects in natural order on value.
static <K,V> Comparator<Map.Entry<K,V>> comparingByValue(Comparator<? super V> cmp)	It returns a comparator that compare the objects by value using the given Comparator.

Java Map Example: Non-Generic (Old Style)

```
1. //Non-generic
2. import java.util.*;
3. public class MapExample1 {
4. public static void main(String[] args) {
5.    Map map=new HashMap();
6.    //Adding elements to map
7.    map.put(1,"Amit");
8.    map.put(5,"Rahul");
9.    map.put(2,"Jai");
10.   map.put(6,"Amit");
11.   //Traversing Map
12.   Set set=map.entrySet();//Converting to Set so that we can traverse
13.   Iterator itr=set.iterator();
14.   while(itr.hasNext()){
```

```
15.      //Converting to Map.Entry so that we can get key and value separately
16.      Map.Entry entry=(Map.Entry)itr.next();
17.      System.out.println(entry.getKey()+" "+entry.getValue());
18.   }
19.}
20.}
```

Output:

```
1 Amit
2 Jai
5 Rahul
6 Amit
```

Java Map Example: Generic (New Style)

```
1. import java.util.*;
2. class MapExample2{
3.  public static void main(String args[]){
4.   Map<Integer,String> map=new HashMap<Integer,String>();
5.   map.put(100,"Amit");
6.   map.put(101,"Vijay");
7.   map.put(102,"Rahul");
8.   //Elements can traverse in any order
9.   for(Map.Entry m:map.entrySet()){
10.   System.out.println(m.getKey()+" "+m.getValue());
11.  }
12. }
13.}
```

Output:

```
102 Rahul
100 Amit
101 Vijay
```

Java Map Example: comparingByKey()

```
1. import java.util.*;
2. class MapExample3{
3.  public static void main(String args[]){
4. Map<Integer,String> map=new HashMap<Integer,String>();
5.      map.put(100,"Amit");
6.      map.put(101,"Vijay");
```

```
7.      map.put(102,"Rahul");
8.      //Returns a Set view of the mappings contained in this map
9.      map.entrySet()
10.     //Returns a sequential Stream with this collection as its source
11.     .stream()
12.     //Sorted according to the provided Comparator
13.     .sorted(Map.Entry.comparingByKey())
14.     //Performs an action for each element of this stream
15.     .forEach(System.out::println);
16. }
17.}
```

Output:

```
100=Amit
101=Vijay
102=Rahul
```

Java Map Example: comparingByKey() in Descending Order

```
1. import java.util.*;
2. class MapExample4{
3.  public static void main(String args[]){
4. Map<Integer,String> map=new HashMap<Integer,String>();
5.      map.put(100,"Amit");
6.      map.put(101,"Vijay");
7.      map.put(102,"Rahul");
8.      //Returns a Set view of the mappings contained in this map
9.      map.entrySet()
10.     //Returns a sequential Stream with this collection as its source
11.     .stream()
12.     //Sorted according to the provided Comparator
13.     .sorted(Map.Entry.comparingByKey(Comparator.reverseOrder()))
14.     //Performs an action for each element of this stream
15.     .forEach(System.out::println);
16. }
17.}
```

Output:

```
102=Rahul
101=Vijay
100=Amit
```

Java Map Example: comparingByValue()

```
1. import java.util.*;
2. class MapExample5{
3.  public static void main(String args[]){
4. Map<Integer,String> map=new HashMap<Integer,String>();
5.      map.put(100,"Amit");
6.      map.put(101,"Vijay");
7.      map.put(102,"Rahul");
8.      //Returns a Set view of the mappings contained in this map
9.      map.entrySet()
10.     //Returns a sequential Stream with this collection as its source
11.     .stream()
12.     //Sorted according to the provided Comparator
13.     .sorted(Map.Entry.comparingByValue())
14.     //Performs an action for each element of this stream
15.     .forEach(System.out::println);
16. }
17.}
```

Output:

```
100=Amit
102=Rahul
101=Vijay
```

Java Map Example: comparingByValue() in Descending Order

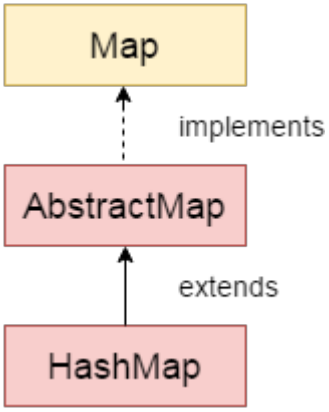
```
1. import java.util.*;
2. class MapExample6{
3.  public static void main(String args[]){
4. Map<Integer,String> map=new HashMap<Integer,String>();
5.      map.put(100,"Amit");
6.      map.put(101,"Vijay");
7.      map.put(102,"Rahul");
8.     //Returns a Set view of the mappings contained in this map
9.     map.entrySet()
10.    //Returns a sequential Stream with this collection as its source
11.    .stream()
12.    //Sorted according to the provided Comparator
13.    .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
14.    //Performs an action for each element of this stream
```

```
15.     .forEach(System.out::println);
16. }
17.}
```

Output:

```
101=Vijay
102=Rahul
100=Amit
```

Java HashMap



Java HashMap class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the `java.util` package.

HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key. Since Java 5, it is denoted as `HashMap<K,V>`, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.

Points to remember

- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.
- Java HashMap may have one null key and multiple null values.
- Java HashMap is non synchronized.
- Java HashMap maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Methods of Java HashMap class

Method	Description
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.
Object clone()	It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
Set entrySet()	It is used to return a collection view of the mappings contained in this map.
Set keySet()	It is used to return a set view of the keys contained in this map.
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null.
V computeIfPresent(K key, BiFunction<? super K,? super V,?	It is used to compute a new mapping given the key and its current mapped value if the value for the

extends V> remappingFunction)	specified key is present and non-null.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V get(Object key)	This method returns the object that contains the value associated with the key.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
boolean isEmpty()	This method returns true if the map is empty; returns false if it contains at least one key.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection<V> values()	It returns a collection view of the values contained in the map.

int size()	This method returns the number of entries in the map.

```
1. import java.util.*;
2. public class HashMapExample1{
3. public static void main(String args[]){
4.  HashMap<Integer,String> map=new HashMap<Integer,String>();//Creating HashMap

5.  map.put(1,"Mango");  //Put elements in Map
6.  map.put(2,"Apple");
7.  map.put(3,"Banana");
8.  map.put(4,"Grapes");
9.
10.  System.out.println("Iterating Hashmap...");
11.  for(Map.Entry m : map.entrySet()){
12.   System.out.println(m.getKey()+" "+m.getValue());
13.  }
14.}
15.}
```

Test it Now

```
Iterating Hashmap...
1 Mango
2 Apple
3 Banana
4 Grapes
```

In this example, we are storing Integer as the key and String as the value, so we are using `HashMap<Integer,String>` as the type. The `put()` method inserts the elements in the map.

To get the key and value elements, we should call the getKey() and getValue() methods. The `Map.Entry` interface contains the getKey() and getValue() methods. But, we should call the entrySet() method of Map interface to get the instance of Map.Entry.

No Duplicate Key on HashMap

You cannot store duplicate keys in HashMap. However, if you try to store duplicate key with another value, it will replace the value.

```
1. import java.util.*;
2. public class HashMapExample2{
```

```
3.  public static void main(String args[]){
4.   HashMap<Integer,String> map=new HashMap<Integer,String>();//Creating HashMap

5.   map.put(1,"Mango");   //Put elements in Map
6.   map.put(2,"Apple");
7.   map.put(3,"Banana");
8.   map.put(1,"Grapes"); //trying duplicate key
9.
10.  System.out.println("Iterating Hashmap...");
11.  for(Map.Entry m : map.entrySet()){
12.   System.out.println(m.getKey()+" "+m.getValue());
13.  }
14.}
15.}
```

Test it Now

```
Iterating Hashmap...
1 Grapes
2 Apple
3 Banana
```

Java HashMap example to add() elements

Here, we see different ways to insert elements.

```
1. import java.util.*;
2. class HashMap1{
3.  public static void main(String args[]){
4.   HashMap<Integer,String> hm=new HashMap<Integer,String>();
5.    System.out.println("Initial list of elements: "+hm);
6.     hm.put(100,"Amit");
7.     hm.put(101,"Vijay");
8.     hm.put(102,"Rahul");
9.
10.    System.out.println("After invoking put() method ");
11.    for(Map.Entry m:hm.entrySet()){
12.     System.out.println(m.getKey()+" "+m.getValue());
13.    }
14.
15.    hm.putIfAbsent(103, "Gaurav");
16.    System.out.println("After invoking putIfAbsent() method ");
```

```
17.     for(Map.Entry  m:hm.entrySet()){
18.         System.out.println(m.getKey()+"  "+m.getValue());
19.        }
20.     HashMap<Integer,String>  map=new  HashMap<Integer,String>();
21.     map.put(104,"Ravi");
22.     map.putAll(hm);
23.     System.out.println("After  invoking  putAll()  method  ");
24.     for(Map.Entry  m:map.entrySet()){
25.         System.out.println(m.getKey()+"  "+m.getValue());
26.        }
27. }
28.}
```

```
Initial list of elements: {}
After invoking put() method
100 Amit
101 Vijay
102 Rahul
After invoking putIfAbsent() method
100 Amit
101 Vijay
102 Rahul
103 Gaurav
After invoking putAll() method
100 Amit
101 Vijay
102 Rahul
103 Gaurav
104 Ravi
```

