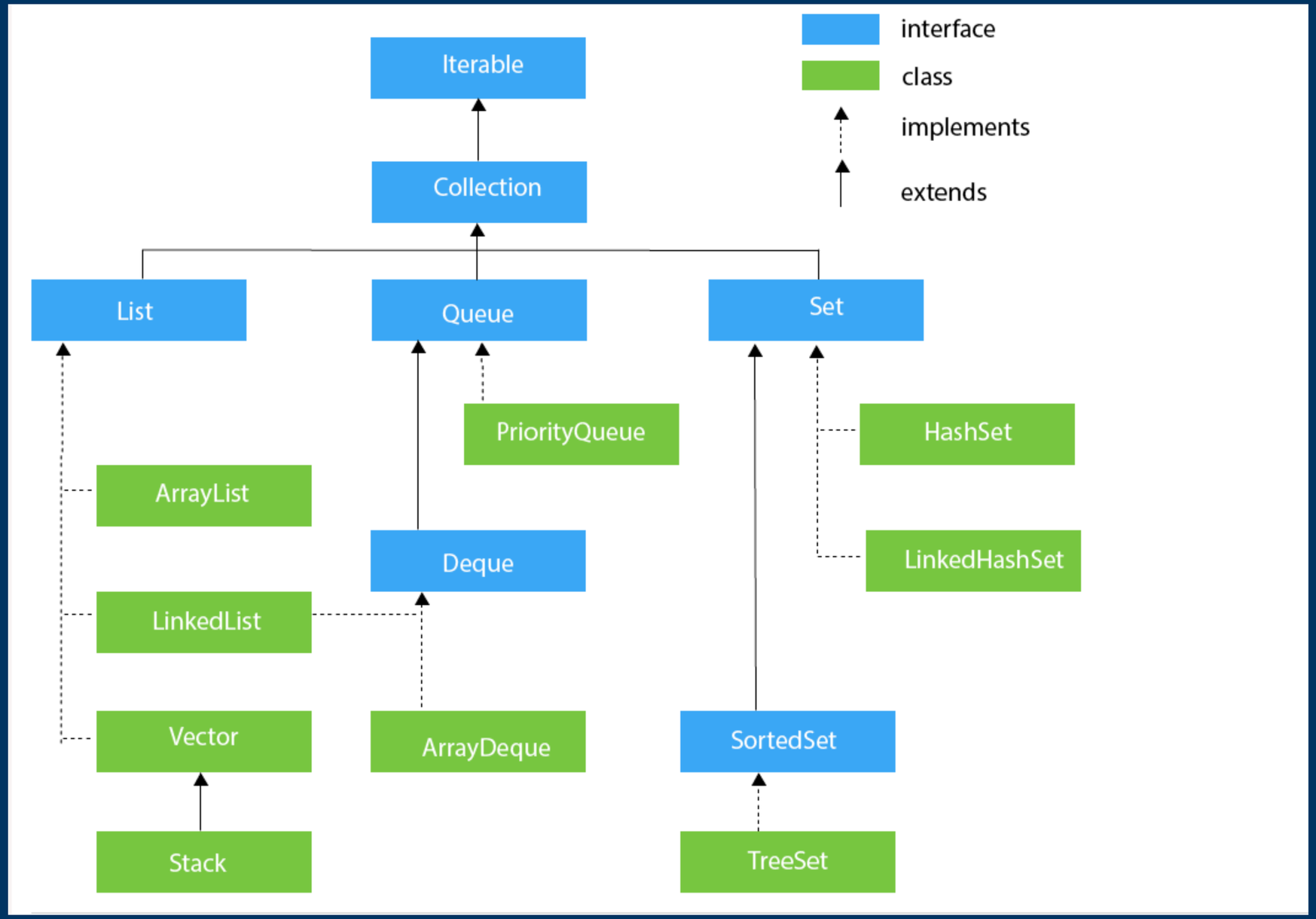
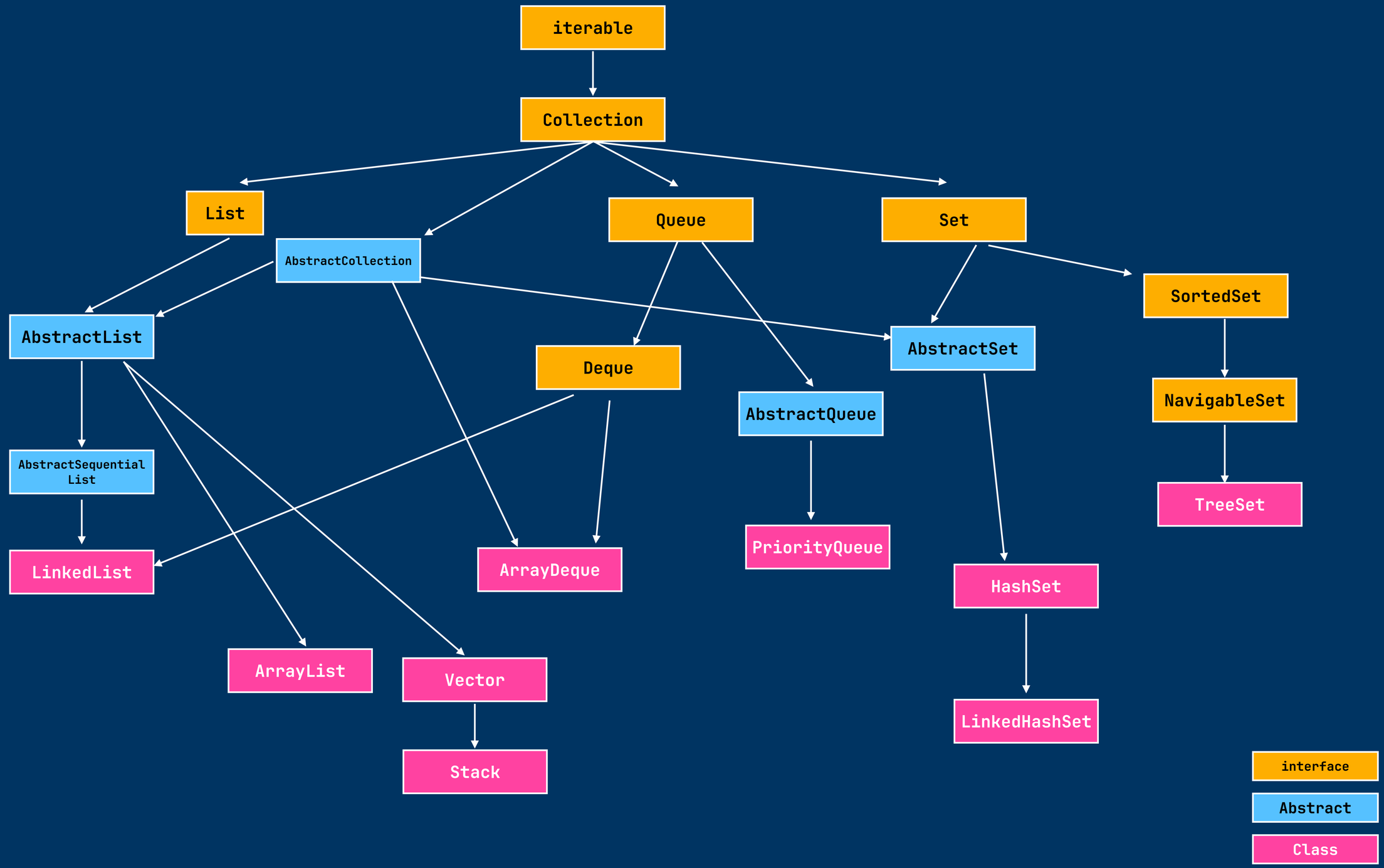
COLLECTION FRAMEWORK

