Time Complexity of Java Collections API

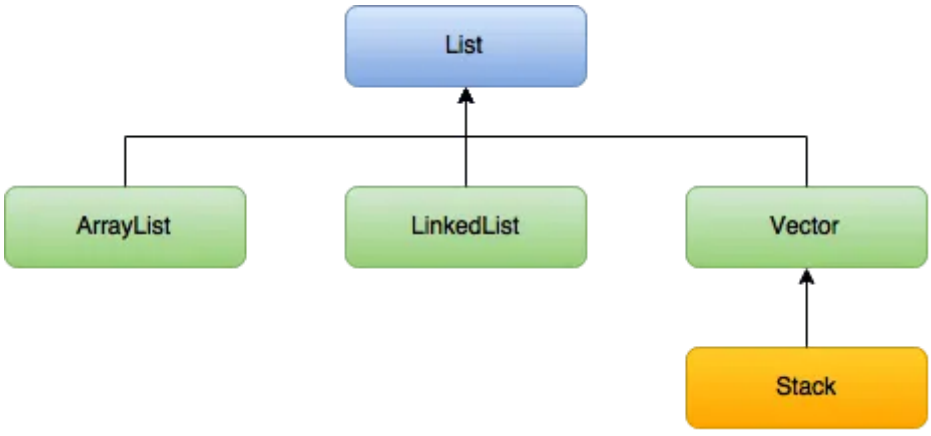
Bikash Dubey · Follow

10 min read · Nov 16, 2020

In this tutorial, **we'll talk about the performance of different collections from the Java Collection API.** When we talk about collections, we usually think about the *List, Map,* and *Set* data structures and their common implementations. Usually, **when we talk about time complexity, we refer to Big-O notation.**

List Interface:

We will start with **list interface**, which is an ordered collection. It is a child interface of Collection. It is an ordered collection of objects in which duplicate values can be stored. Since List preserves the insertion order, it allows positional access and insertion of elements.



List interface implemented classes

We all know that an array is a collection of homogeneous elements stored at contiguous memory locations. An array is the best choice to solve a problem if we already know how much the size of the array should be to solve that problem. But if we don't know that how much should be the size of the array then it is not a good choice to use array. Basically this is the limitation of the array is that the size of the array is predefined and fixed. There are multiple ways to solve this problem. But In this article, we will talk about the difference between two classes which are implemented to solve this problem named as ArrayList and LinkedList.

What is ArrayList?

ArrayList is one of the important part of the collection framework which is present in the java.util package and provides us dynamic arrays in Java. The main difference between built-in array and an ArrayList in Java is that size of built-in array cannot be changed/modified on the other hand we can modify/change the size of an ArrayList as per our requirement. But when we compare the performance of built-in array and ArrayList then we will find that built-in array is more efficient as compared to ArrayList. But ArrayList can be helpful in programs where lots of manipulation in the array is needed. We can dynamically add and remove items. It automatically resizes itself. Lets see a small program to see the implementation of ArrayList :

```
import java.io.*;
import java.util.*;

class ArrayListDemo
{
    public static void main(String[] args)
    {
        ArrayList<Integer> arrList= new ArrayList<Integer>();
        for (int i = 1; i <= 5; i++)
            arrList.add(i);
        System.out.println(arrList);
        arrList.remove(2);
        arrList.add(3,10);
        System.out.println(arrList);
    }
}
```

Output:

```
[1, 2, 3, 4, 5]
[1, 2, 4, 10, 5]
```

Here as you can see in this program I did not give any size at the time of making the ArrayList object and as per requirement, I am adding and removing the elements in the arrList. It automatically resizes itself.

What is LinkedList?

LinkedList is one of the collection framework which is present in the java.util package and provides us dynamic arrays in Java. But LinkedList implementation is different from ArrayList implementation. LinkedList implementation is based on double linked list data structure where where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. Here each object is known as node. Lets see a small program to see the implementation of LinkedList:

```
import java.util.*;

public class LinkedListDemo
{
    public static void main(String args[])
    {
        LinkedList<String> linkedList = new LinkedList<String>();
        linkedList.add("B");
        linkedList.add("I");
        linkedList.add("A");
        linkedList.add(2,"K");
        linkedList.add("S");
        linkedList.addLast("H");
        System.out.println(linkedList);
        linkedList.remove("B");
        linkedList.removeFirst();
        System.out.println("Linked list after deletion: " +
linkedList);
    }
}
```

Output:

```
[B, I, K, A, S, H]
Linked list after deletion: [K, A, S, H]
```

When to use ArrayList & LinkedList?