Java HashMap example to remove() elements

Here, we see different ways to remove elements.

```
1. import  java.util.*;
2. public  class  HashMap2  {
3.   public  static  void  main(String  args[])  {
4.    HashMap<Integer,String>  map=new  HashMap<Integer,String>();
5.     map.put(100,"Amit");
```

```
6.      map.put(101,"Vijay");
7.      map.put(102,"Rahul");
8.      map.put(103,  "Gaurav");
9.     System.out.println("Initial  list  of  elements:  "+map);
10.    //key-based  removal
11.    map.remove(100);
12.    System.out.println("Updated  list  of  elements:  "+map);
13.    //value-based  removal
14.    map.remove(101);
15.    System.out.println("Updated  list  of  elements:  "+map);
16.    //key-value  pair  based  removal
17.    map.remove(102,  "Rahul");
18.    System.out.println("Updated  list  of  elements:  "+map);
19.   }
20.}
```

Output: nitial list of elements: {100=Amit, 101=Vijay, 102=Rahul, 103=Gaurav}

Updated list of elements: {101=Vijay, 102=Rahul, 103=Gaurav}

Updated list of elements: {102=Rahul, 103=Gaurav}

Updated list of elements: {103=Gaurav}

Java HashMap example to replace() elements

Here, we see different ways to replace elements.

```
1. import  java.util.*;
2. class  HashMap3{
3.  public  static  void  main(String  args[]){
4.   HashMap<Integer,String>  hm=new  HashMap<Integer,String>();
5.      hm.put(100,"Amit");
6.      hm.put(101,"Vijay");
7.      hm.put(102,"Rahul");
8.      System.out.println("Initial  list  of  elements:");
9.     for(Map.Entry  m:hm.entrySet())
10.    {
11.      System.out.println(m.getKey()+"  "+m.getValue());
12.    }
13.    System.out.println("Updated  list  of  elements:");
14.    hm.replace(102,  "Gaurav");
15.    for(Map.Entry  m:hm.entrySet())
```

```
16.    {
17.      System.out.println(m.getKey()+" "+m.getValue());
18.    }
19.    System.out.println("Updated list of elements:");
20.    hm.replace(101, "Vijay", "Ravi");
21.    for(Map.Entry m:hm.entrySet())
22.    {
23.      System.out.println(m.getKey()+" "+m.getValue());
24.    }
25.    System.out.println("Updated list of elements:");
26.    hm.replaceAll((k,v) -> "Ajay");
27.    for(Map.Entry m:hm.entrySet())
28.    {
29.      System.out.println(m.getKey()+" "+m.getValue());
30.    }
31. }
32.}
```

```
Initial list of elements:
100 Amit
101 Vijay
102 Rahul
Updated list of elements:
100 Amit
101 Vijay
102 Gaurav
Updated list of elements:
100 Amit
101 Ravi
102 Gaurav
Updated list of elements:
100 Ajay
101 Ajay
102 Ajay
```

Difference between HashSet and HashMap

HashSet contains only values whereas HashMap contains an entry(key and value).

Java HashMap Example: Book

```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.    this.id = id;
8.    this.name = name;
9.    this.author = author;
10.   this.publisher = publisher;
11.   this.quantity = quantity;
12.}
13.}
14.public class MapExample {
15.public static void main(String[] args) {
16.   //Creating map of Books
17.   Map<Integer,Book> map=new HashMap<Integer,Book>();
18.   //Creating Books
19.   Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.   Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Gra
   w Hill",4);
21.   Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.   //Adding Books to map
23.   map.put(1,b1);
24.   map.put(2,b2);
25.   map.put(3,b3);
26.
27.   //Traversing map
28.   for(Map.Entry<Integer, Book> entry:map.entrySet()){
29.      int key=entry.getKey();
30.      Book b=entry.getValue();
31.      System.out.println(key+" Details:");
32.      System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity
   );
33.   }
34.}
35.}
```

Test it Now

Output:

```
1 Details:
101 Let us C Yashwant Kanetkar BPB 8
2 Details:
102 Data Communications and Networking Forouzan Mc Graw Hill 4
3 Details:
103 Operating System Galvin Wiley 6
```

What is Hashing

It is the process of converting an object into an integer value. The integer value helps in indexing and faster searches.

What is HashMap

HashMap is a part of the Java collection framework. 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, and the node is represented as a class. It uses an array and LinkedList data structure internally for storing Key and Value. There are four fields in HashMap.

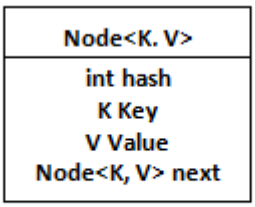


Figure: Representation of a Node

Before understanding the internal working of HashMap, you must be aware of hashCode() and equals() method.