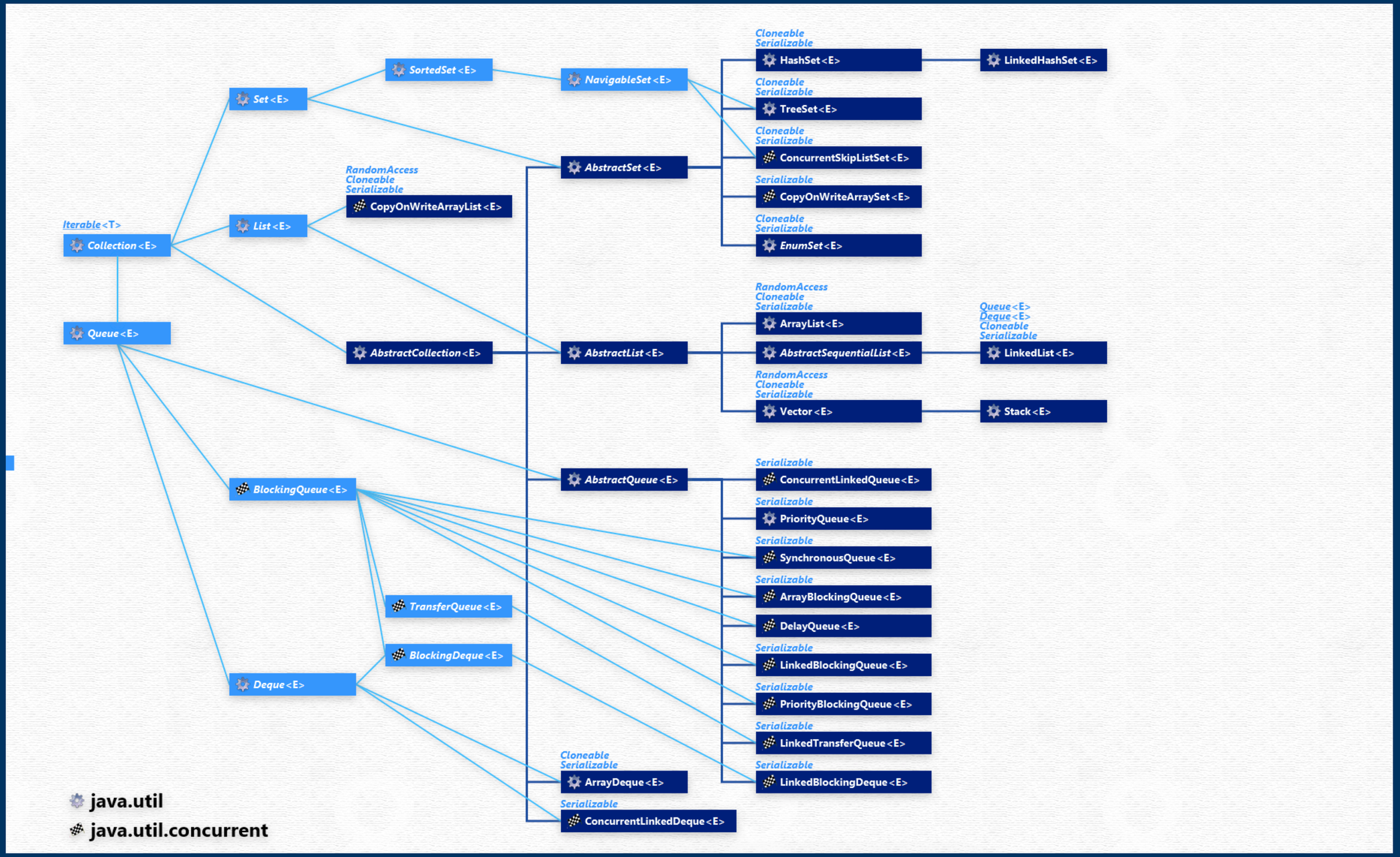
Furkan Şahin KULAKSIZ
Software Developer
Co Founder of Turkiye Java Community

