**Polymorphism**

**Poly :-** many

**Morph:-** Forms

**Many Forms :-** The word polymorphism means having many forms.

**Ability To display Message in More Then One Form :-** We can define polymorphism as ability of message to display more then one form.

**Real Life Example :-** a person at same time can be father , husband , employee.

A same person possess different behavior at different time.

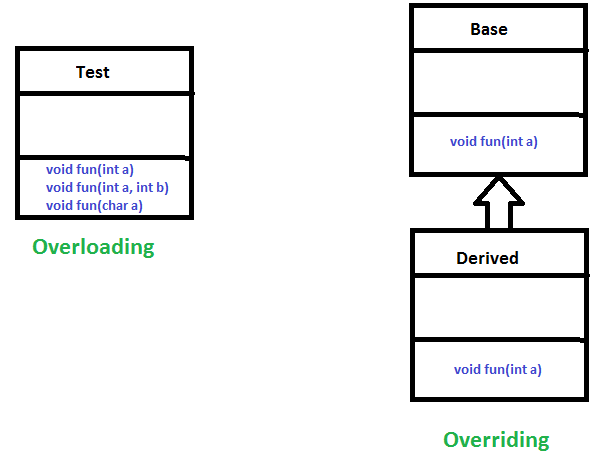
**VLC : - mp3 avi etc**

**Single Action Different Ways**

1. **Compile-time polymorphism:** It is also known as static polymorphism.

This type of polymorphism is achieved by function overloading or operator overloading.

But Java doesn’t support the Operator Overloading.

2 

**Method Overloading:** When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments.

**Runtime polymorphism:** It is also known as Dynamic Method Dispatch.

**Function Call Resolved At Runtime :-** It is a process in which a function call to the overridden method is resolved at Runtime.

This type of polymorphism is achieved by Method Overriding.

**Derived Class definition of base class :-** Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

**Exception**

**2 Exception Handling in Java**

**Abnormal Condition :-** Exception is an abnormal condition.

**Event disrupt Normal Flow** :- In Java, an exception is an event that disrupts the normal flow of the program.

**Object thrown at runtime**

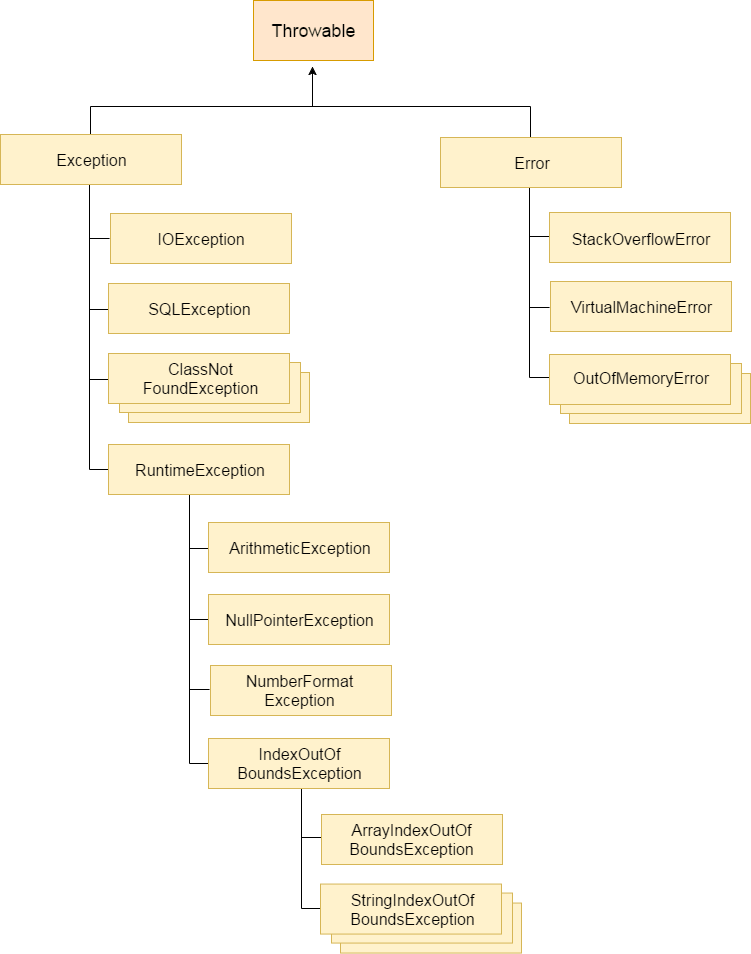
It is an object which is thrown at runtime.