We saw the implementation of both ArrayList and LinkedList and both of them solve the problem of built-in array. LinkedList and ArrayList are two different implementations of the List interface. LinkedList implements it with a doubly-linked list. ArrayList implements it with a dynamically re-sizing array. However, there are many differences between ArrayList and LinkedList classes. But the question is which one to use? Or is ArrayList is better than LinkedList? But there is nothing like ArrayList is better than LinkedList. Its all about the requirement, ArrayList is best choice when our frequent operation in program is to read the elements from the array because ArrayList implements RandomAccess interface which provide support for fast (constant time) random access for ArrayList. But When our frequent operation is addition/deletion operations then we should go for LinkedList. LinkedList is much faster as compare to ArrayList in such cases because less shift operation is required for addition/deletion in LinkedList as compared to ArrayList.

Performance and Time Complexity:

So, let's focus on the time complexity of the common operations, at a high level:

List	Add	Remove	Get	Contains	Next	Data Structure
ArrayList	O(1)	O(n)	O(1)	O(n)	O(1)	Array
LinkedList	O(1)	O(1)	O(n)	O(n)	O(1)	Linked List

What is Vector?

Vector is a class which is exactly same as ArrayList but the main difference is Vector is synchronized in nature. It means it thread-safe. It is introduced in 1.0 version thats why it is also known as **Legacy class**. As it implements Collection , it inherit all the methods of Collection but apart from this Vector class has its own methods also.

```
import java.io.*;
import java.util.*;

class VectorDemo
{
    public static void main(String[] args)
    {
        List<Integer> v = new Vector<Integer>();
        v.add(1);
        v.add(3);
        v.add(2);
        System.out.println("Elements:"+ v);
        v.remove(1);
        System.out.println("Elements:"+ v);
    }
}
```

Output:

```
Elements:[1, 3, 2]
Elements:[1, 2]
```

What is Stack?

Stack is a class which is implements collection framework and extends the vector class . The underline data structure of Stack class is Stack data structure. The class is based on the basic principle of last-in-first-out. It is introduced in 1.0 version that's why it is also known as **Legacy class**. As it implements Collection , it inherit all the methods of Collection but apart from this Stack class has its own methods also.

```
import java.io.*;
import java.util.*;

class StackDemo {
    public static void main(String[] args)
    {
        Stack<Integer> stack = new Stack<Integer>();
        stack.add(1);
        stack.add(2);
        stack.push(3);
        System.out.println("Elements :"+stack);
```

```
            stack.pop();
            System.out.println("Elements :"+stack);
        }
    }
```

Output:

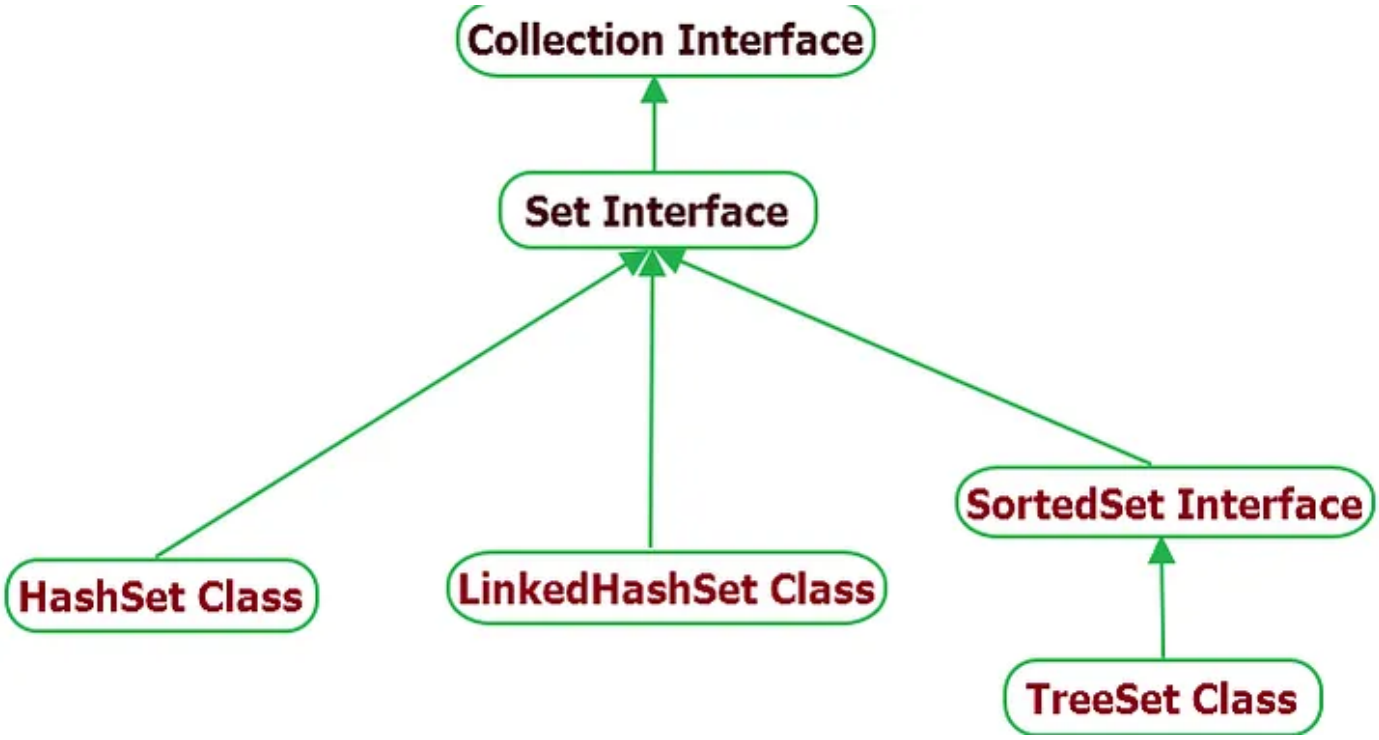
```
Elements :[1, 2, 3]
Elements :[1, 2]
```