- equals(): It checks the equality of two objects. It compares the Key, whether they are equal or not. It is a method of the Object class. It can be overridden. If you override the equals() method, then it is mandatory to override the hashCode() method.

- hashCode(): This is the method of the object class. 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. Hash code of null Key is 0.

❍Buckets: Array of the node is called buckets. Each node has a data structure like a LinkedList. More than one node can share the same bucket. It may be different in capacity.

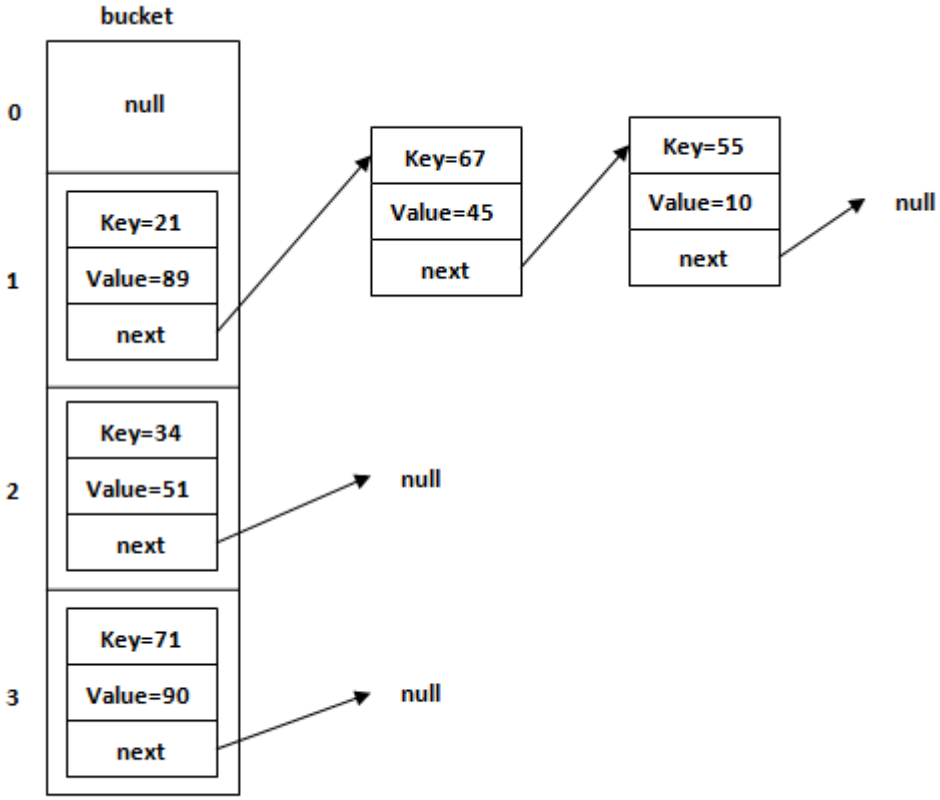


Figure: Allocation of nodes in Bucket

Insert Key, Value pair in HashMap

We use put() method to insert the Key and Value pair in the HashMap. The default size of HashMap is 16 (0 to 15).

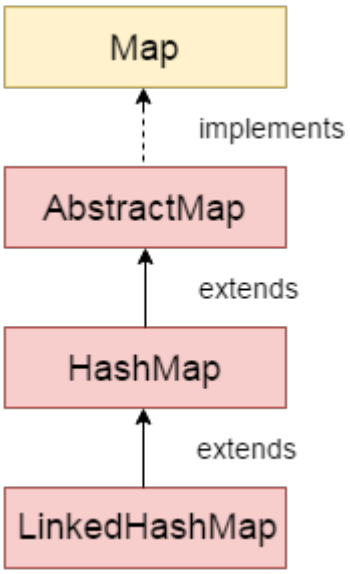
Example

In the following example, we want to insert three (Key, Value) pair in the HashMap.

```
1. HashMap<String, Integer> map = new HashMap<>();
2. map.put("Aman", 19);
3. map.put("Sunny", 29);
4. map.put("Ritesh", 39);
```

Let's see at which index the Key, value pair will be saved into HashMap. When we call the put() method, then it calculates the hash code of the Key "Aman." Suppose the hash code of "Aman" is 2657860. To store the Key in memory, we have to calculate the index.

