Collection Framework

Collection Framework, içerisinde interfaceler, classlar ve bir dizi algoritma bulunduran bir yapıdır. Collection Framework içerisindeki classlardan objeler üretir ve bu objeler içerisinde birden fazla elemanı tutabiliriz. Javada birden fazla elemanı depolamak için Arrayler kullanılırdı ancak Arraylerin birtakım problemleri vardır (Boyutun statik olması – ihtiyaca göre artıp azalamaması gibi). Ve yine arrayler yerine arraylerden çok daha iyi olan Vector, Stack, Dictionary, Properties gibi yapılar da vardı. Ancak bunların problemide; hepsinin temel olarak ortak görevleri aynı olmasına rağmen bu görevleri gerçekleştiren methodların birbiriyle bir bağlantısı yoktu. (Eleman ekleme, eleman silme, eleman arama vb.) Bu durumdan ötürü java kodlayıcısı, bu yapılardan türetilen nesneler üzerinden methodları kullanırken kafası karışabiliyordu. İşte bu tür sebeplerden ötürü Collection Framework geliştirilmiş ve jdk 1.2’den sonra javaya eklenmiştir.

Collection Framework, bize bir standart sağlamaktadır çünkü; Collection Framework’ten üretilen nesnelerin çoğu, Collection Interface’inden geldiği için bir sürü method ortak olacaktır.

Not: ‘Collection’ kelimesine Ingilizce bir kelime olarak bakarsak, içerisinde birden çok eleman bulundurabilen yapı demektir ve birden fazla eleman depolayan her şeye collection diyebiliriz. (Array, Map, ArrayList vb.)

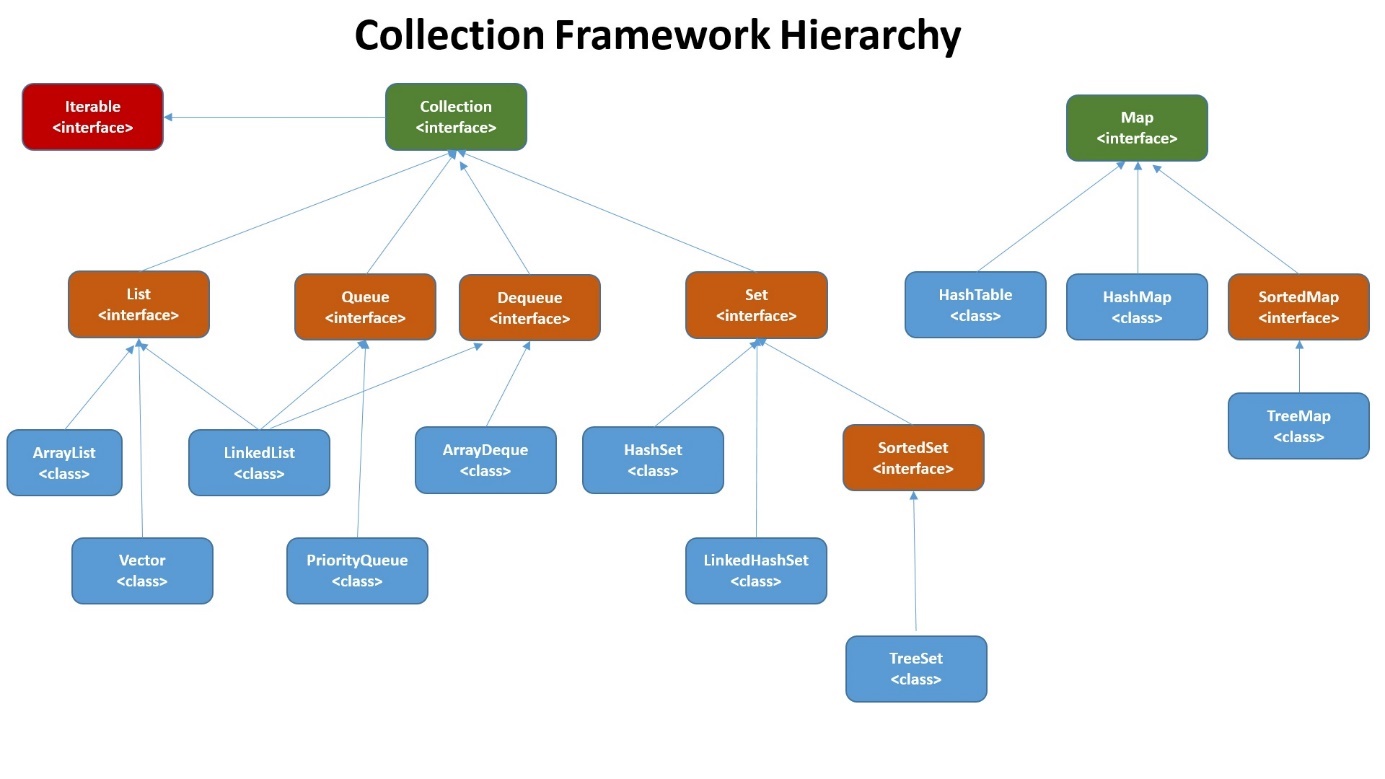
Ancak Java tarafından ‘Collection’ kelimesine bakarsak, Collection interface’inden gelen objelere Collection denir. (Array ve Map Collection interface’inden gelmediği için Collection değildir. Ancak ArrayList bir Collection’dır.)

Collection Framework:

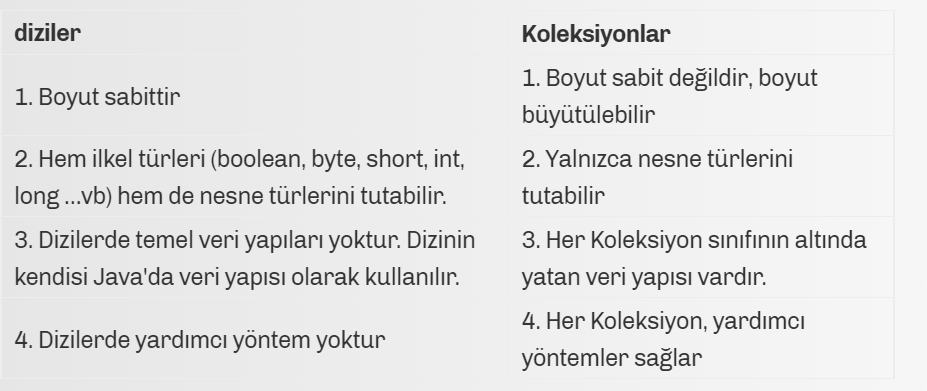
Bunları ezberlemenize gerek yok ama Collection Framework ana 4 interfaceden oluşur.