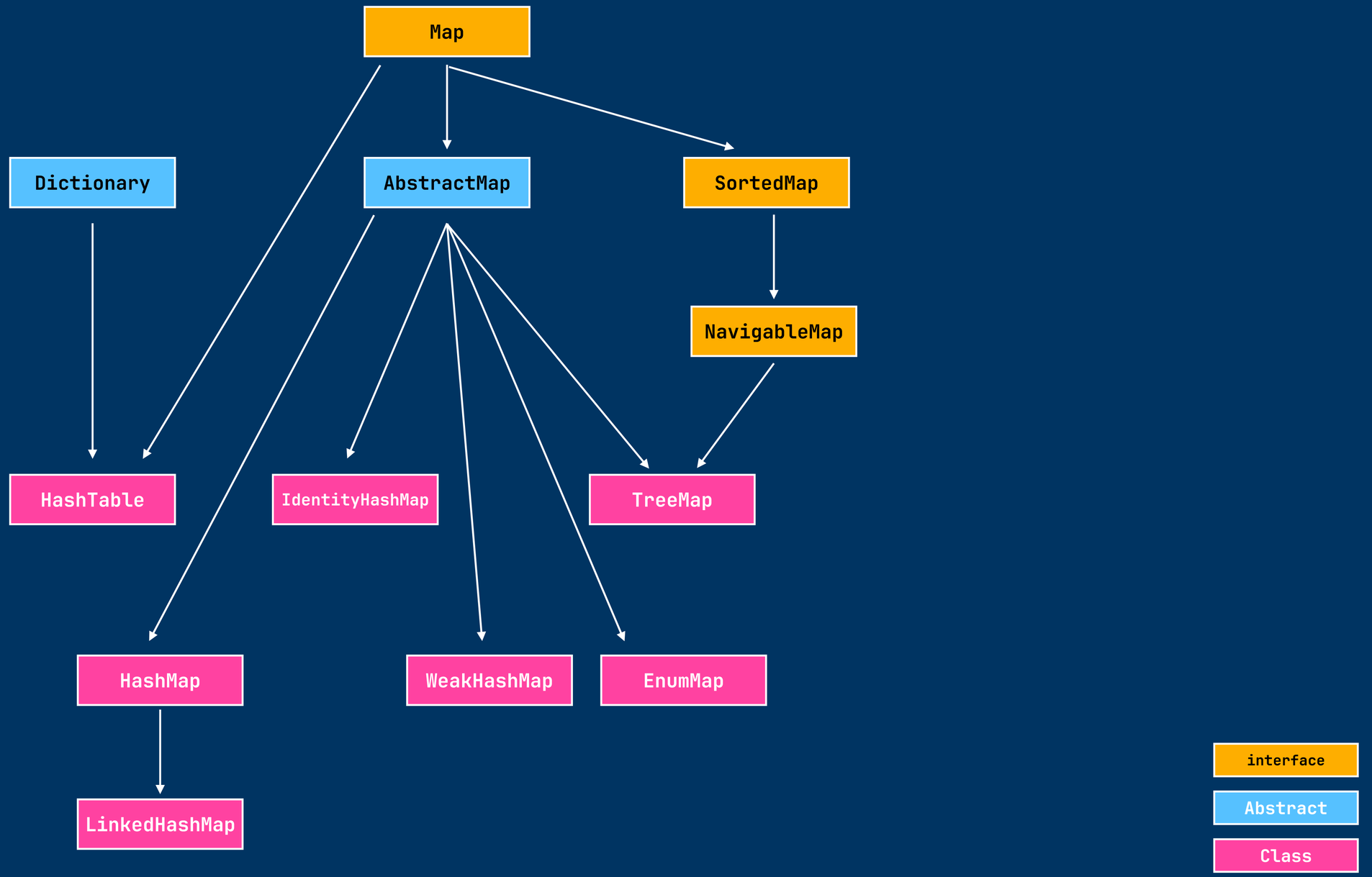
Collection Framework Nedir.?

Java Collection Framework, Java programlama dilinde koleksiyonları (veri gruplarını) manipüle etmek için kullanılan bir API (Application Programming Interface) setidir. Koleksiyonlar, aynı türden veya farklı türlerden verileri depolamak ve işlemek için kullanılan veri yapılarıdır.