Performance and Time Complexity:

Lets talk about the time complexity and performance of Vector and Stack classes. There is not huge difference in performance and time complexity between these classes. Stack internally uses Stack data structure for storing objects and gives O(1) complexity for insertion and removing but at the time of searching the time complexity is O(n). On the other hand, Time complexity of Vector class for retrieving is O(1) and addition ,removal is O(1) if we want to insert and remove at the last .

Set interface:

Set is an interface which present in the java.util package and it extends the Collection interface. Set is an unordered collection of objects in which duplicate values cannot be stored. In a Set, there are no duplicate elements this is one of the major reason to use a Set interface. We have few classes which is the implementation of Set interface i.e. HashSet,LinkedHashSet,TreeSet etc.



Set interface implemented classes

Now the question is When and which to use. We will go one by one, first we will discuss about HashSet.

What is HashSet?

HashSet is a class present in java.util package which implements the Set interface . A HashSet is a collection of elements where every element is unique, it means duplicates are not allowed. The underlying data structure for HashSet is Hashtable. I'm gonna explain this with an example.

```
import java.util.*;

class HashSetDemo
{
    public static void main(String[] args)
    {
        HashSet<String> hashSet = new HashSet<String>();
        hashSet.add("Hello");
        hashSet.add("World!!");
        hashSet.add("Hello");

        System.out.println("Elements :"+ hashSet);
```

```
    }
}
```

Output:

```
Elements :[World!!, Hello]
```

As we can see that we inserted “Hello” 2 times inside hashSet. But as duplicates are not allowed in the HashSet, we did not get “Hello” 2 times in the result. One more thing we should notice that the insertion order is not preserved in HashSet as it adds the elements as per their hashcode.

What is LinkedHashSet?

LinkedHashSet is a class which extends HashSet and implements Set interface. LinkedHashSet is an ordered version of HashSet. A LinkedHashSet is a collection of elements where every element is unique, it means duplicates are not allowed but here the insertion order is preserved. It means LinkedHashSet lets us iterate through the elements in the order in which they were inserted. The underlying data structure of LinkedHashSet is HashTable and LinkedList. I’m gonna explain this with an example.

```
import java.util.*;

class LinkedHashSetDemo
{
    public static void main(String[] args)
    {
        LinkedHashSet<String> hashSet = new LinkedHashSet<String>();
        hashSet.add("Hello");
        hashSet.add("World!!");
        hashSet.add("Hello");

        System.out.println("Elements :"+ hashSet);
    }
}
```

Output:

```
Elements :[Hello, World!!]
```

As we can see just like HashSet duplicates are also not allowed in LinkedHashSet but the insertion order of elements is preserved in LinkedHashSet.

What is TreeSet?

TreeSet is similar to HashSet except that it sorts the elements in the ascending order while HashSet doesn't maintain insertion order. TreeSet uses TreeMap internally to store it's elements. Let's dig in with an example.

```
import java.util.*;

class TreeSetDemo
{
    public static void main(String[] args)
    {
        TreeSet<String> treeSet = new TreeSet<String>();
        treeSet.add("World!!");
        treeSet.add("Hello");
        treeSet.add("Hello");

        System.out.println("Elements :"+ treeSet);
    }
}
```

Output:

```
Elements :[Hello, World!!]
```

When to use HashSet, LinkedHashSet and TreeSet?

