Object Oriented Programming

1. Abstraction :

- Let's suppose you want to turn on the bulb in your room. What do you do to switch on the bulb. You simply press the button and the light bulb turns on. Right? Notice that here you're only concerned with your final result, i.e., turning on the light bulb. You do not care about the circuit of the bulb or how current flows through the bulb. The point here is that you press the switch, the bulb turns on! You don't know how the bulb turned on/how the circuit is made because all these details are hidden from you. This phenomenon is known as abstraction.
- More formally, data abstraction is the way through which only the essential info is shown to the user, and all the internal details remain hidden from the user.

Example :

→ Use this phone without bothering about how it was made

2. Polymorphism :

One entity many forms.

- The word polymorphism comprises two words, poly which means many, and morph, which means forms.
- In OOPs, polymorphism is the property that helps to perform a single task in different ways.
- Let us consider a real-life example of polymorphism. A woman at the same time can be a mother, wife, sister, daughter, etc. Here, a woman is an entity having different forms.
- Let's take another example, a smartphone can work like a camera as well as like a calculator. So, you can see the a smartphone is an entity having different forms. Also :

Laptop is a single entity with Wifi + Speaker + storage in a single box!

3. Encapsulation :

- The act of putting various components together (in a capsule).
- In java, the variables and methods are the components that are wrapped inside a single unit named class.
- All the methods and variables of a class remain hidden from any other class.
- A automatic cold drink vending machine is an example of encapsulation.
- Cold drinks inside the machine are data that is wrapped inside a single unit cold drink vending machine.

4. Inheritance :

The act of deriving new things from existing things.

- In Java, one class can acquire all the properties and behaviours of other some other class
- The class which inherits some other class is known as child class or sub class.
- The class which is inherited is known as parent class or super class.
- Inheritance helps us to write more efficient code because it increases the re-usablity of the code.

Example :

Rickshaw → E-Rickshaw

Phone → Smart Phone

Class & Objects

Class is a user-defined datatype that has its own data members and member functions

object is an instance of class by which we can access the data members and member functions of the class.

```
class <class_name>{
    field;
    method;
```

Access modifiers, getters & setters in Java

Access Modifiers specify where a property/method is accessible. There are four types of access modifiers in java

- private
- default
- protected
- public

Access Modifier	within class	within package	outside package by subclass only	outside package
public	Y	Y	Y	Y
protected	Y	Y	Y	N
Default	Y	Y	N	N
private	Y	N	N	N

Constructors in Java :

- Constructors are similar to methods,, but they are used to initialize an object.
- Constructors do not have any return type(not even void).
- Every time we create an object by using the new() keyword, a constructor is called.
- If we do not create a constructor by ourself, then the default constructor(created by Java compiler) is called.

Rules for creating a Constructor

- The class name and constructor name should be the same.
- It must have no explicit return type.
- It can not be abstract, static, final, and synchronized.

Types of Constructors in Java :

There are two types of constructors in Java :

Defaut constructor : A constructor with 0 parameters is known as default constructor.

Syntax :

```
<class_name>(){
//code to be executed on the execution of the constructor
}
```

Paramerterized constructor : A constructor with some specified number of parameters is known as a parameterized constructor.

Syntax :

```
<class-name>(<data-type> param1, <data-type> param2,.....){
//code to be executed on the invocation of the constructor
}
```

Constructor Overloading in Java :

Just like methods, constructors can also be overloaded in Java. We can overload the Employee constructor like below:

Note:

- Constructors can take parameters without being overloaded
- There can be more than two overloaded constructors

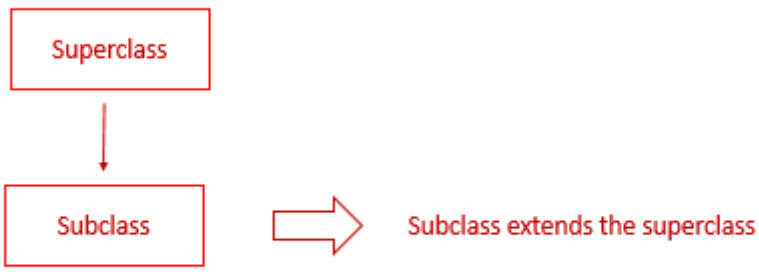
Inheritance in Java

- You might have heard people saying your nose is similar to your father or mother. Or, more formally, we can say that you've inherited the genes from your parents due to which you look similar to them.
- The same phenomenon of inheritance is also valid in programming.
- In Java, one class can easily inherit the attributes and methods from some other class. This mechanism of acquiring objects and properties from some other class is known as inheritance in Java.
- Inheritance is used to borrow properties & methods from an existing class.
- Inheritance helps us create classes based on existing classes, which increases the code's reusability.