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet


iterable
Collection
List
AbstractCollection
Queue
Set
SortedSet
AbstractList
AbstractSet
Deque
AbstractQueue
NavigableSet
AbstractSequential List
TreeSet
LinkedList
ArrayDeque
PriorityQueue
HashSet
ArrayList
Vector
LinkedHashSet
Stack
interface
Abstract
Class


Iterable<T>
Collection<E>
Set<E>
SortedSet<E>
NavigableSet<E>
HashSet<E>
Cloneable
Serializable
LinkedHashSet<E>
TreeSet<E>
ConcurrentSkipListSet<E>
AbstractSet<E>
EnumSet<E>
List<E>
RandomAccess
Cloneable
Serializable
CopyOnWriteArrayList<E>
CopyOnWriteArraySet<E>
Queue<E>
AbstractCollection<E>
AbstractList<E>
ArrayList<E>
AbstractSequentialList<E>
Queue<E>
Deque<E>
LinkedList<E>
Vector<E>
Stack<E>
AbstractQueue<E>
BlockingQueue<E>
TransferQueue<E>
BlockingDeque<E>
Deque<E>
ConcurrentLinkedQueue<E>
PriorityQueue<E>
SynchronousQueue<E>
ArrayBlockingQueue<E>
DelayQueue<E>
LinkedBlockingQueue<E>
PriorityBlockingQueue<E>
LinkedTransferQueue<E>
LinkedBlockingDeque<E>
ArrayDeque<E>
ConcurrentLinkedDeque<E>
java.util
java.util.concurrent


Map
Dictionary
AbstractMap
SortedMap
NavigableMap
HashTable
IdentityHashMap
TreeMap
HashMap
WeakHashMap
EnumMap
LinkedHashMap
interface
Abstract
Class

List

Dinamik bir array gibi düşünülebilir.

Aynı kayıtların birden fazla olmasına izin verir.

Elemanlar sıralandığında, ilgiliyi listeye elemanlar
Hangi sırada eklendiyse o sırada çıktı verir.

Null değerine izin verir

İndeks bazlı çalışır

Set

Aynı kayıtların birden fazla olmasına izin vermez.

Elemanlar sıralandığında, ilgiliyi listeye elemanlar
Hangi sırada eklendiyse o sırada çıktı vermesini garanti etmez.

HashSet null değere izin verirken, TreeSet null değerine izin vermez.

Queue

FIFO - First in First Out - İlk Giren İlk Çıkar

Elemanlara, kuyruğun başındaki eleman üzerinden erişilir.
Kuyruğun sonuna eleman eklenirken, elemanlar kuyruğun başından çıkarılır.

Null Değerine izin veren class'lar

LinkedList

Null Değerine izin vermeyen class'lar

PriorityQueue
ArrayBlockingQueue
LinkedBlockingQueue

DeQue

Elemanlara hem baştan hem de sondan erişilebilir.

ArrayDeQue

Array Double Ended Queue ‘ nun kısaltılması

Bu yapıya hem baştan, hem de sondan erişilebilir.

Hem STACK hem de QUEUE gibi davranır.

Queue: FIFO - First in First Out - İlk giren ilk çıkar

Stack: LIFO - Lost in First Out - Son giren ilk çıkar

Null elemanları kabul etmez.

Map

Key-Value çiftleri ile verileri saklar

Key değerleri sadece bir kere bulunabilir.

Bir key değerine ait bir value’dan birden fazla olabilir.

Key değeri olarak null verilebilir.

Çok önemli Not: Map, Collection interface’inden implemente olmaz. Ama teknik olarak Collection framework’ünün içerisine dahildir. Collection interface’inden implemente olmamasının da sebebi bir grup interface’i olmasından kaynaklıdır. Yani farklı methodlara ihtiyaç duyar.

GENERICLER

< > → Diamonds

Classlarda ve Methodlarda kullanılır

Primitive type’larda kullanılmaz

Static methodlarda Genericler kullanılmaz.

ArrayList’e sınırsız sayıda eleman eklenebilir mi.?

Java'da bir ArrayList'e ekleyebileceğiniz eleman sayısı teorik olarak sınırsızdır. Ancak, gerçekte, bu sayı sisteminizin mevcut bellek (RAM) kapasitesi tarafından sınırlandırılır.

ArrayList sınıfı, içerdiği elemanları genişletmek ve küçültmek için dinamik bir yapı kullanır. ArrayList'e bir eleman eklediğinizde ve mevcut kapasitesi dolu olduğunda, Java'nın ArrayList implementasyonu, genellikle mevcut kapasitenin 1.5 katı büyüklüğünde yeni bir dizi oluşturur ve eski diziden tüm elemanları yeni diziye kopyalar.

Ancak, her bir elemanın kendisi de bellek tüketir ve ayrıca, yeni bir dizi oluşturmak ve eski diziden elemanları kopyalamak için ek bellek gereklidir.

Bu nedenle, bir ArrayList'e ekleyebileceğiniz eleman sayısı, uygulamanızın diğer kısımları tarafından kullanılan bellek miktarına ve sisteminizin toplam bellek kapasitesine bağlıdır.

Ayrıca, ArrayList'in kapasitesi int tipinde belirlenir ve int'in maksimum değeri 2^31-1'dir (Integer.MAX_VALUE). Bu, teorik olarak bir ArrayList'in tutabileceği maksimum eleman sayısıdır. Ancak, pratikte genellikle sistem belleği bu sınıra ulaşılmadan önce tükenir.