**Mechanism To Handle Runtime Errors :-**

Exception Handling is a mechanism to handle runtime errors

ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**Hierarchy**



**Root class Throwable :-** The **java.lang.Throwable** class is the root class of Java Exception hierarchy.

****

**1) Checked Exception**

**Checked at Compile Time:-** Checked exceptions are checked at compile-time.

**Inherit Throwable exc Runtime Exception & Error** :- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions

**e.g**. IOException, SQLException etc.

**2) Unchecked Exception**

**Inherit Runtime Exception :-** The classes which inherit RuntimeException are known as unchecked exceptions

**e.g**. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

**Checked at Runtime:-** Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

**3) Error**

**Irrecoverable :-** Error is irrecoverable

**e.g**. OutOfMemoryError, VirtualMachineError, AssertionError etc.

**Java Exception Keywords**

There are 5 keywords which are used in handling exceptions in Java.

**Try: -**

**block** where we should put exception code**.**

**Followed by : -**

It is followed by catch and finally.

Can't use try **block alone**.

**Catch**

**Handle :-** Useful for handle Exception handling

**Followed By Finally :-**

**Cant use alone :-**

**Finally :-**

**Important code of program:-** it will execute either Exception occurs or not.

**Throw : -** The throw keyword used to **throw Exception.**

**Throws:-**

**As method signature**

**May Occur Exception:-**

* **Declare :'** The "throws" keyword is used to declare exceptions.
* It doesn't throw an exception.
* It specifies that there **may occur an exception** in the method.
* It is always used with method signature.

int a=50/0;//ArithmeticException

String s=null;

System.out.println(s.length());//NullPointerException

String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

**OLAP :-**

**Analytical Processing.**

OLAP is an acronym for Online Analytical Processing.

**Multidimension Analysis** OLAP performs multidimensional analysis of business data and

**Complex Calculations**

provides the capability for complex calculations, trend analysis, and sophisticated data modeling.

**What is OLAP? Cube, Operations & Types in Data Warehouse**

**Allow analyze information from different source :-** Online Analytical Processing (OLAP) is a category of software that allows users to analyze information from multiple database systems at the same time.

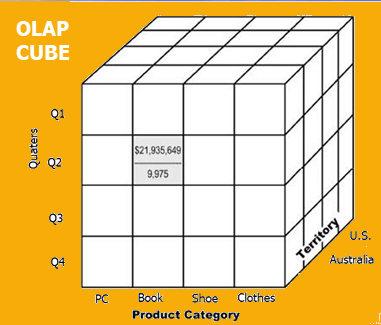
**Technology to extract/view from different point of view :-** It is a technology that enables analysts to extract and view business data from different points of view

**pre-calculated and pre-aggregated makes faster :-**

* Analysts frequently need to group, aggregate and join data.
* These operations in relational databases are resource intensive. With OLAP data can be pre-calculated and pre-aggregated, making analysis faster

**OLAP databases are divided into one or more cubes.**

* The cubes are designed in such a way that creating and viewing reports become easy.
* OLAP stands for Online Analytical Processing



**Hypercube**

* OLAP Cube is also called the hypercube.
* The OLAP cube is a **data structure** **optimized for very quick data analysis**.
* The OLAP Cube consists of numeric facts called measures which are categorized by dimensions.
* OLAP contains **multidimensional data**, with data usually obtained from a different and unrelated source.

**How does it work?**

**Extract Information:-** A Data warehouse would extract information from multiple data sources and formats like text files, excel sheet, multimedia files, etc.

**Cleaned And Transformed :-** The extracted data is cleaned and transformed. Data is loaded into an OLAP server (or OLAP cube) where information is pre-calculated in advance for further analysis.

**Basic analytical operations of OLAP**

Four types of analytical operations in OLAP are:

* Roll-up
* Drill-down
* Slice and dice
* Pivot (rotate)

**Difference between .equals and ==**

In general both equals() and “==” operator in Java are used to compare objects to check equality but here are some of the differences between the two:

**== Operator while equals method :-** Main difference between .equals() method and == operator is that one is method and other is operator.