Java LinkedHashMap class



Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

Points to remember

- Java LinkedHashMap contains values based on the key.
- Java LinkedHashMap contains unique elements.
- Java LinkedHashMap may have one null key and multiple null values.
- Java LinkedHashMap is non synchronized.
- Java LinkedHashMap maintains insertion order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Method	Description
V get(Object key)	It returns the value to which the specified key is mapped.
void clear()	It removes all the key-value pairs from a map.
boolean containsValue(Object value)	It returns true if the map maps one or more keys to the specified value.
Set<Map.Entry<K,V>> entrySet()	It returns a Set view of the mappings contained in the map.

void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped or defaultValue if this map contains no mapping for the key.
Set<K> keySet()	It returns a Set view of the keys contained in the map
protected boolean removeEldestEntry(Map.Entry< K,V> eldest)	It returns true on removing its eldest entry.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection<V> values()	It returns a Collection view of the values contained in this map.

Java LinkedHashMap Example

```
1. import  java.util.*;
2. class  LinkedHashMap1{
3.  public  static  void  main(String  args[]){
4.
5.   LinkedHashMap<Integer,String>  hm=new  LinkedHashMap<Integer,String>();
6.
7.   hm.put(100,"Amit");
8.   hm.put(101,"Vijay");
9.   hm.put(102,"Rahul");
10.
11.for(Map.Entry  m:hm.entrySet()){
12.  System.out.println(m.getKey()+"  "+m.getValue());
13.  }
14. }
```

```
15.}
```

```
Output:100 Amit
       101 Vijay
       102 Rahul
```

Java LinkedHashMap Example: Key-Value pair

```
1. import java.util.*;
2. class LinkedHashMap2{
3.  public static void main(String args[]){
4.   LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>();

5.    map.put(100,"Amit");
6.    map.put(101,"Vijay");
7.    map.put(102,"Rahul");
8.     //Fetching key
9.     System.out.println("Keys: "+map.keySet());
10.    //Fetching value
11.    System.out.println("Values: "+map.values());
12.    //Fetching key-value pair
13.    System.out.println("Key-Value pairs: "+map.entrySet());
14. }
15.}
```

```
Keys: [100, 101, 102]
Values: [Amit, Vijay, Rahul]
Key-Value pairs: [100=Amit, 101=Vijay, 102=Rahul]
```

Java LinkedHashMap Example:remove()

```
1. import java.util.*;
2. public class LinkedHashMap3 {
3.   public static void main(String args[]) {
4.    Map<Integer,String> map=new LinkedHashMap<Integer,String>();
5.    map.put(101,"Amit");
6.    map.put(102,"Vijay");
7.    map.put(103,"Rahul");
8.    System.out.println("Before invoking remove() method: "+map);
9.   map.remove(102);
10.  System.out.println("After invoking remove() method: "+map);
11.  }
```

```
12.}
```

Output:

```
Before invoking remove() method: {101=Amit, 102=Vijay, 103=Rahul}
After invoking remove() method: {101=Amit, 103=Rahul}
```

Java LinkedHashMap Example: Book

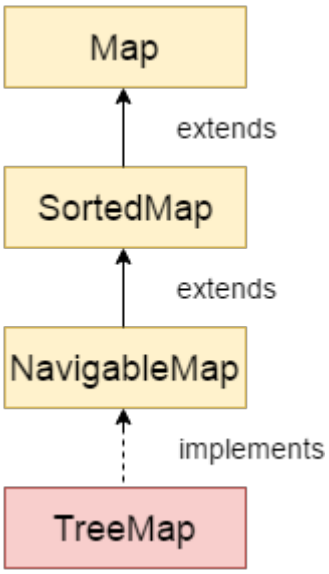
```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.    this.id = id;
8.    this.name = name;
9.    this.author = author;
10.   this.publisher = publisher;
11.   this.quantity = quantity;
12.}
13.}
14.public class MapExample {
15.public static void main(String[] args) {
16.   //Creating map of Books
17.   Map<Integer,Book> map=new LinkedHashMap<Integer,Book>();
18.   //Creating Books
19.   Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.   Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Gra
   w Hill",4);
21.   Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.   //Adding Books to map
23.   map.put(2,b2);
24.   map.put(1,b1);
25.   map.put(3,b3);
26.
27.   //Traversing map
28.   for(Map.Entry<Integer, Book> entry:map.entrySet()){
29.      int key=entry.getKey();
30.      Book b=entry.getValue();
31.      System.out.println(key+" Details:");
```

```
32.       System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity
    );
33.    }
34.}
35.}
```

Output:

```
2 Details:
102 Data Communications & Networking Forouzan Mc Graw Hill 4
1 Details:
101 Let us C Yashwant Kanetkar BPB 8
3 Details:
103 Operating System Galvin Wiley 6
```

Java TreeMap class



Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

- Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- Java TreeMap contains only unique elements.
- Java TreeMap cannot have a null key but can have multiple null values.
- Java TreeMap is non synchronized.

- Java TreeMap maintains ascending order.

TreeMap class declaration

Let's see the declaration for java.util.TreeMap class.