ArrayList eleman ekleme stratejisi

1. Java'da ArrayList veri yapısının kapasite yönetimi için birkaç strateji vardır. İlk önce ArrayList nesnesi oluşturulduğunda, hemen bellekte büyük bir alan ayrılmaması için bu gibi stratejiler kullanılır.

ArrayList eleman ekleme stratejisi

1. Yani, ArrayList başlangıçta küçük bir kapasiteyle başlar ve eleman ekledikçe ihtiyaç duyduğu kadarını genişletir. Bu, özellikle ArrayList'in son boyutu hakkında belirsizlik olduğunda, bellek kullanımını optimize etmeye yardımcı olur.

ArrayList eleman ekleme stratejisi

DEFAULTCAPACITY_EMPTY_ELEMENTDATA değişkeni, boş bir ArrayList oluşturulduğunda kullanılır. ArrayList'in varsayılan kapasitesi 10'dur.

ArrayList eleman ekleme stratejisi

Ancak, bu ArrayList oluşturulduğunda hemen 10 elemanlık bir alan ayrılmaz. Bunun yerine, ArrayList'in elementData dizisi boş bir dizi (DEFAULTCAPACITY_EMPTY_ELEMENTDATA) ile başlatılır.

ArrayList eleman ekleme stratejisi

İlk eleman eklendiğinde, ArrayList'in kapasitesi varsayılan kapasiteye (10) genişletilir.

Bu, ArrayList'in gereksiz yere bellek tüketmesini önler, çünkü boş bir ArrayList oluşturulduğunda hemen 10 elemanlık bir alan ayrılmaz.

ArrayList eleman ekleme stratejisi

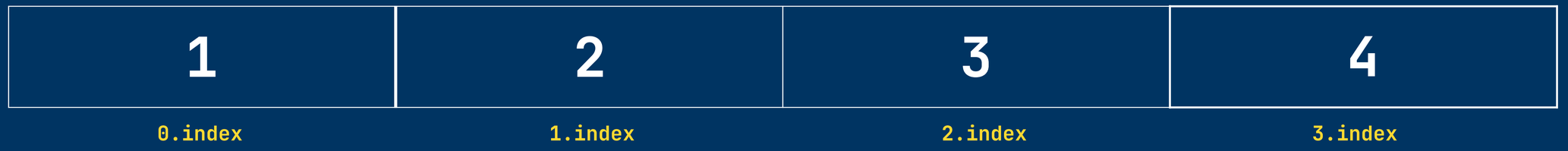
elementData dizisi, ArrayList'in elemanlarını depolayan dizidir ve transient anahtar kelimesi, bu dizinin Java'nın serileştirme mekanizması tarafından dikkate alınmamasını sağlar. Bu, ArrayList'in kendisinin serileştirilebileceği, ancak elemanlarının (yani, elementData dizisindeki objelerin) serileştirilmediği anlamına gelir.

ArrayList eleman ekleme stratejisi

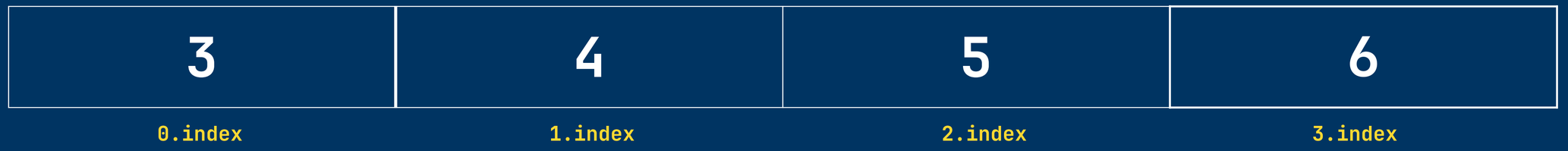
Bu yaklaşım, bellek kullanımını optimize etmek ve ArrayList'in büyüklüğünün dinamik olarak değiştirilebilmesini sağlamak için kullanılır. Genellikle, bir ArrayList'in başlangıçtaki boyutu belirsiz olduğunda ve büyük miktarda belleği gereksiz yere tüketme riski olduğunda kullanılır.

UFAK BİR PROBLEM

List1



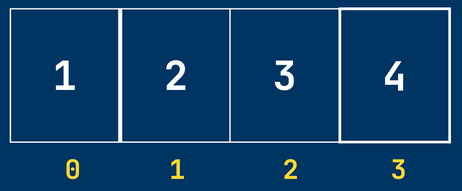
List2



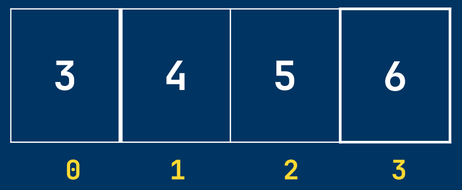
List 1 içerisinden; List1 ve List2'deki ortak elemanları çıkarmak istiyoruz.

```
for (int i = 0; i < integers1.size(); i++) {
    if (integers2.contains(integers1.get(i))) {
        integers1.remove(i);
    }
}
```