**Address Comparison Vs Content Comparison :-** We can use == operators for reference comparison (**address comparison**) and .equals() method for **content comparison**. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.

**Override - == Method can not override while equal method can override**

**Primitive Data Type can apply cant apply for custom object.**

**HashMap in 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.
* One null key any number of null value.
* public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable
* HashMap(): It is the default constructor which creates an instance of HashMap with initial capacity 16 and load factor 0.75.
* **Unordered** insertion order not retain.
* **Put** method with same key override.
* **Remove -** Remove the element**.**
* **for (Map.Entry<String, Integer> e : map.entrySet()**

**System.out.println("Key: " + e.getKey()+ " Value: " + e.getValue());**

* **Internally HashMap contains an array of Node and a node is represented as a class which contains 4 fields:**
* **int hash**
* **K key**
* **V value**
* **Node next**

It internally store as a linked list.

Sign In

Home

Courses

Algorithms

Data Structures

Languages

Interview Corner

GATE

CS Subjects

Web Technologies

School Learning

Student

Jobs

GBlog

Puzzles

What's New ?

Change Language

Related Articles

▲

Related Articles

Basics

Flow Control

Strings

Arrays

Methods

Constructors

OOP Concepts

Exception Handling

Interfaces & Abstract Classes

Collections

Collections in Java

Collections Class in Java

Collection vs Collections in Java with Example

Java | Implementing Iterator and Iterable Interface

List Interface in Java with Examples

ArrayList in Java

Vector Class in Java

Stack Class in Java

LinkedList in Java

Queue Interface In Java

PriorityQueue in Java

Deque interface in Java with Example

ArrayDeque in Java

Set in Java

HashSet in Java

LinkedHashSet in Java with Examples

SortedSet Interface in Java with Examples

NavigableSet in Java with Examples

TreeSet in Java

Map Interface in Java

HashMap in Java with Examples

Hashtable in Java

LinkedHashMap in Java

SortedMap Interface in Java with Examples

TreeMap in Java

Multithreading

File Handling

HashMap in Java with Examples

Difficulty Level : Medium

Last Updated : 19 Nov, 2020

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 is similar to the HashTable, but it is unsynchronized. It allows to store the null keys as well, but there should be only one null key object and there can be any number of null values. This class makes no guarantees as to the order of the map. To use this class and its methods, you need to import java.util.HashMap package or its superclass.

Java

// Java program to illustrate

// Java.util.HashMap

import java.util.HashMap;

public class GFG {

public static void main(String[] args)

{

// Create an empty hash map

HashMap<String, Integer> map = new HashMap<>();

// Add elements to the map

map.put("vishal", 10);

map.put("sachin", 30);

map.put("vaibhav", 20);

// Print size and content

System.out.println("Size of map is:- "

+ map.size());

System.out.println(map);

// Check if a key is present and if

// present, print value

if (map.containsKey("vishal")) {

Integer a = map.get("vishal");

System.out.println("value for key"

+ " \"vishal\" is:- " + a);

}

}

}

Output

Size of map is:- 3

{vaibhav=20, vishal=10, sachin=30}

value for key "vishal" is:- 10

The Hierarchy of HashMap

Hierarchy of HashMap in Java

Declaration:

public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable

Type Parameters:

K – the type of keys maintained by this map

V – the type of mapped values

HashMap implements Serializable, Cloneable, Map<K, V> interfaces. HashMap extends AbstractMap<K, V> class. The direct subclasses are LinkedHashMap, PrinterStateReasons.

Constructors in HashMap:

HashMap provides 4 constructors and access modifier of each is public:

1. HashMap(): It is the default constructor which creates an instance of HashMap with initial capacity 16 and load factor 0.75.