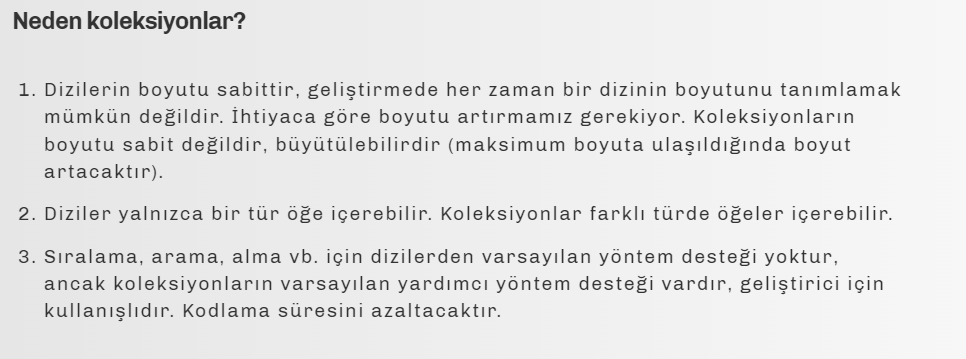
1. List -> Nesneleri liste olarak saklar, Arraylist,Vector ve LinkedList
2. Queue -> İlk giren son çıkar prensibi ile elemanları saklar. Bu interfaceyi gerçekleştiren sınıflar LinkedList ve PriorityQueue
3. Set -> Elemanları unique yani tekrarsız olarak saklar. HashSet, LinkedHashSet ve SortedSet somut sınıflarını kullanırız.
4. Map -> Bu interface Collection interfacesinden türetilmemiştir. Elemanlarını anahtar-değer ilişkisi ile saklar. Hashmap ve HashTable somut sınıfları ile TreeMap sınıfları bu interfaceyi implemente eden somut sınıflardır.

!!Önemli: List, Queue ve Set, Collection interface’inden gelirken, Map, Collection interfacesinden gelmemektedir.



Arrays Vs Collections:





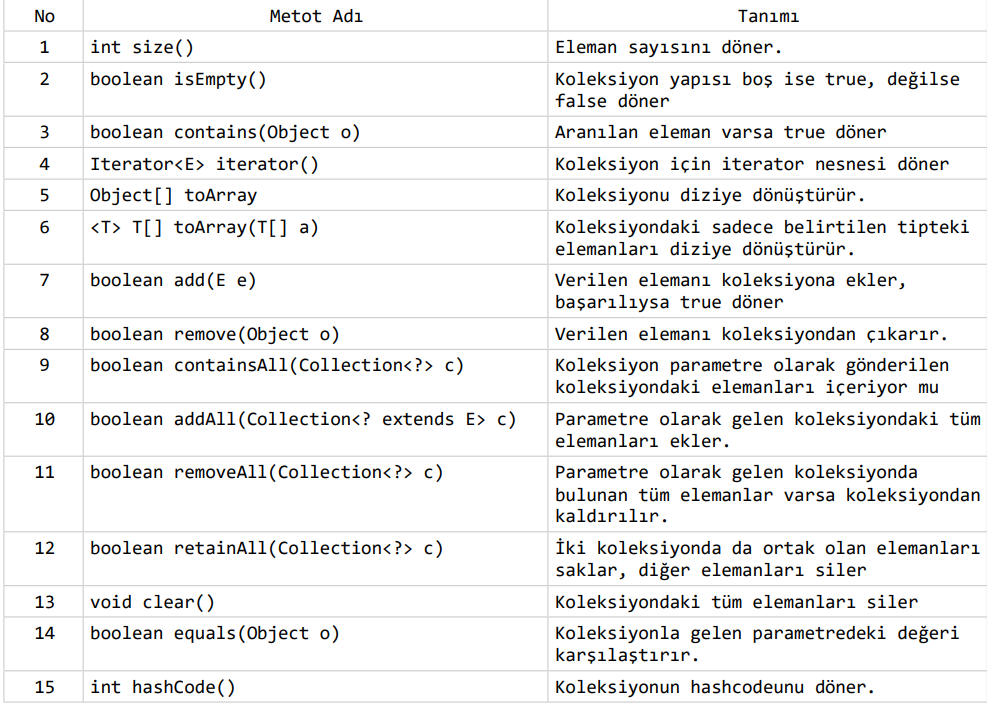
Not: Collection’lar, Array’lerin dezavantajlarını ortadan kaldırırlar ancak bu her yerde Array yerine Collection kullanmamız anlamına gelmez. Birbirlerine göre avantajları ve dezavantajları vardır. Uygulamamızda hangisi kullanacağımıza bunları düşünerek karar vermeliyiz.

Not: Collectionlarda primitive tipten veriler tutamayız ancak bu tiplerin wrapper class tiplerinden objeleri tutabiliriz.

Collection Interface

Bu interface tüm collection frameworkünün kök interfacesidir ve bu da Iterable interfacesinden kalıtılmıştır. Bundan dolayı iterator() isimli metota sahiptir. Bu metot geriye Iterator nesnesi döndürür ve biz bunu kullanarak koleksiyonumuzdaki elemanları gezebiliriz. Map neden bu interfaceyi implemente etmemiş? Çünkü değerler indekse göre değil anahtara göre saklanıyor.

Burada pek çok metot tanımı vardır ve bunlar farklı koleksiyon yapıları için de ortaktır (Map hariç).