List1

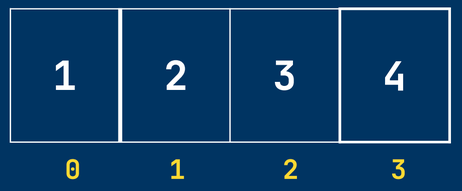


List2

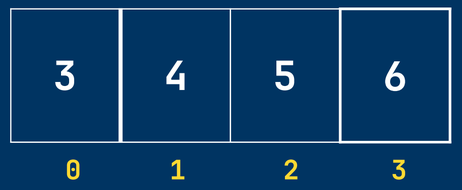


```
for (int i = 0; i < integers1.size(); i++) {
    if (integers2.contains(integers1.get(i))) {
        integers1.remove(i);
    }
}
```

List1



List2

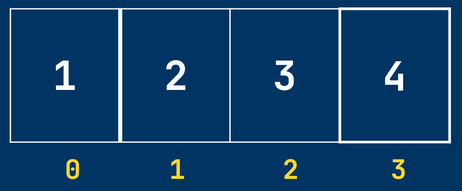


i = 0 için;

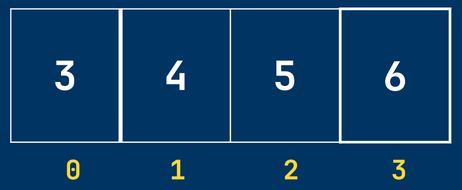
list1.get(i) ≠ list2.get(i)

```java
for (int i = 0; i < integers1.size(); i++) {
    if (integers2.contains(integers1.get(i))) {
        integers1.remove(i);
    }
}
```

List1



List2



i = 1 için;

list1.get(i) ≠ list2.get(i)

```
for (int i = 0; i < integers1.size(); i++) {
    if (integers2.contains(integers1.get(i))) {
        integers1.remove(i);
    }
}
```

List1

1	2	3	4
0	1	2	3

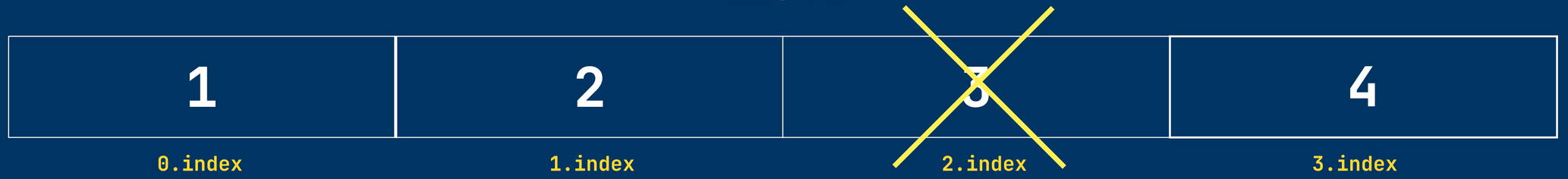
List2

3	4	5	6
0	1	2	3

i = 2 için;

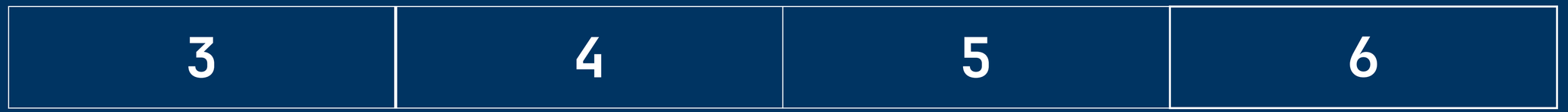
list1.get(i) == list2.get(i)

List1



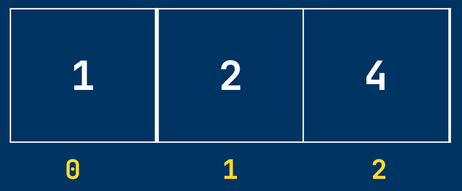
List1’in yeni hali

List2

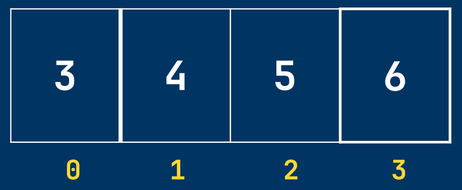


```java
for (int i = 0; i < integers1.size(); i++) {
    if (integers2.contains(integers1.get(i))) {
        integers1.remove(i);
    }
}
```

List1



List2



i = 3 için;

For döngüsü şartı sağlanmaz

List Implementasyonlarının Karşılaştırılması

ArrayList

İndex bazlı çalışır.

Veri güncellemek ya da veri ekleyip silmek daha yavaştır.

LinkedList

İndex bazlı çalışmaz.

Veri okumak daha yavaştır.

LinkedList bellekte daha fazla yer kaplar.

Set Implementasyonlarının Karşılaştırılması

HashSet

Elemanlarını bir has tablosunda saklar

Elemanların eklenmesi ve bir elemanın aranması sabit zamanlıdır

Elemanlar eklendiği sırayı kaybeder

TreeSet

Elemanlarını bir tree yapısında saklar.

Elemanlar sıralı bir şekilde ekrana gelir.

Ağaç büyüdükçe HashSet’e göre performansı azalır.

Set Implementasyonlarının Karşılaştırılması

HashSet

Elemanlarını bir has tablosunda saklar

Elemanların eklenmesi ve bir elemanın aranması sabit zamanlıdır

Elemanlar eklendiği sırayı kaybeder

TreeSet

Elemanlarını bir tree yapısında saklar.

Elemanlar sıralı bir şekilde ekrana gelir.

Ağaç büyüdükçe HashSet’e göre performansı azalır.