1. **public class** TreeMap<K,V> **extends** AbstractMap<K,V> **implements** NavigableMap <K,V>, Cloneable, Serializable

TreeMap class Parameters

Let's see the Parameters for java.util.TreeMap class.

- K: It is the type of keys maintained by this map.
- V: It is the type of mapped values.

Constructors of Java TreeMap class

Constructor	Description
TreeMap()	It is used to construct an empty tree map that will be sorted using the natural order of its key.
TreeMap(Comparator<? super K> comparator)	It is used to construct an empty tree-based map that will be sorted using the comparator comp.
TreeMap(Map<? extends K,? extends V> m)	It is used to initialize a treemap with the entries from m, which will be sorted using the natural order of the keys.
TreeMap(SortedMap<K,? extends V> m)	It is used to initialize a treemap with the entries from the SortedMap sm, which will be sorted in the same order as sm.

Methods of Java TreeMap class

Method	Description
Map.Entry<K,V> ceilingEntry(K key)	It returns the key-value pair having the least key, greater than or equal to the specified key, or null if there is no such key.
K ceilingKey(K key)	It returns the least key, greater than the specified key or null if there is no such key.
void clear()	It removes all the key-value pairs from a map.
Object clone()	It returns a shallow copy of TreeMap instance.

Comparator<? super K> comparator()	It returns the comparator that arranges the key in order, or null if the map uses the natural ordering.
NavigableSet<K> descendingKeySet()	It returns a reverse order NavigableSet view of the keys contained in the map.
NavigableMap<K,V> descendingMap()	It returns the specified key-value pairs in descending order.
Map.Entry firstEntry()	It returns the key-value pair having the least key.
Map.Entry<K,V> floorEntry(K key)	It returns the greatest key, less than or equal to the specified key, or null if there is no such key.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
SortedMap<K,V> headMap(K toKey)	It returns the key-value pairs whose keys are strictly less than toKey.
NavigableMap<K,V> headMap(K toKey, boolean inclusive)	It returns the key-value pairs whose keys are less than (or equal to if inclusive is true) toKey.
Map.Entry<K,V> higherEntry(K key)	It returns the least key strictly greater than the given key, or null if there is no such key.
K higherKey(K key)	It is used to return true if this map contains a mapping for the specified key.
Set keySet()	It returns the collection of keys exist in the map.
Map.Entry<K,V> lastEntry()	It returns the key-value pair having the greatest key, or null if there is no such key.
Map.Entry<K,V> lowerEntry(K key)	It returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key.
K lowerKey(K key)	It returns the greatest key strictly less than the given key, or null if there is no such key.

NavigableSet<K> navigableKeySet()	It returns a NavigableSet view of the keys contained in this map.
Map.Entry<K,V> pollFirstEntry()	It removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty.
Map.Entry<K,V> pollLastEntry()	It removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.
V put(K key, V value)	It inserts the specified value with the specified key in the map.
void putAll(Map<? extends K,? extends V> map)	It is used to copy all the key-value pair from one map to another map.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
NavigableMap<K,V> subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)	It returns key-value pairs whose keys range from fromKey to toKey.
SortedMap<K,V> subMap(K fromKey, K toKey)	It returns key-value pairs whose keys range from fromKey, inclusive, to toKey, exclusive.
SortedMap<K,V> tailMap(K fromKey)	It returns key-value pairs whose keys are greater than or equal to fromKey.
NavigableMap<K,V> tailMap(K fromKey, boolean inclusive)	It returns key-value pairs whose keys are greater than (or equal to, if inclusive is true) fromKey.
boolean containsKey(Object key)	It returns true if the map contains a mapping for the specified key.

boolean containsValue(Object value)	It returns true if the map maps one or more keys to the specified value.
K firstKey()	It is used to return the first (lowest) key currently in this sorted map.
V get(Object key)	It is used to return the value to which the map maps the specified key.
K lastKey()	It is used to return the last (highest) key currently in the sorted map.
V remove(Object key)	It removes the key-value pair of the specified key from the map.
Set<Map.Entry<K,V>> entrySet()	It returns a set view of the mappings contained in the map.
int size()	It returns the number of key-value pairs exists in the hashtable.
Collection values()	It returns a collection view of the values contained in the map.