Examples :

Important terminologies used in Inheritance :

- ✓ Parent class/superclass: The class from which a class inherits methods and attributes is known as parent class.
- ✓ Child class/sub-class: The class that inherits some other class's methods and attributes is known as child class.



Extends keyword in inheritance :

The extends keyword is used to inherit a subclass from a superclass.

Syntax :

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

this and super keyword in Java

- **this keyword in Java :**
- **this** is a way for us to reference an object of the class which is being created/referenced.
- It is used to call the default constructor of the same class.
- **this** keyword eliminates the confusion between the parameters and the class attributes with the same name. Take a look at the example given below :

```
class ark{
    int x;

//    getter of x
    public  int getX(){
        return  x;
    }

    // Constructor with a parameter
    ark(int x) {
```

```
        x = x;
    }

    // Call the constructor
    public static void main(String[] args) {
        ark obj1 = new ark(65);
        System.out.println(obj1.getX());

    }
}
```

Output : 0

```
class ark{
   int x;


//    getter of x
   public  int getX(){
       return  x;
   }


   // Constructor with a parameter
   ark(int x) {
```

```
        this.x = x;
    }


    // Call the constructor
    public static void main(String[] args) {
        ark obj1 = new ark(65);
        System.out.println(obj1.getX());


    }
}
```

Output : 65

Super keyword

- A reference variable used to refer immediate parent class object.
- It can be used to refer immediate parent class instance variable.
- It can be used to invoke the parent class method.

Method Overriding in Java

Method Overriding in Java:

- If the child class implements the same method present in the parent class again, it is know as method overriding.
- Method overriding helps us to classify a behaviour that is specific to the child class.
- The subclass can override the method of the parent class only when the method is not declared as final.
- Example :
- In the below code, we've created two classes: class A & class B.
- Class B is inheriting class A.

- In the main() method, we've created one object for both classes. We're running the meth1() method on class A and B objects separately, but the output is the same because the meth1() is defined in the parent class, i.e., class A.

Method Overloading in Java

In java, we do method overloading in two ways: –

1. By changing the number of parameters.
2. By changing data types.

- Change the number of arguments:

In the example below, we have two methods, the first method has two arguments, and the second method has three arguments.

```
class Demo
{
 void multiply(int a, int b)
 {
  System.out.printIn("Result is"+(a*b)) ;
 }
 void multiply(int a, int b,int c)
 {
  System.out.printIn("Result is"+(a*b*c));
 }
 public static void main(String[] args)
 {
  Demo obj = new Demo();
  obj.multiply(8,5);
  obj.multiply(4,6,2);
 }
}
```

Output:-

Result is 40

Abstract Class & Abstract Methods

What does Abstract mean?

Abstract in English means existing in through or as an idea without concrete existence.

Abstract class :

- An abstract class cannot be instantiated.
- Java requires us to extend it if we want to access it.
- It can include abstract and non-abstract methods.
- If a class includes abstract methods, then the class itself must be declared abstract, as in:
- Abstract class are used when we want to achieve security & abstraction(hide certain details & show only necessary details to the user

Abstract method :

- A method that is declared without implementation is known as the abstract method.
- An abstract method can only be used inside an abstract class.
- The body of the abstract method is provided by the class that inherits the abstract class in which the abstract method is present.
- In the above example, on() is the abstract method.

Interfaces in Java :

- Just like a class in java is a collection of the related methods, an interface in java is a collection of abstract methods.
- The interface is one more way to achieve abstraction in Java.
- An interface may also contain constants, default methods, and static methods.
- All the methods inside an interface must have empty bodies except default methods and static methods.
- We use the **interface** keyword to declare an interface.
- There is no need to write **abstract** keyword before declaring methods in an interface because an interface is implicitly abstract.
- An interface cannot contain a constructor (as it cannot be used to create objects)
- In order to implement an interface, java requires a class to use the **implement** keyword.