Even though, HashSet,LinkedHashSet and TreeSet are all implementations of Set interface, there are some differences exist between them. Its all about the requirement , as per requirement we should choose HashSet,LinkedHashSet and TreeSet. Suppose If we don't want to maintain insertion order but want store only unique objects then in

this case we should go for HashSet. If we want to maintain insertion order of elements as well as we dont want to store duplicates in our collection then we can use LinkedHashSet. If we want to sort the elements according to some sorting order and we dont want to store duplicate elements then we should use TreeSet.

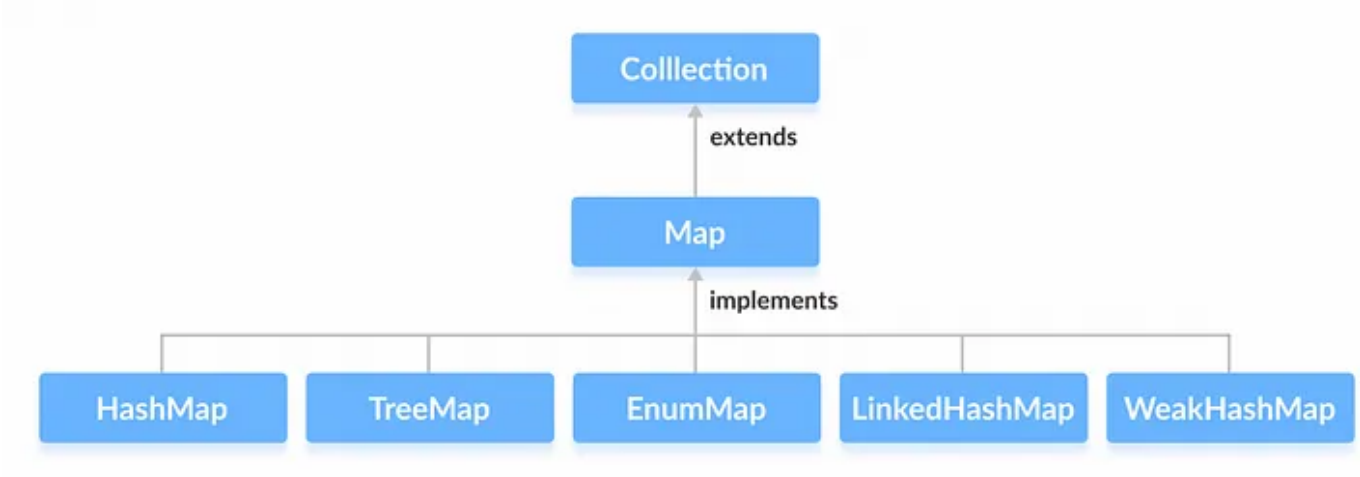
Performance and Time Complexity:

Lets talk about the time complexity and performance of these classes. There is not huge difference in performance and time complexity between these classes. HashSet internally uses HashMap for storing objects and gives O(1) complexity for insertion, removing and retrieving objects. HashSet performance is better according to LinkedHashSet and TreeSet. The performance of LinkedHashSet is almost similar to HashSet and time complexity for insertion, removing and retrieving operations is order O(1). But it is slower because, LinkedHashSet maintains LinkedList internally to maintain the insertion order of elements. TreeSet uses TreeMap internally to store objects and TreeSet performance is better to LinkedHashSet excluding insertion and removal operations because, it has to sort the elements after each insertion and removal operations. The time complexity of TreeSet is order O(log(n)) for insertion, removing and retrieving operations.

Set	Add	Remove	Contains	Next	Size	Data Structure
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Red-black tree
HashSet	O(1)	O(1)	O(1)	O(h/n)	O(1)	Hash Table
LinkedHashSet	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List

Map interface:

The Map interface present in java.util package , it represents a mapping between a key and a value. The Map interface is not a child interface of the Collection interface. Therefore the implementation of map interface is different from Collection interface.



Since Map is an interface, it can be used only with a class that implements this interface. Now, let's see how to perform a few frequently used operations on a Map using its implemented classes.

What is HashMap?

HashMap class implements the Map interface which allows us *to store key and value pair,* where keys should be unique. It is easy to perform operations using the key index like updation, deletion, etc. It is introduced in 1.2 version and present inside java.util package.

```
import java.util.HashMap;

public class HashMapDemo
{
    public static void main(String[] args)
    {
        HashMap<Integer, String> hashMap = new HashMap<>();
        hashMap.put(1,"Hello");
        hashMap.put(2,"World!!");
        hashMap.put(3,"Hello!!");
        System.out.println("Elements:"+hashMap);
        System.out.println("Element based on key:"+hashMap.get(1));
    }
}
```

Output:

```
Elements:{1=Hello, 2=World!!, 3=Hello!!}
Element based on key:Hello
```

What is LinkedHashMap?