HashMap<K, V> hm = new HashMap<K, V>();

Java

// Java program to demonstrate the HashMap() constructor

import java.io.\*;

import java.util.\*;

class AddElementsToHashMap {

public static void main(String args[])

{

// No need to mention the

// Generic type twice

HashMap<Integer, String> hm1 = new HashMap<>();

// Initialization of a HashMap

// using Generics

HashMap<Integer, String> hm2

= new HashMap<Integer, String>();

// Add Elements using put method

hm1.put(1, "one");

hm1.put(2, "two");

hm1.put(3, "three");

hm2.put(4, "four");

hm2.put(5, "five");

hm2.put(6, "six");

System.out.println("Mappings of HashMap hm1 are : "

+ hm1);

System.out.println("Mapping of HashMap hm2 are : "

+ hm2);

}

}

Output

Mappings of HashMap hm1 are : {1=one, 2=two, 3=three}

Mapping of HashMap hm2 are : {4=four, 5=five, 6=six}

2. HashMap(int initialCapacity): It creates a HashMap instance with specified initial capacity and load factor 0.75.

HashMap<K, V> hm = new HashMap<K, V>(int initialCapacity);

Java

// Java program to demonstrate the

// HashMap(int initialCapacity)

// constructor

import java.io.\*;

import java.util.\*;

class AddElementsToHashMap {

public static void main(String args[])

{

// No need to mention the

// Generic type twice

HashMap<Integer, String> hm1 = new HashMap<>(10);

// Initialization of a HashMap

// using Generics

HashMap<Integer, String> hm2

= new HashMap<Integer, String>(2);

// Add Elements using put method

hm1.put(1, "one");

hm1.put(2, "two");

hm1.put(3, "three");

hm2.put(4, "four");

hm2.put(5, "five");

hm2.put(6, "six");

System.out.println("Mappings of HashMap hm1 are : "

+ hm1);

System.out.println("Mapping of HashMap hm2 are : "

+ hm2);

}

}

Output

Mappings of HashMap hm1 are : {1=one, 2=two, 3=three}

Mapping of HashMap hm2 are : {4=four, 5=five, 6=six}

3. HashMap(int initialCapacity, float loadFactor): It creates a HashMap instance with specified initial capacity and specified load factor.

HashMap<K, V> hm = new HashMap<K, V>(int initialCapacity, int loadFactor);

Java

// Java program to demonstrate the HashMap(int initialCapacity,

// float loadFactor) Constructor

import java.io.\*;

import java.util.\*;

class AddElementsToHashMap {

public static void main(String args[])

{

// No need to mention the

// Generic type twice

HashMap<Integer, String> hm1

= new HashMap<>(5, 0.75f);

// Initialization of a HashMap

// using Generics

HashMap<Integer, String> hm2

= new HashMap<Integer, String>(3, 0.5f);

// Add Elements using put method

hm1.put(1, "one");

hm1.put(2, "two");

hm1.put(3, "three");

hm2.put(4, "four");

hm2.put(5, "five");

hm2.put(6, "six");

System.out.println("Mappings of HashMap hm1 are : "

+ hm1);

System.out.println("Mapping of HashMap hm2 are : "

+ hm2);

}

}

Output

Mappings of HashMap hm1 are : {1=one, 2=two, 3=three}

Mapping of HashMap hm2 are : {4=four, 5=five, 6=six}

4. HashMap(Map map): It creates an instance of HashMap with the same mappings as the specified map.

HashMap<K, V> hm = new HashMap<K, V>(Map map);

Java

// Java program to demonstrate the

// HashMap(Map map) Constructor

import java.io.\*;

import java.util.\*;

class AddElementsToHashMap {

public static void main(String args[])

{

// No need to mention the

// Generic type twice

Map<Integer, String> hm1 = new HashMap<>();

// Add Elements using put method

hm1.put(1, "one");

hm1.put(2, "two");

hm1.put(3, "three");

// Initialization of a HashMap

// using Generics

HashMap<Integer, String> hm2

= new HashMap<Integer, String>(hm1);

System.out.println("Mappings of HashMap hm1 are : "

+ hm1);

System.out.println("Mapping of HashMap hm2 are : "

+ hm2);

}

}

Output

Mappings of HashMap hm1 are : {1=one, 2=two, 3=three}

Mapping of HashMap hm2 are : {1=one, 2=two, 3=three}

Performing Various Operations on HashMap

1. Adding Elements: In order to add an element to the map, we can use the put() method. However, the insertion order is not retained in the Hashmap. Internally, for every element, a separate hash is generated and the elements are indexed based on this hash to make it more efficient.