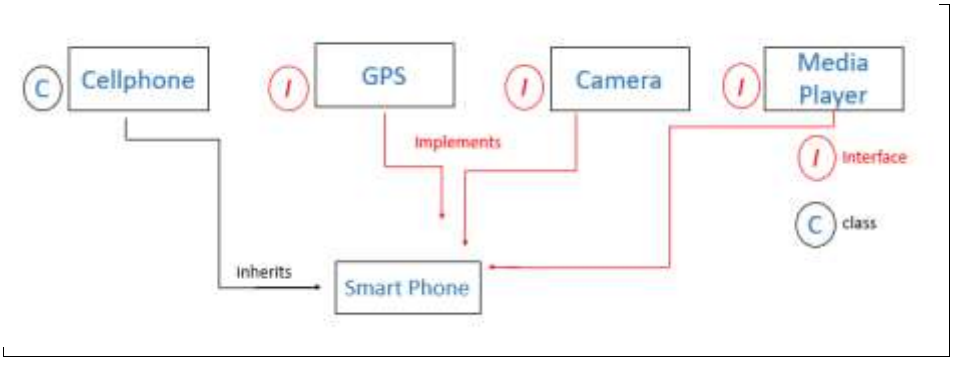
Why multiple inheritance is not supported in java?

- Is multiple inheritance allowed in Java?
- Multiple inheritance faces problems when there exists a method with the same signature in both the superclasses.
- Due to such a problem, java does not support multiple inheritance directly, but the similar concept can be achieved using interfaces.
- A class can implement multiple interfaces and extend a class at the same time.

Some Important points :

- Interfaces in java are a bit like the class but with a significantly different.
- An Interface can only have method signatures field and a default method.
- The class implementing an interface needs to declare the methods (not field)
- You can create a reference of an interface but not the object
- Interface methods are public by default

Polymorphism in Interfaces

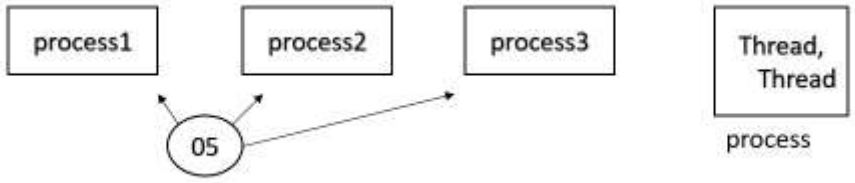


GPS g = new Smartphone ( ); can only use GPS method

Smartphones = new Smartphone ( ); can only use smartphone methods

Multithreading in Java

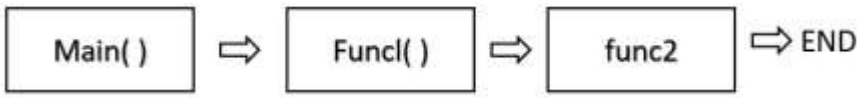
Multiprocessing and multithreading both are used to achive multitasking



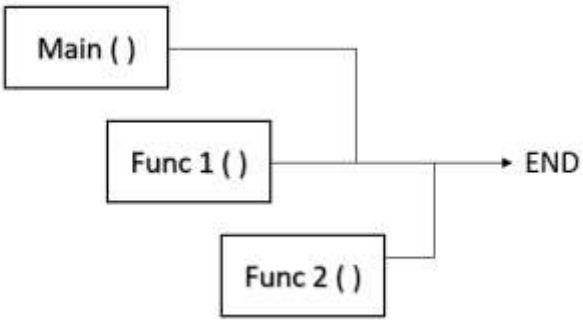
- threads use shared memory area
- threads = faster content switching
- A thread is light-weight where a process is heavyweight

Flow Control in Java

Without threading :



With threading :



Creating a Threading

There are two ways to create a thread in java

- By extending thread class
- By implementing Runnable interface

Creating a Java Thread Using Runnable Interface

In the previous tutorial, I told you that there are two ways to create a thread in java :

1. By Extending Thread Class
2. By implementing Runnable interface

Steps To Create A Java Thread Using Runnable Interface:

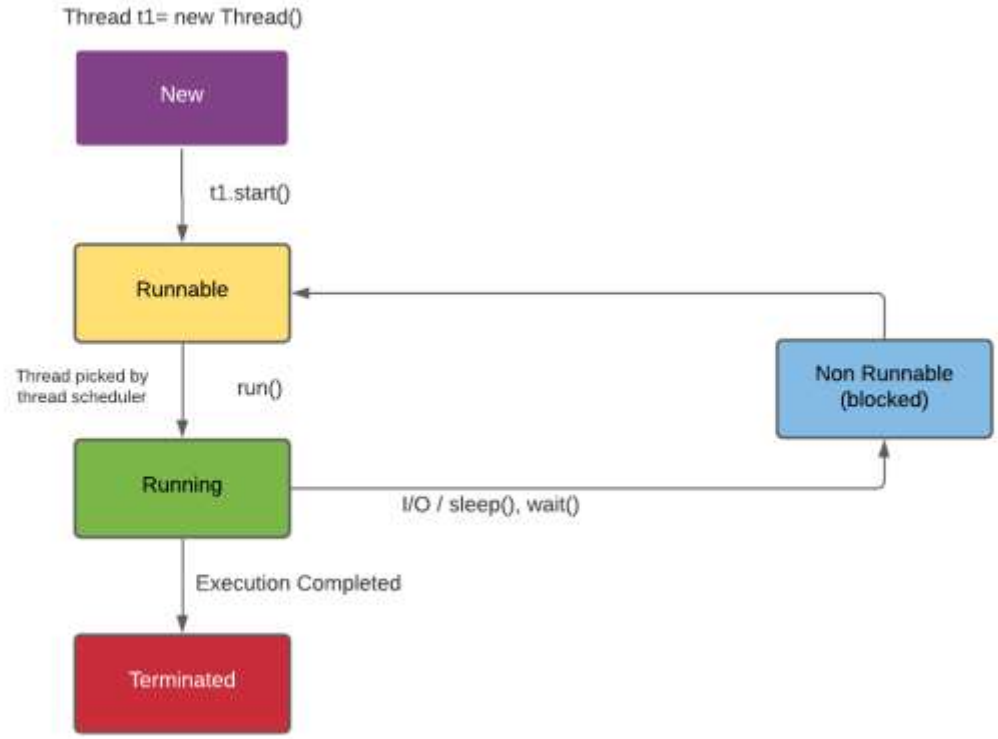
- Create a class and implement the Runnable interface by using the implements keyword.
- Override the run() method inside the implementer class.
- Create an object of the implementer class in the main() method.
- Instantiate the Thread class and pass the object to the Thread constructor.
- Call start() on the thread. start()will call the run()method.
-

Runnable Interface Vs Extending Thread Class :

Since we've discussed both the ways to create a thread in Java. There might be a question in your mind that should we use the Runnable interface or Thread class to implement a thread in Java. Let me answer this question for you. The Runnable interface is preferred over extending the Thread class because of the following reasons :

- As multiple inheritance is not supported in Java, it is impossible to extend the Thread class if your class had already extended some other class.
- While implementing Runnable, we do not modify or change the thread's behavior.
- More memory is required while extending the Thread class because each thread creates a unique object.
- Less memory is required while implementing Runnable because multiple threads share the same object.

Java Thread Life Cycle



- New - Instance of thread created which is not yet started by involving start(). In this state, the thread is also known as the born thread.

- Runnable - After invocation of start()& before it is selected to be run by the scheduler.
- Running - After the thread scheduler has selected it.
- Non-runnable - thread alive, not eligible to run.
- Terminated - run() method has exited.

Java Collections Framework

Collection Framework

A collection represents a group of object Java collections provide classes and Interfaces for us to be able to write code interfaces for us to be able to write code quickly and efficiently

Why do we need collections

We need collections for efficient storage and better manipulation of data in java

For ex: we use arrays to store integers but what if we want to

- Resize this array?
- Insert an element in between?
- Delete an elements in Array?
- Apply certain operations to change this array?