The **LinkedHashMap** is just same as HashMap with an additional feature of preserving the order of elements inserted into it. It is introduced in 1.4 version and it is also present inside java.util package.

```
import java.util.LinkedHashMap;

public class LinkedHashMapDemo
{
    public static void main(String[] args)
    {
        LinkedHashMap<String, Integer> linkedHashMap = new
LinkedHashMap<>();
        linkedHashMap.put("Hello",1);
        linkedHashMap.put("World!!",2);
        linkedHashMap.put("Hello!!",3);
        System.out.println("Elements:"+linkedHashMap);
        System.out.println("Element based on
key:"+linkedHashMap.get("Hello"));
    }
}
```

Output:

```
Elements:{Hello=1, World!!=2, Hello!!=3}
Element based on key:1
```

What is TreeMap?

TreeMap is a class which in present inside java.util package. It provides an efficient means of storing key-value pairs in sorted order. Java TreeMap maintains ascending order based on key.

```
import java.util.*;
class TreeMapDemo
{
    public static void main(String args[])
    {
        TreeMap<Integer,String> treeMap = new TreeMap<Integer,String>
();
        treeMap.put(3, "Hello");
        treeMap.put(2, "World");
        treeMap.put(1, "Hello");
        System.out.println("Elements:"+treeMap);
    }
}
```

Output:

```
Elements:{Hello=1, World!!=2, Hello!!=3}
Element based on key:1
```

When to use HashMap, LinkedHashMap and TreeMap?

Even though, **HashMap, LinkedHashMap and TreeMap** are all implementations of Map interface, there are some differences exist between them. Its all about the requirement , as per requirement we should choose HashMap,LinkedHashMap and TreeMap. Suppose If we want to store person name with their address. Here we have to use collection api where we can store like key-value pair. For this requirement, we can use any map implemented class. But suppose we don't want to maintain insertion order then in this case we should go for HashMap. If we want to maintain insertion order of elements in our collection then we can use LinkedHashMap. If we want to sort the elements according to some sorting order then we should use TreeMap.

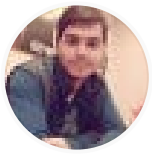
Performance and Time Complexity:

Lets see the time complexity of these classes.

Map	Get	ContainsKey	Next	Data Structure
TreeMap	O(log n)	O(log n)	O(log n)	Red-black tree
HashMap	O(1)	O(1)	O(h / n)	Hash Table
LinkedHashMap	O(1)	O(1)	O(1)	Hash Table + Linked List

With this, we come to an end to this blog. I hope you got a clear understanding about java collection apis and their uses.