Buradaki hashcode ve equals metotları Object sınıfındakiler değildir çünkü interfacelerin üst sınıfı olamaz.

List Interface

List interface sıralanmış veya sıralanmamış elemanları tutan yapıdır. Bu interfacenin bazı metotları sıralanmış elemanlar içindir. Bu interfaceyi gerçekleştiren somut sınıflar Arraylist, Vector ve LinkedListtir.

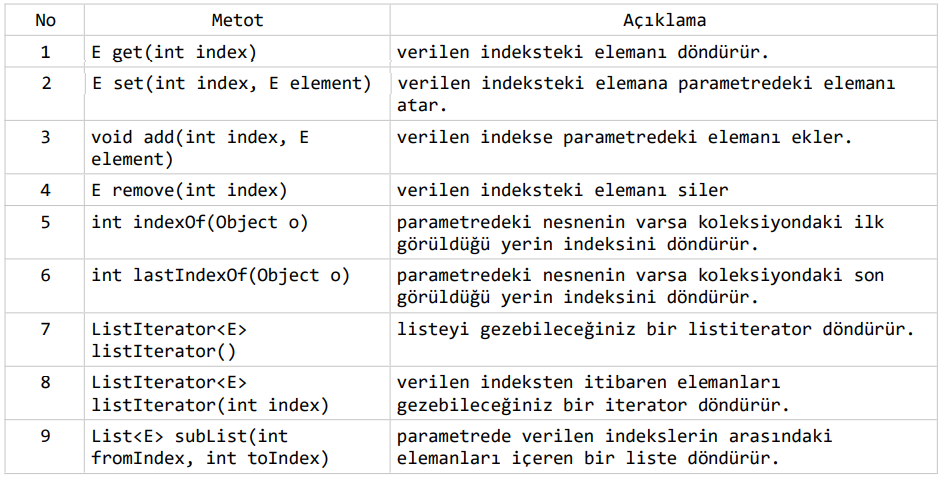
• Elemanlar index 0 dan baslanarak listelenir.

• Elemanlara integer indexler ile erişebilirsiniz. Istenilen indexe eleman yerleştirebilirsiniz. mesela 3. indekse eleman eklerseniz sonrasındaki elemanlar sağa doğru otomatik kaydırılır.

• Istenilen indeksten eleman silebilirsiniz. Mesela 3. indeksten eleman silerseniz sonrasındaki elemanlar sağa doğru otomatik kaydırılır.

• Bu listede tekrar eden eleman ve null değerler olabilir.

• List interface collection interfacesindeki tüm metotlara sahip olup kendine özgü bazı metotları da barındırır.



ArrayList:

* Arraylist boyutu dinamik olan ve hiyerarşik olarak list interface’i altında bulunan bir veri yapısıdır.

ArrayList Verileri Nerede Saklar?

* Arraylist arkaplanda bir Object[] dizisi üzerinde verileri saklar.

ArrayList nesnesi oluşturulduğunda arkaplanda neler olur?

Javada arraylist sınıfı bir arraylist oluşturmak için 3 constructor sağlar.

1. public ArrayList(int initialCapacity): Bu constructor cagrildiğinda initialCapacity = 0 ise arkaplanda boş bir dizi oluşturulur. Eğer initialCapacity > 0 ise arkaplanda bu parametre değeri boyutunda bir dizi oluşturulur. Eğer initialCapacity < 0 ise bir exception fırlatır.
2. public ArrayList(): Bu constructor cagrildiğinda arkaplanda boş bir dizi oluşturulur.
3. Public ArrayList(Collection <? extends E> c): Parametredeki collection boş ise arkaplanda boş bir dizi oluşturur. Boş değil ise arkaplanda bu collection boyutunda bu verileri tutacak bir dizi oluşturur. Colleciton’daki elemanlar da bu diziye atanır.

ArrayList Dinamik Olarak Nasıl Büyür?

* Eğer arraylist için arkaplandaki dizi boş bir dizi ise ve biz arrayliste ilk elemanı atamak istiyorsak arkaplandaki diziye new Object[Default capacity(yani 10)] komutunu atar.

Yani arkaplandaki boş diziyi 10 elemanlık bir dizi yapar.

* Eğer arraylist için arkaplandaki dizi boş bir dizi değilse arkaplandaki diziye eski kapasitesinin %50 daha fazla kapasiteli bir dizi atar ve eski dizinin elemanlarını bu diziye kopyalar.

Yük Faktörü:

Yük faktörü, ArrayList kapasitesinin ne zaman artırılacağına karar veren ölçüdür. ArrayList'in varsayılan yük faktörü 0.75f'dir. Örneğin, mevcut kapasite 10'dur. Yani yük faktörü = 10\*0.75=7 ‘dir.

7. eleman eklenirken dizi boyutu artacaktır. Bu nedenle, beklenen elemanların sayısını yaklaşık olarak akılda tutarak ilk kapasiteyi seçmemiz iyi bir uygulama olacaktır.

ArrayListten Bir Eleman Kaldırıldığında Ne Olur?

Bir eleman kaldırıldığında o elemanın indexinden daha büyük indexteki değerler sola doğru 1 kutu kaydırılır. Bu Arrays.copy methoduyla yapılır. (Arrays.copy methodu, diziyi farklı bir diziye kopyalamaz yine aynı diziye kopyalar ve time complexity’i O(n)’dir.)

Dikkat Edilmesi Gereken Noktalar:

1. Java'daki ArrayList, List arabiriminin Yeniden Boyutlandırılabilir dizi uygulamasıdır.
2. Dahili olarak ArrayList sınıfı, öğelerini depolamak için bir Object sınıfı dizisi kullanır.
3. Bir ArrayList'i başlatırken, başlangıç ​​kapasitesi sağlayabilirsiniz, bu durumda dizi, başlangıç ​​kapasitesi olarak sağlanan boyutta olacaktır.
4. Başlangıç ​​kapasitesi belirtilmemişse, bir dizi oluşturmak için varsayılan kapasite kullanılır. Varsayılan kapasite 10'dur.
5. ArrayList'e bir eleman eklendiğinde, önce yeni elemanı barındırıp barındıramayacağını veya büyümesi gerekip gerekmediğini doğrular, kapasitenin arttırılması gerekiyorsa, eski kapasiteden %50 daha fazla olan yeni kapasite hesaplanır ve dizi bu kadar kapasite artırıldı.
6. Öğeler, bir öğenin kaldırılmasıyla oluşturulan bir ArrayList alanından kaldırıldığında, alttaki dizide doldurulmalıdır. Bu, sonraki öğelerin sola kaydırılmasıyla yapılır.