Collections Hierarchy in Java

How are collections available

Collections in java are available as class and interfaces Folling are few commonly used collections in java :

- ArrayList -> For variables size collections
- Set -> For distinct collection
- Stack-> A LIFO data structure
- HashMap -> For strong key - value pairs

Collections class is available in java util package collection class also provides static methods for sorting , searching etc.

```
package com.company;


import java.util.ArrayList;

import java.util.Set;
```

```
import java.util.TreeSet;
```

ArrayList in Java: Demo & Methods

- The ArrayList class is the dynamic array found in the java.util package.
- The size of the normal array can not be changed, but ArrayList provides us the ability to increase or decrease the size.
- ArrayList is slower than the standard array, but it helps us to manipulate the data easily.

How to declare an ArrayList :

```
ArrayList<Integer> l1 = new ArrayList<>(); /*Creates an ArrayList object of
                                integer type */
```

Performing various operations on ArrayList :

ArrayList comes with a number of methods that can be used to manipulate the data of the ArrayList. Let's take a look at some of the important methods of ArrayList :

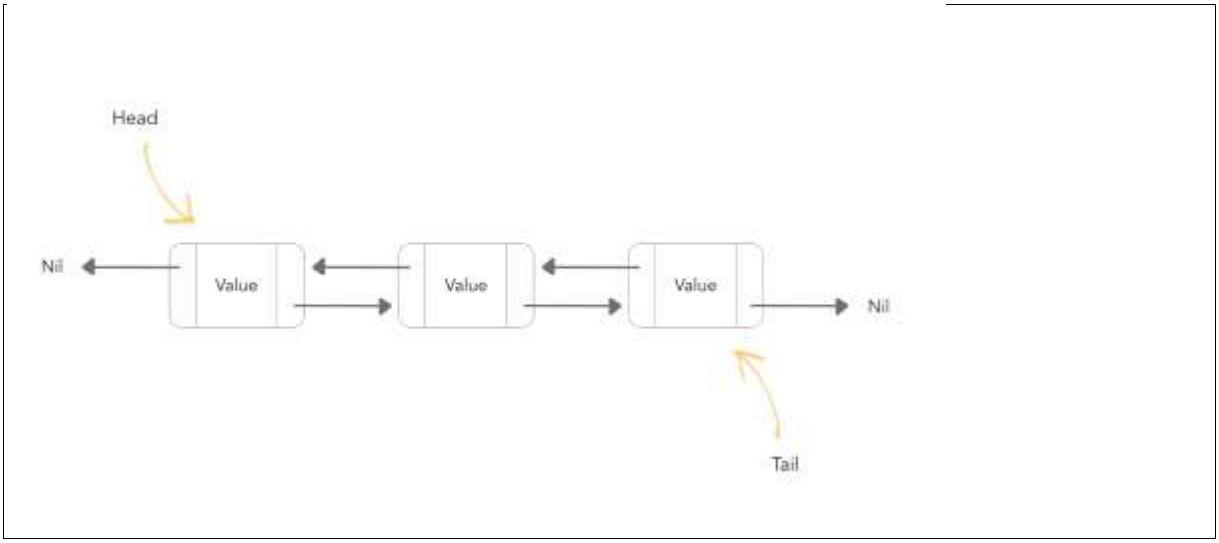
- Adding an element :
- add() method is used to insert an element in the ArrayList.
- add(Object): Inserts an element at the end of the ArrayList.
- add(Index,Object) : Inserts an element at the given index.

```
import java.util.*;
public class ARK extends Thread{
   public static void main(String[] args) {


      ArrayList<Integer> l1 = new ArrayList<>();
```

```
        ll.add(1);
        ll.add(2);
        ll.add(3);
        ll.add(4);
        ll.add(6);
        ll.add(5,5);  // inserts 5 at the 5th index in ll

        System.out.println(ll);
    }
}
```

LinkedList in Java: Demo & Methods

- The LinkedList class in Java provides us with the doubly linked list data structure.
- Each element of the linked list is known as a node.
- Each node points to the address of the next node & its previous node.



- Linked lists are preferred over the Array list because the insertion & deletion in the linked lists can be done in a constant time. But, in arrays, if we want to add or delete an element in between then, we need to shift all the other elements.
- In a linked list, it is impossible to directly access an element because we need to traverse the whole linked list to get the desired element.