Blog Java Collections In Java Time Complexity

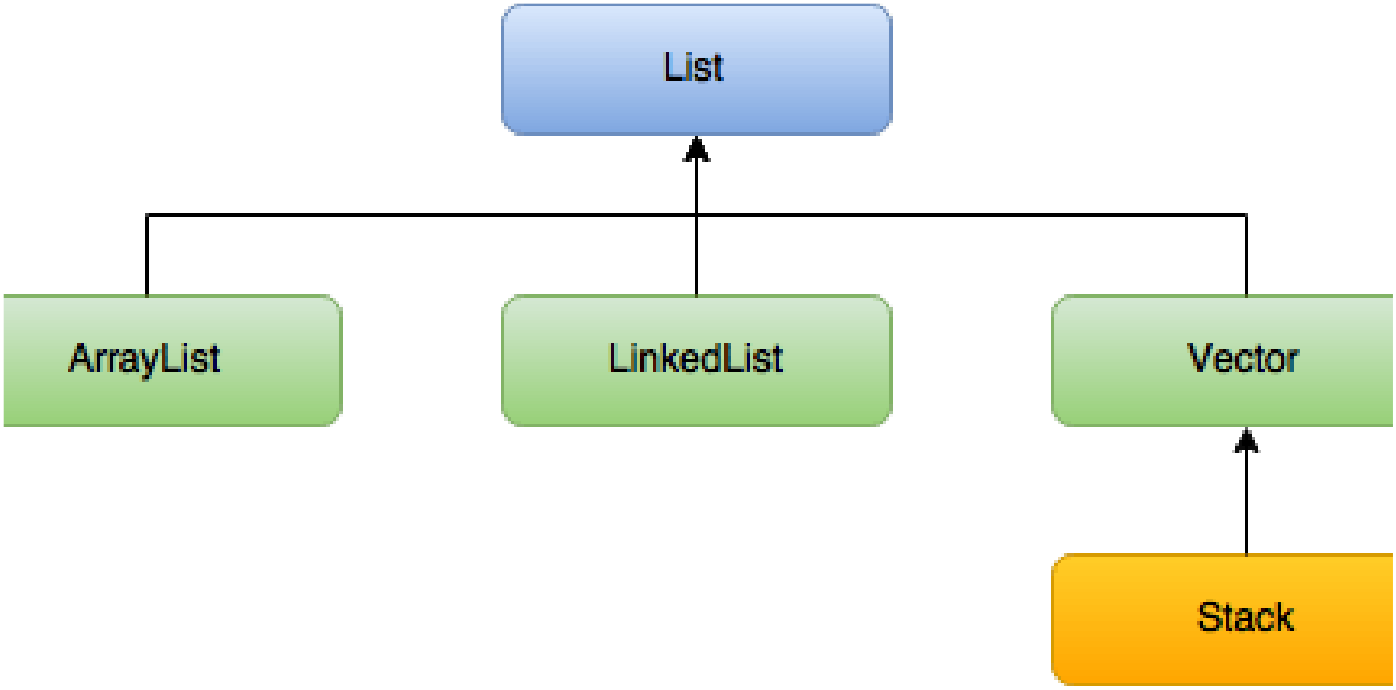


Follow

Written by Bikash Dubey

7 Followers

More from Bikash Dubey



Bikash Dubey

ArrayList vs LinkedList in Java

We will start with a simple definition of array. We all know that an array is a collection of homogeneous elements stored at contiguous...

10 min read · Nov 15, 2020



Bikash Dubey

Introduction to Servlet & JSP :

Hello friends, are you interested in developing web applications in Java and you heard about Servlets and JSP and you want to know what...

4 min read · Dec 6, 2020

2

```
    color: white;
    height: 100px;
    display: flex;
    flex-flow: row wrap;
    text-align: center;
  }
  .box {
    padding: 10px;
    flex: 1 0 auto;
  }
  .larger {
    background: linear-gradient(red, yellow, green);
    flex: 3 0 auto;
  }
  </style>
  <div class="outer-container">
    <div class="box">Container 1</div>
```

Bikash Dubey

A beginner’s guide to Flexbox

In this article, we will discuss about Flexbox. What is Flexbox and Why we should use this layout for user interface design? Let's start...

4 min read · Nov 29, 2020

Bikash Dubey

Database Normalization

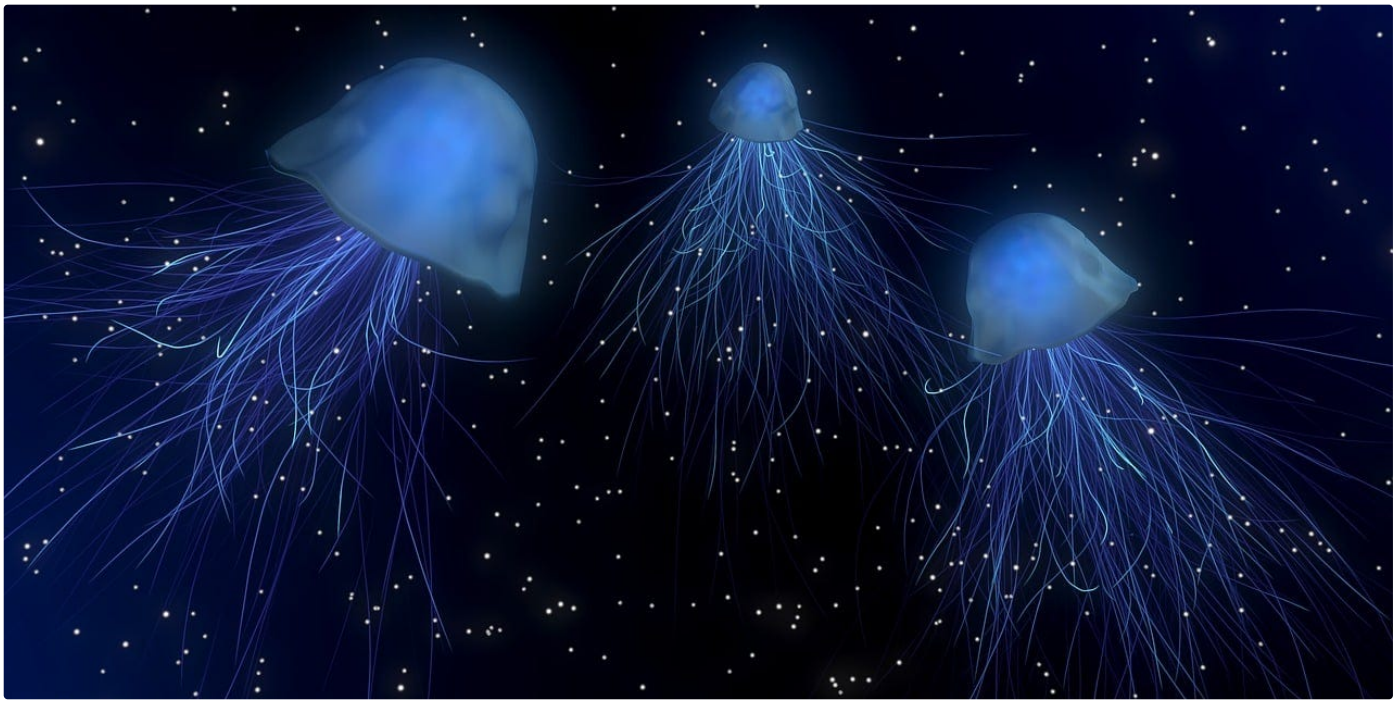
This article explains about database normalization which is helpful when discussing the design of a relational database. This article also...