ArrayList Vs Array:

Arraylerde elemanlar primitive tipdense bellekte bitişik konumda depolanırlar. Elemanlar object tipindense bellekte bitişik olarak depolanan objelerin kendileri değil referanslarıdır.

Arraylistlerde elemanlar primitive tip olamayacağından dolayı bellekte objelerin kendileri değil referansları bitişik olarak depolanırlar.

• Arraylist, arraylerde olduğu gibi sabit bir boyuta sahip değildir.

• Dinamik olarak kapladığı alan arttırılıp azaltılabilir. Istenilen indekse eleman ekleyip, çıkarılabilir. Sağa veya sola kaydırmalar otomatik olarak yapılır.

• Arraylist sınıfında çoğu zaman ihtiyaç duyacağımız faydalı metotlar zaten var olduğu için kendimizin bu özellikleri yazmasına ihtiyaç yoktur.

• Eğer generik tip belirtmezsen arraylistte farklı türde veriler saklayabiliriz.

• Bazıları arrayin arrayliste göre çok daha hızlı çalıştığını düşünebilir. Sonuçta ne kadar yer kaplayacağı, ne için yer kaplayacağı bellidir. Ama bunu test edersek aslında o kadar büyük bir farkın olmadığı görülür

• Arraylist tek boyutludur ancak diziler çok boyutlu olabilir.

Iterator Interface:

* Collection interface’inin extends ettiği interface’dir.
* Iterable edilebilen (üzerinde gezenebilinen) objeler için bir iterator objesi sağlar. Bu iterator objesi ile iterable objeler üzerinde gezinebiliriz.
* Java 1.2’den sonra javaya katılmıştır.
* Collection Framework’ün tüm classları için kullanılabilir olduğundan ‘Universal Cursor of Java’ (Java evrensel imleci) olarak bilinir.
* 4 tane methodu vardır:

1. boolean hasNext(): Iterator, null değeri gösteriyorsa false, null değeri göstermiyorsa true return eder.
2. E next(): Iterator’ı, iterator’ın gösterdiği element üzerinden zıplatır ve üstünden geçtiği elementi return eder. Iterator null değeri gösteriyorken bu method çağrılırsa NoSuchElementException hatası fırlatır.
3. default void remove(): next() yönteminden return edilen son değeri collection’dan siler.

next() methodu çağrılmadan önce bu method çağrılırsa IllegalStateException fırlatır.

Her next() methodu çağrısı başına bu method bir kere çağrılabilir.

Kaldırma işlemi desteklenmiyorsa da UnSupportedOperationException exception fırlatır.

1. Default void forEachRemaining(Consumer action): Parametre olarak gönderdiğimiz eylemi, collection’daki tüm elemanlar üzerinde gezerek gerçekleştirir.

Eğer eylem null ise NullPointerException fırlatır.

ListIterator Interface:

* Sadece List Interface’inden gelen objeler için kullanabiliriz.
* Iterator interface’ini extends ederek iterator’ın daha kullanışlı bir halini sunar.
* (collection\_name).listIterator(size) diyerek listIterator’ı listenin sonunda konumlandırarak başlatabiliriz.
* Iterator interface’inde bulunan bütün methodlara sahip olmakla beraber ayrıyeten kendine özgü 6 methodu daha bulunur.

1. boolean hasPrevious(): ListIterator’ın arkasında gösterdiği değer varsa true return eder. Yoksa false return eder.
2. int nextIndex(): next() methodu çağrıldığında return edilecek elemanın indexini return eder. Eğer ListIterator listenin sonundaysa (null değeri return ediyorsa) listenin boyutunu return eder.
3. İnt previousIndex(): previous() methodu çağrıldığında return edilecek elemanın indexini return eder. Eğer ListIterator listenin başındaysa -1 return eder.
4. E previous(): ListIterator’ın arkasında gösterdiği elemanın üzerinden geçerek bu elemanı return eder. Eğer ListIterator’ın arkası null değeri gösterirken bu method çağrılırsa NoSuchElementException fırlatır.
5. void set(E e): next() veya previous() methodu tarafından son dönderilen öğenin değerini değiştirmek için kullanılır.

Not: set() methodu, remove() ve add() methodu çağrılmadığı sürece çağrılabilir.

Eğer parametre olarak gönderilen değer, Liste elemanları tipinde değilse

ClassCastException fırlatır.

Eğer next() veya previous() methodu çağrılmadan set() methodu çağrılırsa IllegalStateException fırlatır.

Eğer parametre olarak verilen öğenin bazı özellikleri, öğenin listeye eklenmesini engellerse IllegalArgumentException fırlatır.

Eğer set işlemi, ListIterator tarafından desteklenmiyorsa

**UnsupportedOperationException** fırlatır.

1. Void add(E e): Parametre olarak aldığı öğeyi, ListIterator’ın gösterdiği elemanın öncesine ekler. Ekleme sonrası ListIterator’ın yeri değişmez. Yani ekleme olmadan önce gösterdiği elemanı göstermeye devam eder. Ama ListIterator, listenin sonundayken eleman eklenirse, bu eleman listenin sonuna eklenir ve ListIterator’ında yeri değişir. ListIterator’ın yeni yeri listenin sonu olur.

Eğer parametre olarak gönderilen değer, Liste elemanları tipinde değilse

ClassCastException fırlatır.

Eğer parametre olarak verilen öğenin bazı özellikleri, öğenin listeye eklenmesini engellerse IllegalArgumentException fırlatır.