Java TreeMap Example

```
1. import java.util.*;
2. class TreeMap1{
3.  public static void main(String args[]){
4.   TreeMap<Integer,String> map=new TreeMap<Integer,String>();
5.     map.put(100,"Amit");
6.     map.put(102,"Ravi");
7.     map.put(101,"Vijay");
8.     map.put(103,"Rahul");
9.
10.    for(Map.Entry m:map.entrySet()){
11.     System.out.println(m.getKey()+"  "+m.getValue());
12.    }
13. }
14.}
```

Output:100 Amit

```
101 Vijay
102 Ravi
103 Rahul
```

Java TreeMap Example: remove()

```
1. import java.util.*;
2. public class TreeMap2 {
3.   public static void main(String args[]) {
4.    TreeMap<Integer,String> map=new TreeMap<Integer,String>();
5.     map.put(100,"Amit");
6.     map.put(102,"Ravi");
7.     map.put(101,"Vijay");
8.     map.put(103,"Rahul");
9.     System.out.println("Before invoking remove() method");
10.    for(Map.Entry m:map.entrySet())
11.    {
12.       System.out.println(m.getKey()+" "+m.getValue());
13.    }
14.    map.remove(102);
15.    System.out.println("After invoking remove() method");
16.    for(Map.Entry m:map.entrySet())
17.    {
18.       System.out.println(m.getKey()+" "+m.getValue());
19.    }
20.    }
21.}
```

Output:

```
Before invoking remove() method
100 Amit
101 Vijay
102 Ravi
103 Rahul
After invoking remove() method
100 Amit
101 Vijay
103 Rahul
```

Java TreeMap Example: NavigableMap

```
1. import java.util.*;
2. class TreeMap3{
3.  public static void main(String args[]){
4.   NavigableMap<Integer,String> map=new TreeMap<Integer,String>();
5.     map.put(100,"Amit");
6.     map.put(102,"Ravi");
7.     map.put(101,"Vijay");
8.     map.put(103,"Rahul");
9.     //Maintains descending order
10.    System.out.println("descendingMap: "+map.descendingMap());
11.    //Returns key-value pairs whose keys are less than or equal to the specified key
    .
12.    System.out.println("headMap: "+map.headMap(102,true));
13.    //Returns key-value pairs whose keys are greater than or equal to the specified
   key.
14.    System.out.println("tailMap: "+map.tailMap(102,true));
15.    //Returns key-value pairs exists in between the specified key.
16.    System.out.println("subMap: "+map.subMap(100, false, 102, true));
17. }
18.}
```

```
descendingMap: {103=Rahul, 102=Ravi, 101=Vijay, 100=Amit}
headMap: {100=Amit, 101=Vijay, 102=Ravi}
tailMap: {102=Ravi, 103=Rahul}
subMap: {101=Vijay, 102=Ravi}
```

Java TreeMap Example: SortedMap

```
1. import java.util.*;
2. class TreeMap4{
3.  public static void main(String args[]){
4.   SortedMap<Integer,String> map=new TreeMap<Integer,String>();
5.     map.put(100,"Amit");
6.     map.put(102,"Ravi");
7.     map.put(101,"Vijay");
8.     map.put(103,"Rahul");
9.     //Returns key-value pairs whose keys are less than the specified key.
10.    System.out.println("headMap: "+map.headMap(102));
```

```
11.    //Returns key-value pairs whose keys are greater than or equal to the specified key.
12.    System.out.println("tailMap: "+map.tailMap(102));
13.    //Returns key-value pairs exists in between the specified key.
14.    System.out.println("subMap: "+map.subMap(100, 102));
15. }
16.}
```

```
headMap: {100=Amit, 101=Vijay}
tailMap: {102=Ravi, 103=Rahul}
subMap: {100=Amit, 101=Vijay}
```

What is difference between HashMap and TreeMap?

HashMap	TreeMap
1) HashMap can contain one null key.	TreeMap cannot contain any null key.
2) HashMap maintains no order.	TreeMap maintains ascending order.

Java TreeMap Example: Book

```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.    this.id = id;
8.    this.name = name;
9.    this.author = author;
10.   this.publisher = publisher;
11.   this.quantity = quantity;
12.}
13.}
14.public class MapExample {
15.public static void main(String[] args) {
16.   //Creating map of Books
17.   Map<Integer,Book> map=new TreeMap<Integer,Book>();
18.   //Creating Books
19.   Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
```

```
20.  Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Gra
     w Hill",4);
21.  Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.  //Adding Books to map
23.  map.put(2,b2);
24.  map.put(1,b1);
25.  map.put(3,b3);
26.
27.  //Traversing map
28.  for(Map.Entry<Integer, Book> entry:map.entrySet()){
29.     int key=entry.getKey();
30.     Book b=entry.getValue();
31.     System.out.println(key+" Details:");
32.     System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity
    );
33.  }
34.}
35.}
```

Output: 1 Details:

101 Let us C Yashwant Kanetkar BPB 8

2 Details:

102 Data Communications & Networking Forouzan Mc Graw Hill 4

3 Details:

103 Operating System Galvin Wiley 6

Difference between HashMap and Hashtable

HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.

But there are many differences between HashMap and Hashtable classes that are given below.

HashMap	Hashtable
1) HashMap is non synchronized. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is synchronized. It is thread-safe and can be shared with many threads.