6 min read · Nov 22, 2020

1

See all from Bikash Dubey

Recommended from Medium



Vesper Lee

A Deep Dive into HashMap

Unlocking the Mysteries of HashMap: A Comprehensive Examination of Its Hashing, Addressing, and Collision Handling Mechanisms

4 min read · Jul 30

2

Belghaouia Amal

Mastering Best Practices in Spring Boot: A Simple Guide

what best practices i follow in my spring boot project

3 min read · 6 days ago

23 1

Lists

It's never too late or early to start something

15 stories · 83 saves

General Coding Knowledge

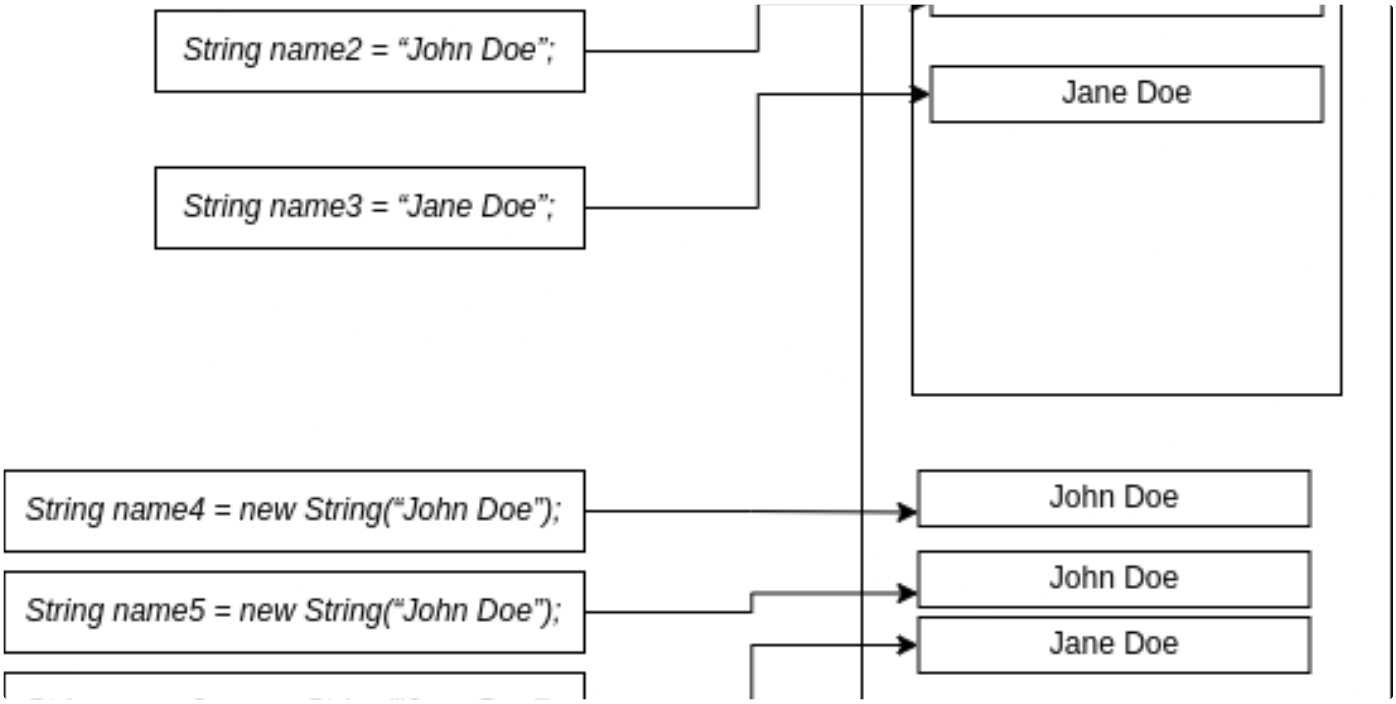
20 stories · 224 saves

Modern Marketing

33 stories · 75 saves

New_Reading_List

174 stories · 76 saves



Ayush Basnet

String Pool in Java

In this article, We'll explore concept of String Pool in Java and how String are managed by JVM.