Eğer add işlemi, ListIterator tarafından desteklenmiyorsa

**UnsupportedOperationException** fırlatır.

Iterator Vs ListIterator:

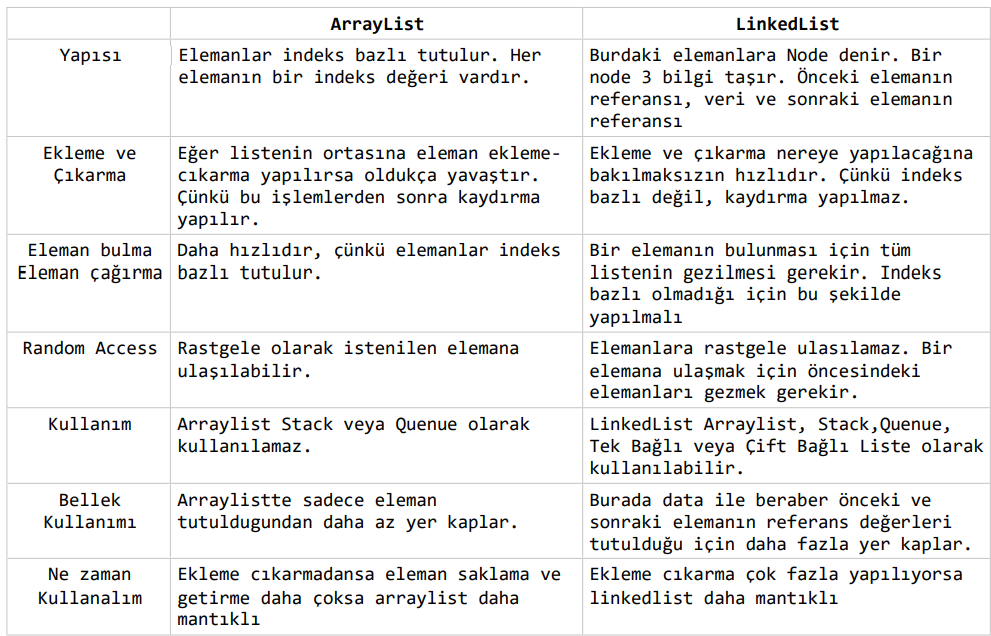
ListIterator, Iterator interface’ini extends eden bir interface’dir.

1. Iterator ve ListIterator arasındaki temel fark, her ikisinin de imleç olduğu için Iterator'ın bir koleksiyondaki öğeleri yalnızca ileri yönde geçebilmesidir. Öte yandan, ListIterator hem ileri hem de geri yönlerde hareket edebilir.
2. Iterator kullanarak bir koleksiyona herhangi bir öğe ekleyemezsiniz. Ancak ListIterator kullanarak bir koleksiyona öğeler ekleyebilirsiniz.
3. Iterator kullanarak Harita, Liste, Küme gibi tüm koleksiyonları gezebilirsiniz. Ancak ListIteror ile yalnızca List uygulanan nesneleri arasında geçiş yapabilirsiniz.
4. Iterator kullanarak bir öğenin indexini alamazsınız. Ancak Liste sıralı ve index tabanlı olduğundan, ListIterator kullanarak bir öğenin indexini alabilirsiniz.

LinkedList:

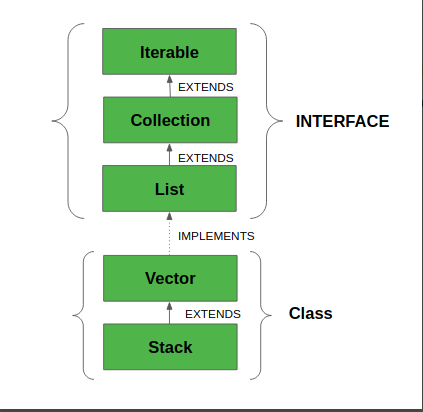
* Java arkaplanda doubly linkedlist kullanır.
* Linkedlist node’lardan oluşur ve node’ların data değeri, next değeri ve prev değeri vardır.
* İlk node’u göstermek için bir first pointer ve son node’u göstermek için bir last pointer bulunur.
* Ekleme ve silme işlemleri ArrayList ve Array’e göre daha kolaydır.
* Eleman arama işlemi ArrayList ve Array’e göre daha zordur.

ArrayList Vs LinkedList:



Vector Sınıfı:

* Vector sınıfı, arraylist sınıfında olduğu gibi arkaplanda bir dizi tutar.
* Boyut dinamik olarak büyüyüp küçülebilir.
* Vector javanın eski sınıflarından biridir. Ancak Collection Framework ile de uyumludur.
* List interface’ini impelement ettiği için oradaki methodlar bu sınıftan üretilen nesneler içinde kullanılabilirdir.
* ArrayList’e çok benzer ancak buradaki methodlar senkronizedir. Ve Collection Framework’ün içermediği bazı eski methodlara sahiptir.
* Senkronize olduğu için, thread’lerin olmadığı yerlerde nadiren kullanılır.



Önemli Not: Vector, arrayliste çok benzerdir ancak vectoru arralistten ayıran en önemli fark buradaki methodların senkronize olmasıdır.

Senkronize: Bir thread bir yapı üzerinde işlem yaparken diğer threadlar bu yapı üzerinde işlem yapamaz. İlk threadin işini beklemek zorundadırlar.

* Vector sınıfı tarafından dönderilen iterator’lar fail-fast’tir.