Java

// Java program to add elements

// to the HashMap

import java.io.\*;

import java.util.\*;

class AddElementsToHashMap {

public static void main(String args[])

{

// No need to mention the

// Generic type twice

HashMap<Integer, String> hm1 = new HashMap<>();

// Initialization of a HashMap

// using Generics

HashMap<Integer, String> hm2

= new HashMap<Integer, String>();

// Add Elements using put method

hm1.put(1, "Geeks");

hm1.put(2, "For");

hm1.put(3, "Geeks");

hm2.put(1, "Geeks");

hm2.put(2, "For");

hm2.put(3, "Geeks");

System.out.println("Mappings of HashMap hm1 are : "

+ hm1);

System.out.println("Mapping of HashMap hm2 are : "

+ hm2);

}

}

Output

Mappings of HashMap hm1 are : {1=Geeks, 2=For, 3=Geeks}

Mapping of HashMap hm2 are : {1=Geeks, 2=For, 3=Geeks}

2. Changing Elements: After adding the elements if we wish to change the element, it can be done by again adding the element with the put() method. Since the elements in the map are indexed using the keys, the value of the key can be changed by simply inserting the updated value for the key for which we wish to change.

Java

// Java program to change

// elements of HashMap

import java.io.\*;

import java.util.\*;

class ChangeElementsOfHashMap {

public static void main(String args[])

{

// Initialization of a HashMap

HashMap<Integer, String> hm

= new HashMap<Integer, String>();

// Change Value using put method

hm.put(1, "Geeks");

hm.put(2, "Geeks");

hm.put(3, "Geeks");

System.out.println("Initial Map " + hm);

hm.put(2, "For");

System.out.println("Updated Map " + hm);

}

}

Output

Initial Map {1=Geeks, 2=Geeks, 3=Geeks}

Updated Map {1=Geeks, 2=For, 3=Geeks}

3. Removing Element: In order to remove an element from the Map, we can use the remove() method. This method takes the key value and removes the mapping for a key from this map if it is present in the map.

Java

// Java program to remove

// elements from HashMap

import java.io.\*;

import java.util.\*;

class RemoveElementsOfHashMap{

public static void main(String args[])

{

// Initialization of a HashMap

Map<Integer, String> hm

= new HashMap<Integer, String>();

// Add elements using put method

hm.put(1, "Geeks");

hm.put(2, "For");

hm.put(3, "Geeks");

hm.put(4, "For");

// Initial HashMap

System.out.println("Mappings of HashMap are : "

+ hm);

// remove element with a key

// using remove method

hm.remove(4);

// Final HashMap

System.out.println("Mappings after removal are : "

+ hm);

}

}

Output

Mappings of HashMap are : {1=Geeks, 2=For, 3=Geeks, 4=For}

Mappings after removal are : {1=Geeks, 2=For, 3=Geeks}

4. Traversal of HashMap

We can use the Iterator interface to traverse over any structure of the Collection Framework. Since Iterators work with one type of data we use .Entry< ? , ? > to resolve the two separate types into a compatible format. Then using the next() method we print the entries of HashMap.

Java

// Java program to traversal a

// Java.util.HashMap

import java.util.HashMap;

import java.util.Map;

public class TraversalTheHashMap {

public static void main(String[] args)

{

// initialize a HashMap

HashMap<String, Integer> map = new HashMap<>();

// Add elements using put method

map.put("vishal", 10);

map.put("sachin", 30);

map.put("vaibhav", 20);

// Iterate the map using

// for-each loop

for (Map.Entry<String, Integer> e : map.entrySet())

System.out.println("Key: " + e.getKey()

+ " Value: " + e.getValue());

}

}

Output

Key: vaibhav Value: 20

Key: vishal Value: 10

Key: sachin Value: 30

Important Features of HashMap

To access a value one must know its key. HashMap is known as HashMap because it uses a technique called Hashing. Hashing is a technique of converting a large String to small String that represents the same String. A shorter value helps in indexing and faster searches. HashSet also uses HashMap internally.

Few important features of HashMap are:

HashMap is a part of java.util package.

HashMap extends an abstract class AbstractMap which also provides an incomplete implementation of Map interface.

It also implements

Cloneable

and

Serializable

interface. K and V in the above definition represent Key and Value respectively.

HashMap doesn’t allow duplicate keys but allows duplicate values. That means A single key can’t contain more than 1 value but more than 1 key can contain a single value.

HashMap allows null key also but only once and multiple null values.

This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time. It is roughly similar to HashTable but is unsynchronized.