2) HashMap allows one null key and multiple null values.	Hashtable doesn't allow any null key or value.
3) HashMap is a new class introduced in JDK 1.2.	Hashtable is a legacy class.
4) HashMap is fast.	Hashtable is slow.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is traversed by Iterator.	Hashtable is traversed by Enumerator and Iterator.
7) Iterator in HashMap is fail-fast.	Enumerator in Hashtable is not fail-fast.
8) HashMap inherits AbstractMap class.	Hashtable inherits Dictionary class.

Difference between Comparable and Comparator

Comparable and Comparator both are interfaces and can be used to sort collection elements.

However, there are many differences between Comparable and Comparator interfaces that are given below.

Comparable	Comparator
1) Comparable provides a single sorting sequence. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides multiple sorting sequences. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable affects the original class, i.e., the actual class is modified.	Comparator doesn't affect the original class, i.e., the actual class is not modified.
3) Comparable provides compareTo() method to sort elements.	Comparator provides compare() method to sort elements.

4) Comparable is present in java.lang package.	A Comparator is present in the java.util package.
5) We can sort the list elements of Comparable type by Collections.sort(List) method.	We can sort the list elements of Comparator type by Collections.sort(List, Comparator) method.

compareTo(Object obj) method

public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

We can sort the elements of:

1.String objects

2.Wrapper class objects

3.User-defined class objects

Java Comparable Example

Let's see the example of a Comparable interface that sorts the list elements on the basis of age.

File: TestSort3.java

```
1. //Java Program to demonstrate the use of Java Comparable.
2. //Creating a class which implements Comparable Interface
3. import java.util.*;
4. import java.io.*;
5. class Student implements Comparable<Student>{
6. int rollno;
7. String name;
8. int age;
```

```
9. Student(int rollno,String name,int age){
10.this.rollno=rollno;
11.this.name=name;
12.this.age=age;
13.}
14.public int compareTo(Student st){
15.if(age==st.age)
16.return 0;
17.else if(age>st.age)
18.return 1;
19.else
20.return -1;
21.}
22.}
23.//Creating a test class to sort the elements
24.public class TestSort3{
25.public static void main(String args[]){
26.ArrayList<Student> al=new ArrayList<Student>();
27.al.add(new Student(101,"Vijay",23));
28.al.add(new Student(106,"Ajay",27));
29.al.add(new Student(105,"Jai",21));
30.
31.Collections.sort(al);
32.for(Student st:al){
33.System.out.println(st.rollno+" "+st.name+" "+st.age);
34.}
35.}
36.}
```

Test it Now

Output:

```
105 Jai 21
101 Vijay 23
106 Ajay 27
```

Java Comparator Example

Let's see an example of the Java Comparator interface where we are sorting the elements of a list using different comparators.

Student.java

```
class Student{
int rollno;
String name;
int age;
Student(int rollno,String name,int age){
this.rollno=rollno;
this.name=name;
this.age=age;
}
}
```

AgeComparator.java

```
import java.util.*;
class AgeComparator implements Comparator<Student>{
public int compare(Student s1,Student s2){
if(s1.age==s2.age)
return 0;
else if(s1.age>s2.age)
return 1;
else
return -1;
}
}
```

NameComparator.java

This class provides comparison logic based on the name. In such case, we are using the compareTo() method of String class, which internally provides the comparison logic.

```
import java.util.*;
class NameComparator implements Comparator<Student>{
public int compare(Student s1,Student s2){
return s1.name.compareTo(s2.name);
}
}
```

TestComparator.java

In this class, we are printing the values of the object by sorting on the basis of name and age.

```
//Java Program to demonstrate the use of Java Comparator
import java.util.*;
import java.io.*;
```

```
4. class TestComparator{
5. public static void main(String args[]){
6. //Creating a list of students
7. ArrayList<Student> al=new ArrayList<Student>();
8. al.add(new Student(101,"Vijay",23));
9. al.add(new Student(106,"Ajay",27));
10.al.add(new Student(105,"Jai",21));
11.
12.System.out.println("Sorting by Name");
13.//Using NameComparator to sort the elements
14.Collections.sort(al,new NameComparator());
15.//Traversing the elements of list
16.for(Student st: al){
17.System.out.println(st.rollno+" "+st.name+" "+st.age);
18.}
19.
20.System.out.println("sorting by Age");
21.//Using AgeComparator to sort the elements
22.Collections.sort(al,new AgeComparator());
23.//Travering the list again
24.for(Student st: al){
25.System.out.println(st.rollno+" "+st.name+" "+st.age);
26.}
27.
28.}
29.}
```

Output:

```
Sorting by Name
106 Ajay 27
105 Jai 21
101 Vijay 23

Sorting by Age
105 Jai 21
101 Vijay 23
106 Ajay 27
```