Map Implementasyonlarının Karşılaştırılması

HashMap

Elemanlarını bir has tablosunda saklar

Elemanların eklenmesi ve bir elemanın aranması sabit zamanlıdır

Elemanlar eklendiği sırayı kaybeder. Eğer bu durumu istemezseniz LinkedHashMap kullanabilirsiniz.

TreeMap

Elemanlarını bir tree yapısında saklar.

Elemanlar sıralı bir şekilde ekrana gelir.

Ağaç büyüdükçe HashMap’e göre performansı azalır.

SORU

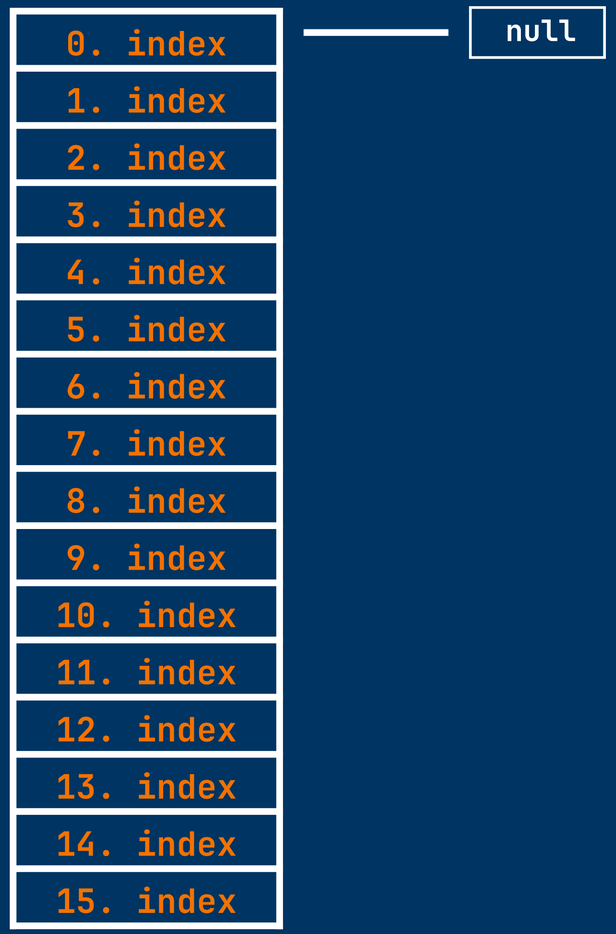
10 tane öğrencinin tutulduğu bir HashMap’te, parametre olarak verilen bir öğrenci kaç adımda bulunur.?

HashMap çalışma Mantığı

HashMap oluşturulduğunda default olarak 16 Haneli bir yapı oluşturulur.

HashMap çalışma Mantığı

Key değerlerine göre hash algoritması çalışır ve bir hashCode üretir. Null değerler 0. İndex’e yazılır.

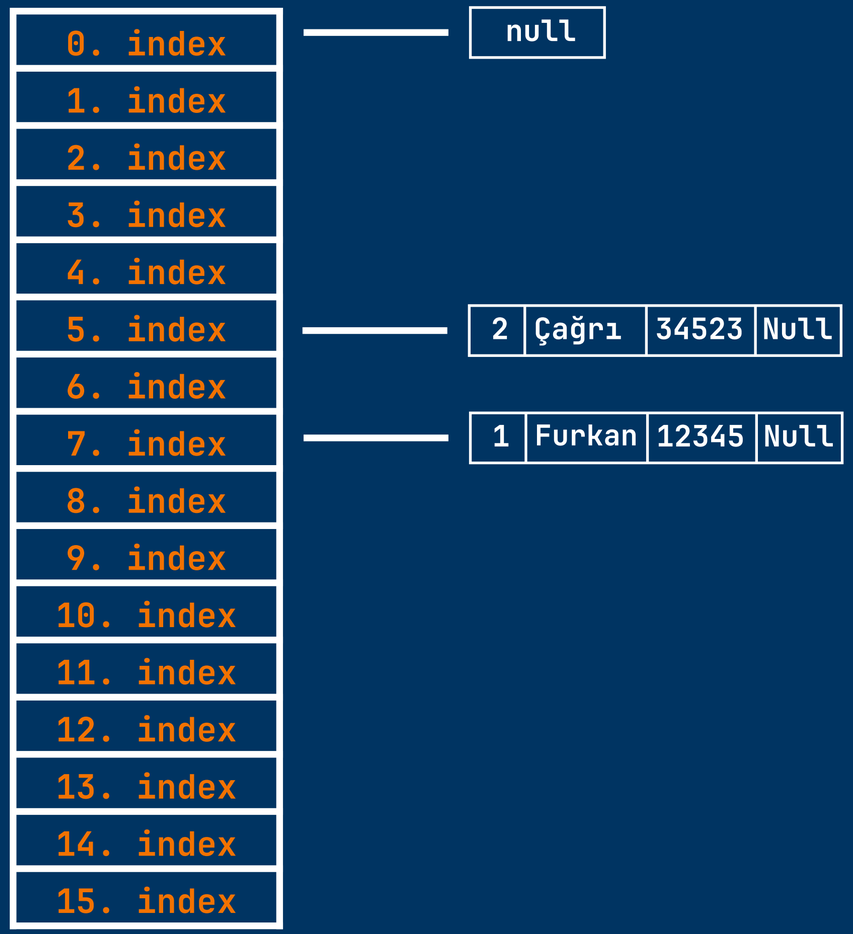


HashMap çalışma Mantığı

Elimizde new Ogr(1, “Furkan”); şeklinde bir öğrenci olsun ve bu öğrencinin hashCode’u 12345 olsun. Bu hashCode değerine ilgili dizi için bir index atanır.
Örn; 7. index