Internal Structure of HashMap

Internally HashMap contains an array of Node and a node is represented as a class which contains 4 fields:

int hash

K key

V value

Node next

It can be seen that node is containing a reference of its own object. So it’s a linked list.

HashMap:

array

Node:

node\_hash\_map

Performance of HashMap

Performance of HashMap depends on 2 parameters:

Initial Capacity

Load Factor

Initial Capacity – It is the capacity of HashMap at the time of its creation (It is the number of buckets a HashMap can hold when the HashMap is instantiated). In java, it is 2^4=16 initially, meaning it can hold 16 key-value pairs.

Load Factor – It is the percent value of the capacity after which the capacity of Hashmap is to be increased (It is the percentage fill of buckets after which Rehashing takes place). In java, it is 0.75f by default, meaning the rehashing takes place after filling 75% of the capacity.

Threshold – It is the product of Load Factor and Initial Capacity. In java, by default, it is (16 \* 0.75 = 12). That is, Rehashing takes place after inserting 12 key-value pairs into the HashMap.

Rehashing – It is the process of doubling the capacity of the HashMap after it reaches its Threshold. In java, HashMap continues to rehash(by default) in the following sequence – 2^4, 2^5, 2^6, 2^7, …. so on.

From Java 8 onward, Java has started using Self Balancing BST instead of a linked list for chaining.

Hashing

Hashing is a process of converting an object into integer form by using the method hashCode(). Its necessary to write hashCode() method properly for better performance

HashMap contains an array of Node and Node can represent a class having following objects

* int hash
* K key
* V value
* Node next

Hashing

* **Hashing** is a process of converting an object into integer form by using the method hashCode().
* Its necessary to write hashCode() method properly for better performance of HashMap.
* Here I am taking key of my own class so that I can override hashCode() method to show different scenarios

hashCode() method

**Hashcode**

hashCode() method is used to get the hash Code of an object. hashCode() method of object class returns the memory reference of object in integer form. Definition of hashCode() method is public native hashCode()

**Buckets**

A bucket is one element of HashMap array. It is used to store nodes. Two or more nodes can have the same bucket. In that case link list structure is used to connect the nodes. Buckets are different in capacity. A relation between bucket and capacity is as follows

Go To TopJavatpoint Logo

Home

Java

Programs

OOPs

String

Regex

Exception

Multithreading

Collections

JDBC

Swing

JavaFX

Servlet

JSP

Spring

Spring Boot

Projects

Interview Questions

Working of HashMap in Java

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.

ADVERTISEMENT

Working of HashMap in Java

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

ADVERTISEMENT

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.

Working of HashMap in Java

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).

ADVERTISEMENT

Example

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

HashMap<String, Integer> map = new HashMap<>();

map.put("Aman", 19);

map.put("Sunny", 29);

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.

Calculating Index

ADVERTISEMENT

Index minimizes the size of the array. The Formula for calculating the index is:

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

Where n is the size of the array. Hence the index value for "Aman" is:

Index = 2657860 & (16-1) = 4

The value 4 is the computed index value where the Key and value will store in HashMap.

Working of HashMap in Java

Hash Collision

This is the case when the calculated index value is the same for two or more Keys. Let's calculate the hash code for another Key "Sunny." Suppose the hash code for "Sunny" is 63281940. To store the Key in the memory, we have to calculate index by using the index formula.

ADVERTISEMENT

Index=63281940 & (16-1) = 4

The value 4 is the computed index value where the Key will be stored in HashMap. In this case, equals() method check that both Keys are equal or not. If Keys are same, replace the value with the current value. Otherwise, connect this node object to the existing node object through the LinkedList. Hence both Keys will be stored at index 4.

Working of HashMap in Java

Similarly, we will store the Key "Ritesh." Suppose hash code for the Key is 2349873. The index value will be 1. Hence this Key will be stored at index 1.

ADVERTISEMENT

Working of HashMap in Java

get() method in HashMap

get() method is used to get the value by its Key. It will not fetch the value if you don't know the Key. When get(K Key) method is called, it calculates the hash code of the Key.

Suppose we have to fetch the Key "Aman." The following method will be called.

map.get(new Key("Aman"));

It generates the hash code as 2657860. Now calculate the index value of 2657860 by using index formula. The index value will be 4, as we have calculated above. get() method search for the index value 4. It compares the first element Key with the given Key. If both keys are equal, then it returns the value else check for the next element in the node if it exists. In our scenario, it is found as the first element of the node and return the value 19.