ArrayList Vs. LinkedList :

Although ArrayList & LinkedList both implement the List interface and have the same methods, it is important to understand when to use which one.

- The insertion & deletion can be done in constant time in Linked List, so it is best to use the linked list when you need to add or remove elements frequently.
- Use ArrayList when you want to access the random elements frequently, as it can't be done in a linked list in constant time.

Performing various operations on Linked List :

Adding Element in LinkedList:

- Similar to ArrayList, add() method is used to add elements in a linked list.
- add(Object): Inserts an element at the end of the ArrayList.
- add(Index,Object) : Inserts an element at the given index.

  Example :

```
import java.util.*;
public class ARK extends Thread{
    public static void main(String[] args) {

        LinkedList<Integer> ll = new LinkedList<>();

        ll.add(11);
        ll.add(22);
        ll.add(33);
```

```
        l1.add(44);
        l1.add(55);
        l1.add(77);
        l1.add(5,77); // Inserts 77 at index 5
        System.out.println("L1 Linked list : "+ l1);

    }
```

```
}
```

Output: L1 Linked list : [11, 22, 33, 44, 55, 77, 77]

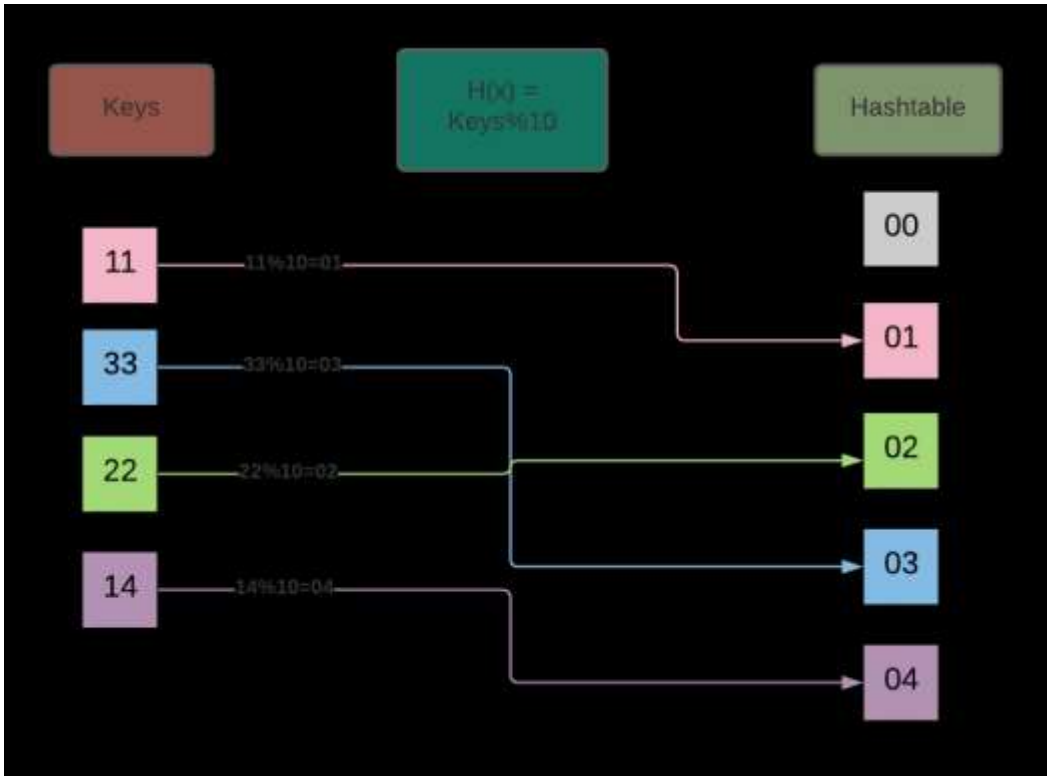
Hashing in Java

Hashing is the technique to convert the range of key-value pairs to a range of indices. In hashing, we use hash functions to map keys to some values.