**Vektör kapasitesi artışı ile ilgili önemli noktalar şunlardır:**

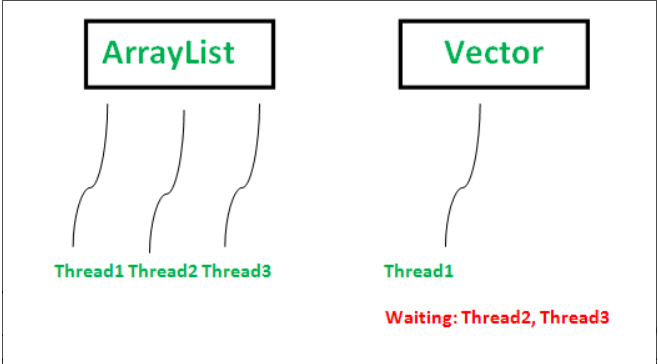
* Kapasite için artış belirtilirse vectorun eleman sayısı kapasiteyi geçtikçe bu artış miktarı kadar kapasite artırılır.
* Kapasite için artış belirtilmezse vectorun eleman sayısı kapasiteyi geçtikçe kapasite iki katı olacak şekilde ayarlanır.
* Bir vektörün kapasitesi, vektörün eleman sayısından düşük olamaz. Eleman sayısıyla eşit veya eleman sayısından büyük olabilir.

Constructors:

* 1. **Vector():** Başlangıç ​​kapasitesinin 10 olduğu varsayılan bir vektör oluşturur.
  2. **Vector(int size):** Başlangıç ​​kapasitesi belirtilen size kadar olan bir vektör oluşturulur.
  3. **Vector(int size, int incr):** Başlangıç ​​kapasitesi belirtilen size kadar ve kapasite artışıda belirtilen incr kadar olan bir vektör oluşturulur.
  4. **Vektör(Koleksiyon c):** c koleksiyonunun öğelerini içeren bir vektör oluşturur.

ArrayList Vs Vector:

1. Vektör **senkronizedir**. Yani bir seferde yalnızca bir thread koda erişebilirken arrayList **senkronize edilmez.** Bu da birden fazla iş parçacığının aynı anda arrayList üzerinde çalışabileceği anlamına gelir.

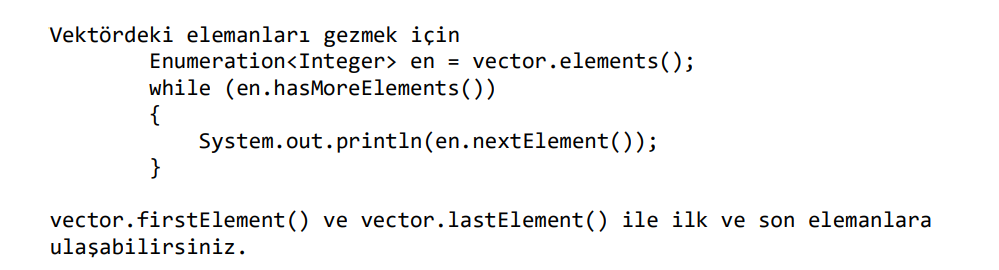


1. **Performans: ArrayList** senkronize olmadığı **için daha hızlıdır** , vektör işlemleri ise senkronize olduklarından daha yavaş performans verir. Bir iş parçacığı bir vektör üzerinde çalışıyorsa, üzerinde bir kilit elde etmiştir, bu da üzerinde çalışmak isteyen diğer herhangi bir iş parçacığını kilit serbest bırakılana kadar beklemeye zorlar.
2. **Veri Büyümesi:** ArrayList ve Vector , depolamanın optimum kullanımını sağlamak için **dinamik** olarak **büyür ve küçülür** - ancak yeniden boyutlandırma biçimleri farklıdır. ArrayList, eleman sayısı kapasitesini aşarsa mevcut dizi boyutunu %50 artırır, vektör ise %100 artar - esasen geçerli dizi boyutunu iki katına çıkarır.
3. [ArrayList, Collections.synchronizedList kullanılarak açıkça senkronize edilebildiğinden](https://www.geeksforgeeks.org/synchronization-arraylist-java/) , programcılar çoğu zaman Vector yerine ArrayList'i tercih eder .

**ArrayList ve Vector arasında nasıl seçim yapılır?**

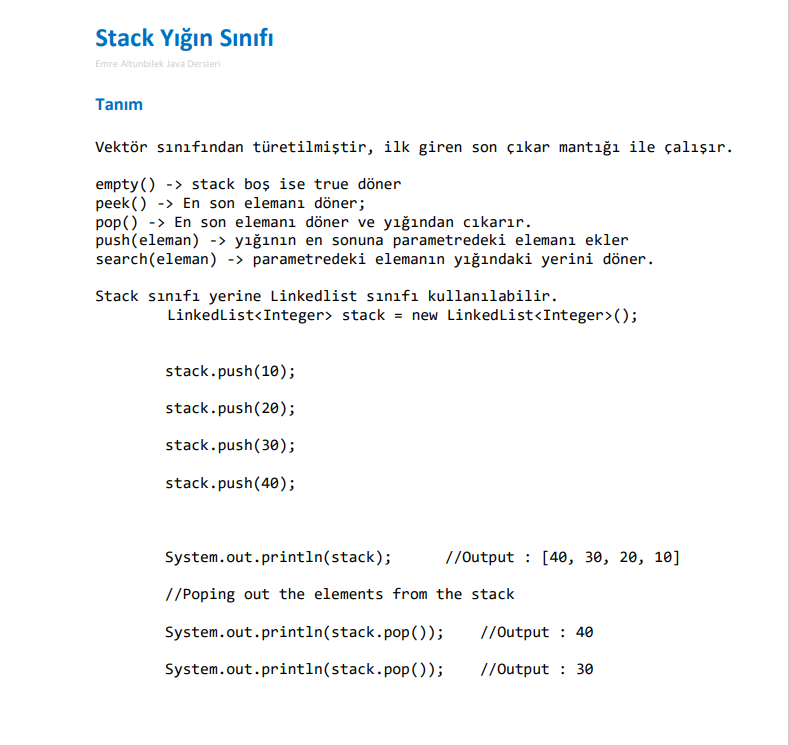
* ArrayList senkronize değil ve iş parçacığı için güvenli değil, oysa Vektörler. Bir Vector üzerinde aynı anda yalnızca bir iş parçacığı yöntemleri çağırabilir, bu biraz ek yüktür, ancak güvenlik bir endişe olduğunda yardımcı olur. Bu nedenle, tek iş parçacıklı bir durumda, arraylist bariz bir seçimdir, ancak çoklu iş parçacığı söz konusu olduğunda, vektörler genellikle tercih edilir.
* Ne kadar veriye sahip olacağımızı bilmiyorsak, ancak büyüme hızını biliyorsak, Vektör'ün bir avantajı vardır, çünkü artış değerini vektörlerde ayarlayabiliriz.
* ArrayList daha yeni ve daha hızlı. Bunlardan herhangi birini kullanmak için herhangi bir açık gereksinimimiz yoksa, vektör üzerinde ArrayList kullanırız.