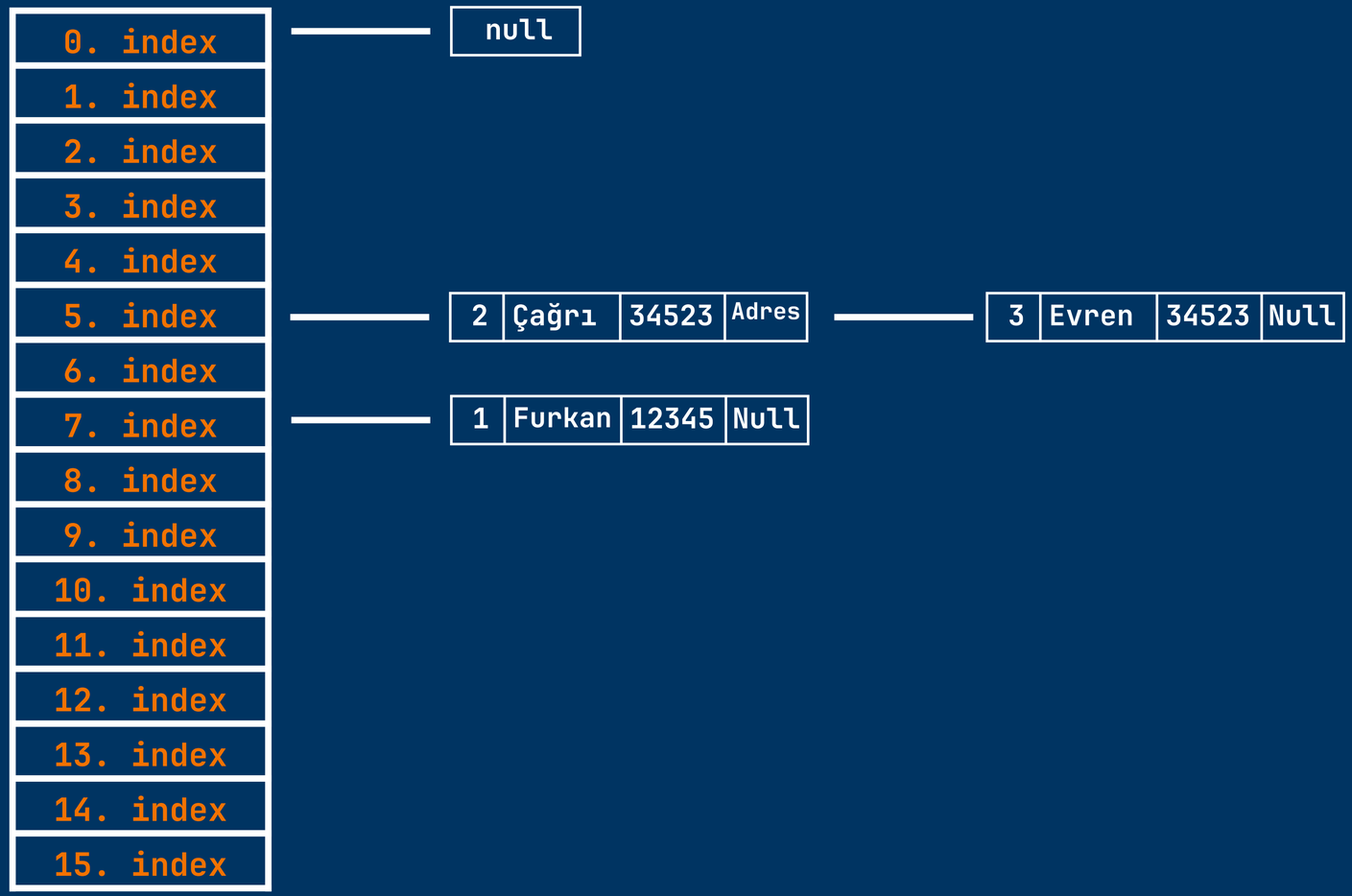
HashMap çalışma Mantığı

İkinci öğrencimizi oluşturalım. new Ogr(2, “Çağrı”); şeklinde bir öğrenci olsun ve bu öğrencinin hashCode’u 34523 olsun. Bu hashCode değerine ilgili dizi için bir index atanır.
Örn; 5. index



HashMap çalışma Mantığı

Üçüncü öğrencimizi oluşturalım. new Ogr(3, “Evren”); şeklinde bir öğrenci olsun ve bu öğrencinin hashCode’u 34523 olsun. Bu hashCode değerine ilgili dizi için bir index atanır.
Örn; 5. index



MapMap vs HashTable

HashMap

HashMap thread safe değildir.

HashMap senkronizasyon olmadığından dolayı daha hızlı çalışır.

HashMap key - value değerlerinin her ikisinde de null değerleri kabul eder.

HashMap eklendiği sıranın yansıtılmasını garanti etmez.

HashTable

Thread safedir.

HashTable senkronizasyon olduğundan dolayı daha yavaş çalışır.

HashMap key - value değerlerinin hiç birinde null değer kabul etmez.

HashTable eklendiği sıranın yansıtılmasını garanti eder.