Example :

Let arr=[11,33,22,14]

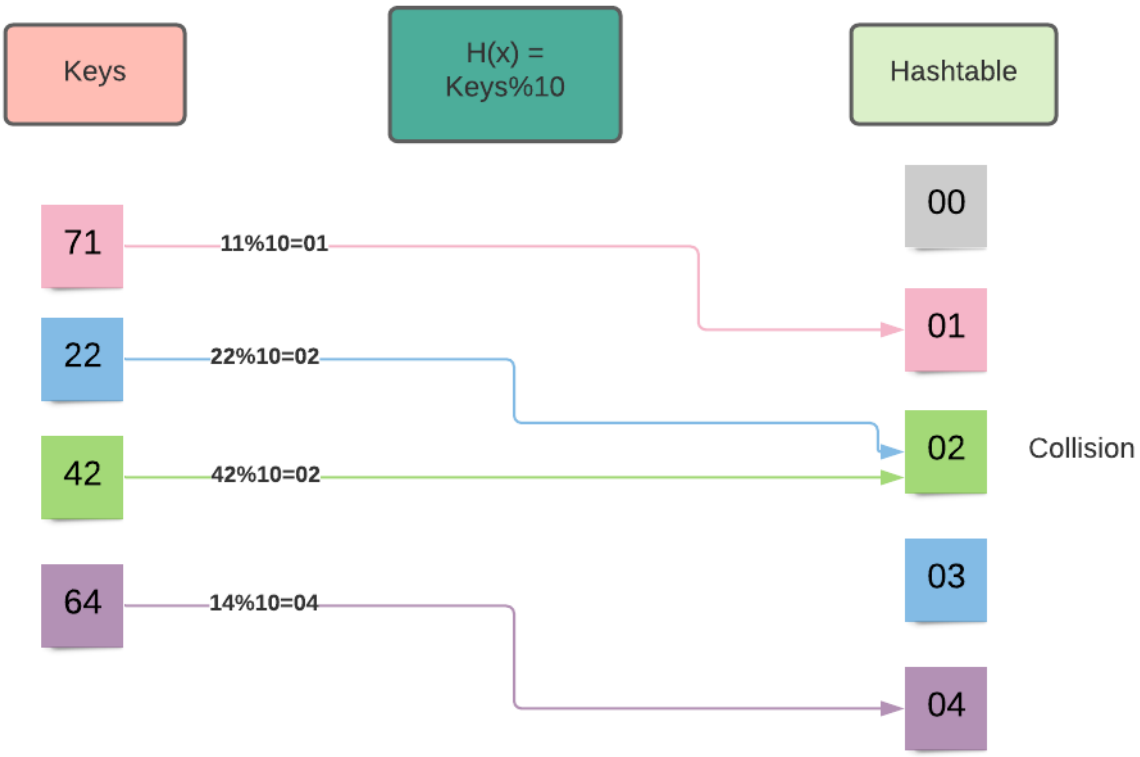
hashIndex = (key %10)



Collision: The hash function may map two key values to a single index. Such a situation is known as a collision.

Example : Let 1l=[22,42,64,71]

H(x) = keys%10



In the above image, you can see that the 22 and 44 are mapped to the index number 2. Therefore we need to avoid the collision. Following techniques are used to avoid collision in hashing :

- Open addressing
- Chaining

HashSet in Java

- HashSet class uses a hash table for storing the elements.
- It implements the set interface.
- Duplicate values are not allowed.
- Before storing any object, the hashset uses the hashCode() and equals() method to check any duplicate entry in the hash table.
- Allows null value.
- Best suited for search operations.

Constructors Of HashSet :

- **HashSet():** This constructor is used to create a new empty HashSet that can store 16 elements and have a load factor of 0.75.
- **HashSet(int initialCapacity):** This constructor is used to create a new empty HashSet which has the capacity to store the specified number of elements and having a load factor of 0.75.
- **HashSet(int initialCapacity, float loadFactor):** This constructor is used to create a new empty HashSet with the capacity & load factor equal to specified integer and float value.
- **HashSet(Collection<? extends E> c):** This constructor is used to create a HashSet using the elements of collection c.

Example :

```
import java.util.*;

public class ARK extends Thread{

   public static void main(String[] args) {

      HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);

      myHashSet.add(6);

      myHashSet.add(8);

      myHashSet.add(3);

      myHashSet.add(11);

      myHashSet.add(11); // This element will be ignored


      System.out.println(myHashSet);


   }

}
```

Output :

```
[8, 3, 11, 6]
```