Arraylist sınıfı gibi dinamik olarak artan bir dizi gibidir. Arraylistten en temel farkı vector sınıfı synchronized metotlara sahiptir yani vector üzerinde sadece tek bir thread işlem yapar, diğer threadler burdaki işlemin bitmesini beklemek zorundadır. Multi thread bir ortamda vector kullanımı daha mantıklıdır. Tabi bu da beraberinde performans sorunları getirebilir. Çünkü bir thread işlem yaparken vector üzerine kilit koyar ve diğer threadler bu işlemin yapılmasını beklemek zorunda kalır. Eğer multi thread bir uygulama yapmıyorsak arraylisti tercih etmek daha mantıklı olacaktır.

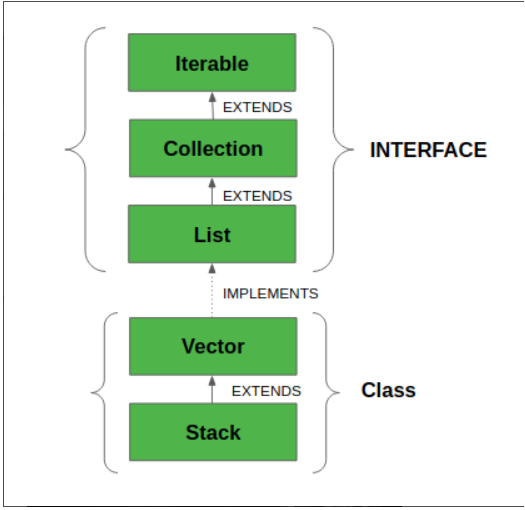


Bunların dışında vektör kullanımı çok önerilmez, peki neden ?

1. Collections.synchronizedList(arraylist) komutu ile Thread safe özelliğini arraylistlere de kazandırabiliriz.
2. Vektörlerdeki thread safe özelliği çok fazla zaman alır. Eski bir sınıftır, eksi özellikleri çoktur o yüzden arraylist kullanalım.



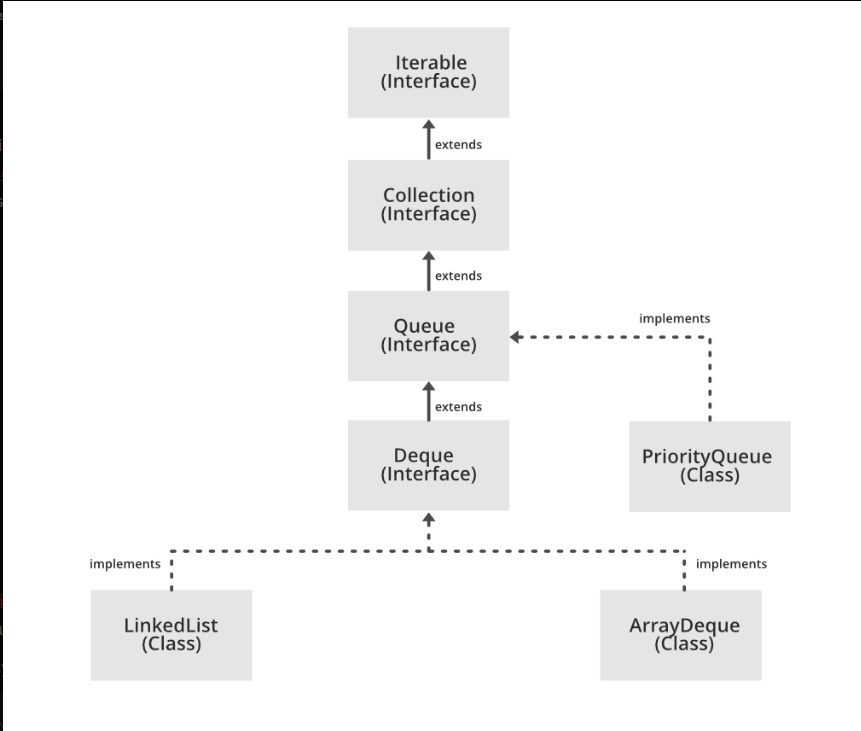
* Stack sınıfı, vector sınıfını extends ettiği için list interface’indeki bütün methodlara sahiptir.



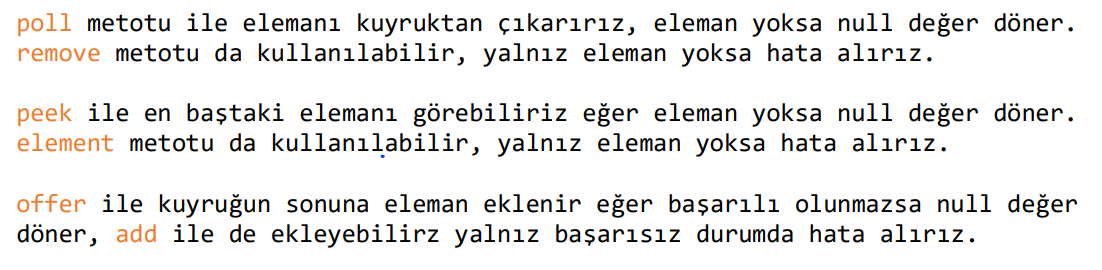
* Tek constructor’ı vardır ve o da parametre almaz.
* Stack sınıfı yerine, LinkedList sınıfını kullanabiliriz.

Queue Interface:

* Direkt olarak Collection interface’ini extends eden bir interface’dir.