2 min read · Apr 27

74 1

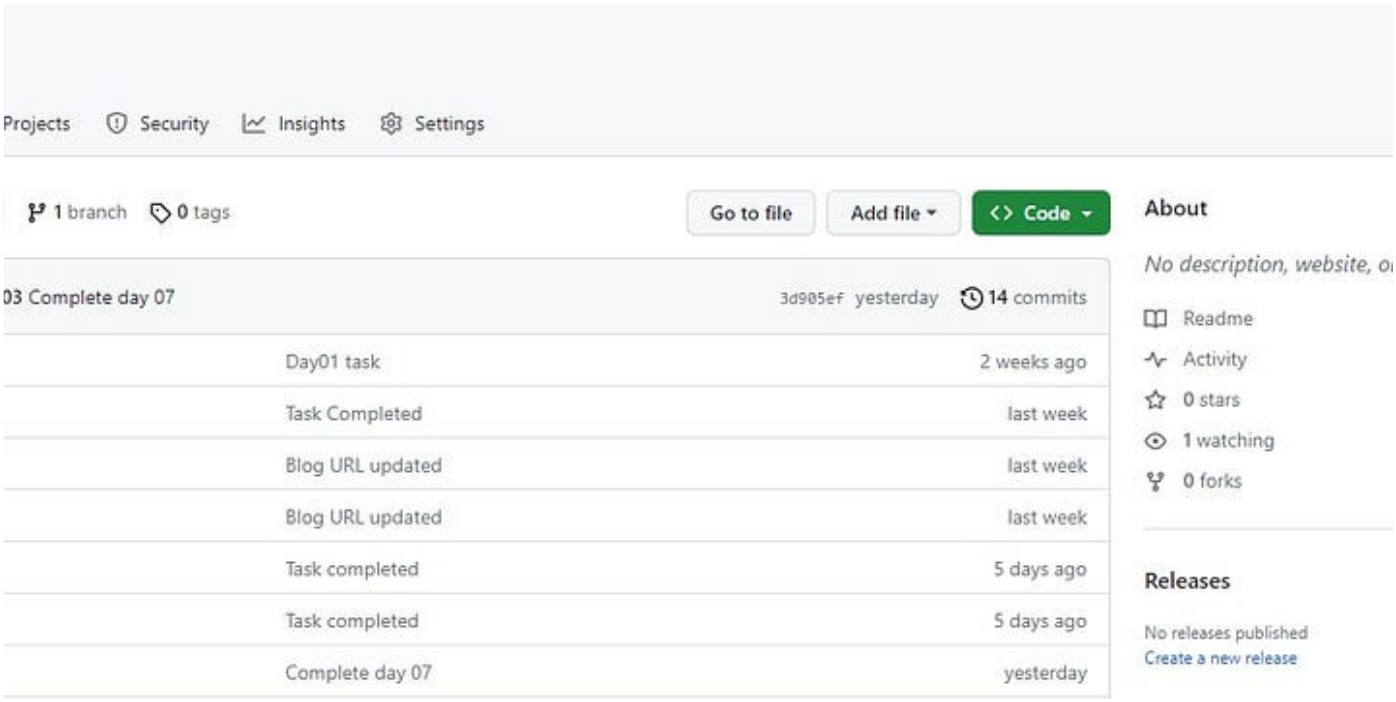
Rupert Waldron

Create a non-blocking REST Api using Spring @Async and Polling

Get the great asynchronous, non-blocking experience you deserve with Spring's basic Rest Api, polling and @Aysnc annotation.

12 min read · Feb 25

40



Kshitij Tripathi

Day-08: Basic Git & GitHub

This is the #90DaysofDevops challenge under the guidance of Shubham Londhe sir.

4 min read · Jun 3

1

Brandon Wohlwend

Full Step by Step Guide in OOP—Java

Hello! Follow me through this article and on this journey about Object Oriented Programming in Java! Why Java? Just because it is one of...

9 min read · Aug 13


2

See more recommendations