ADVERTISEMENT

Let's fetch another Key "Sunny."

The hash code of the Key "Sunny" is 63281940. The calculated index value of 63281940 is 4, as we have calculated for put() method. Go to index 4 of the array and compare the first element's Key with the given Key. It also compares Keys. In our scenario, the given Key is the second element, and the next of the node is null. It compares the second element Key with the specified Key and returns the value 29. It returns null if the next of the node is null.

← PrevNext →

Youtube For Videos Join Our Youtube Channel: Join Now

Help Others, Please Share

facebook twitter pinterest

Learn Latest Tutorials

SoapUI tutorial

SoapUI

RPA tutorial

RPA

manual testing tutorial

Manual T.

cucumber tutorial

Cucumber

Appium tutorial

Appium

postgresql tutorial

PostgreSQL

Apache Solr Tutorial

Solr

MongoDB tutorial

MongoDB

Gimp Tutorial

Gimp

Verilog Tutorial

Verilog

Teradata Tutorial

Teradata

PhoneGap Tutorial

PhoneGap

Preparation

Aptitude

Aptitude

Logical Reasoning

Reasoning

Verbal Ability

Verbal A.

Interview Questions

Interview

Company Interview Questions

Company

Trending Technologies

Artificial Intelligence Tutorial

AI

AWS Tutorial

AWS

Selenium tutorial

Selenium

Cloud tutorial

Cloud

Hadoop tutorial

Hadoop

ReactJS Tutorial

ReactJS

Data Science Tutorial

D. Science

Angular 7 Tutorial

Angular 7

Blockchain Tutorial

Blockchain

Git Tutorial

Git

Machine Learning Tutorial

ML

DevOps Tutorial

DevOps

B.Tech / MCA

DBMS tutorial

DBMS

Data Structures tutorial

DS

DAA tutorial

DAA

Operating System tutorial

OS

Computer Network tutorial

C. Network

Compiler Design tutorial

Compiler D.

Computer Organization and Architecture

COA

Discrete Mathematics Tutorial

D. Math.

Ethical Hacking Tutorial

E. Hacking

Computer Graphics Tutorial

C. Graphics

Software Engineering Tutorial

Software E.

html tutorial

Web Tech.

Cyber Security tutorial

Cyber Sec.

Automata Tutorial

Automata

C Language tutorial

C

C++ tutorial

C++

Java tutorial

Java

.Net Framework tutorial

.Net

Python tutorial

Python

List of Programs

Programs

Control Systems tutorial

Control S.

Data Mining Tutorial

Data Mining

ADVERTISEMENT

ADVERTISEMENT

ADVERTISEMENT

ADVERTISEMENT

Javatpoint Services

JavaTpoint offers too many high quality services. Mail us on hr@javatpoint.com, to get more information about given services.

Website Designing

Website Development

Java Development

PHP Development

WordPress

Graphic Designing

Logo

Digital Marketing

On Page and Off Page SEO

PPC

Content Development

Corporate Training

Classroom and Online Training

Data Entry

Training For College Campus

JavaTpoint offers college campus training on Core Java, Advance Java, .Net, Android, Hadoop, PHP, Web Technology and Python. Please mail your requirement at hr@javatpoint.com.

Duration: 1 week to 2 week

Like/Subscribe us for latest updates or newsletter RSS Feed Subscribe to Get Email Alerts Facebook Page Twitter Page YouTube Blog Page

LEARN TUTORIALS

Learn Java

Learn Data Structures

Learn C Programming

Learn C++ Tutorial

Learn C# Tutorial

Learn PHP Tutorial

Learn HTML Tutorial

Learn JavaScript Tutorial

Learn jQuery Tutorial

Learn Spring Tutorial

OUR WEBSITES

Javatpoint.com

Hindi100.com

Lyricsia.com

Quoteperson.com

Jobandplacement.com

OUR SERVICES

Website Development

Android Development

Website Designing

Digital Marketing

Summer Training

Industrial Training

College Campus Training

CONTACT

Address: G-13, 2nd Floor, Sec-3

Noida, UP, 201301, India

Contact