* First In First Out prensibiyle çalışır.
* Queue mantığını kullanmak istiyorsak, kullanabileceğimiz somut classlar; LinkedList ve ArrayDeque classlarıdır.
* Queue interface’i, Collection interface’inden geldiğinden dolayı buradaki methodlara sahiptir. Ancak biz FIFO prensibiyle çalışmasını istediğimiz zamanlarda Collection interface’inden gelen methodları değilde Queue interface’inin kendine özgü methodlarını kullanmalıyız.



PriorityQueue:

* Queue interface’ini implement eden bir classtır.
* Torpilli queue’dur. FIFO prensibiyle çalışır ancak elemanlar bir önceliğe göre sıralanır.
* Karşılaştırılamayan objelerin PriorityQueue’sunu oluşturamayız.

metin içeren bir resim

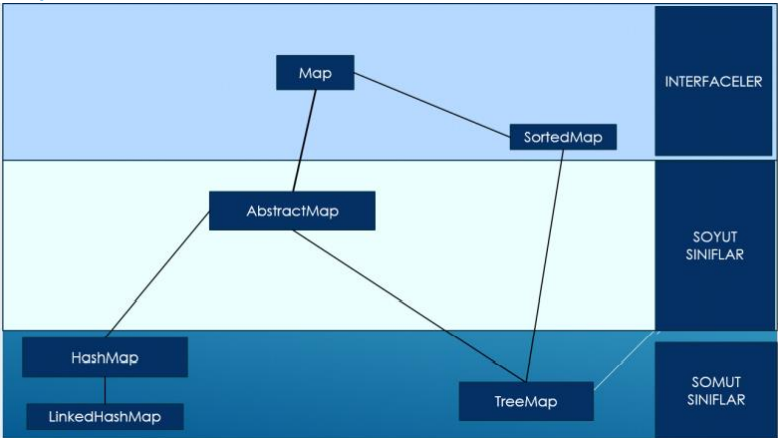
Açıklama otomatik olarak oluşturuldu

Object – String – Wrapper Classlarındaki HashCode() Nasıl Çalışır?

* Arka planda nesneyi bir hash fonksiyonuna sokarak nesnenin int bir hash kodunu return eder.
* Equals() methodunun içeriği değiştirilmediği sürece birden çok hashcode() çağrısı aynı değeri return etmelidir.
* HashCode() metohdu, uygulamayı çalıştırdığımız farklı zamanlarda farklı değerler return edebilir.
* Equals() methodu, true return ediyorsa; iki objenin hashcode değerleri aynı olmalıdır.
* Equals() methodu, false return ediyorsa; iki objenin hashcode değerlerinin farklı olması gerekmez. Aynı veya farklı olabilir.
* Yani; iki objenin hashcode’u aynı ise, iki obje aynı obje demek değildir. (equals methodu true dönmek zorunda değil.) Ancak iki obje eşitse (equals methodu true return ediyorsa) bu objelerin hashcode’ları aynı olmak zorundadır.

! Object, String ve Wrapper classlar için; varsayılan haliyle equals methoduna göre eşit olan objelerin hashcode’ları aynıdır, equals methoduna göre farklı olan objelerin hashcode’ları muhtemelen farklıdır ancak aynı da olabilir.

Map Interface:



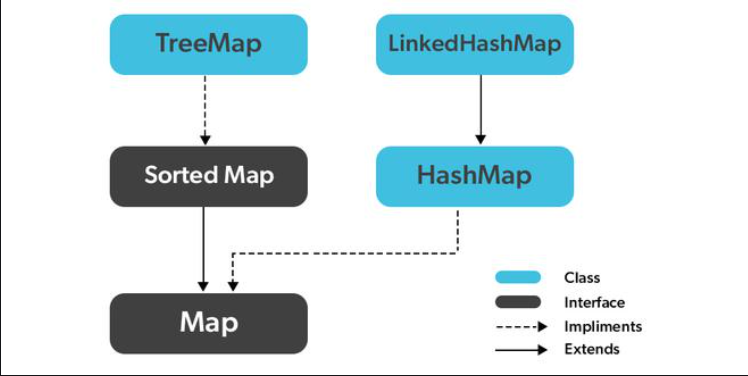
* Map interface’i Collection interface’inin bir alt interface’i değildir.
* Key – value arasında bir eşleşmeyi temsil eder.
* Key değerleri unique(tekil – eşşiz) olamalıdır. Yani bir map yapısında aynı key değerine sahip birden fazla eleman olmaz.
* Value değerleri unique değildir. Bir map’de aynı value değerine sahip birden fazla öğe bulunabilir.
* Map interface’i içinde Entry denen inner interface vardır. Key – value değerlerinin ikisine birden Entry denir.
* Entry’ler arkaplanda kovalarda (buckets) tutulur. Başlangıç bucket sayısı 16’dır.
* Yük faktörü varsayılan olarak 0.75’dir. 16\*0.75 = 12’dir. Bu yüzden HashMap’e 12 entry eklendikten sonra kova(bucket) sayısı 2 katına çıkar ve yeniden hashing işlemi yapılır. Sonra bu durum böyle devam eder. (ReHasing.)

Neden Kullanılır?

* Genel olarak değerleri bir anahtar ile tutmak istediğimizde kullanırız.
* Hata kodlarının ve açıklamalarının haritası.
* Posta kodları ve şehirlerin bir haritası. (Plaka - Sehir)
* Yöneticilerin ve çalışanların haritası. Her yönetici (anahtar), yönettiği bir çalışan (değer) listesi ile ilişkilendirilir.
* Sınıfların ve öğrencilerin haritası. Her sınıf (anahtar) bir öğrenci listesi (değer) ile ilişkilendirilir.

Map Nesnesi Oluşturma:

* Map bir interface olduğundan dolayı map tipinde bir obje oluşturulamaz.
* Map interface’inden türeyen somut sınıflar kullanılarak obje oluşturmamız gerekir ki bunlar; HashMap, LinkedHashMap, TreeMap’dir.



HashMap:

* Key değerleri aynı olan farklı iki eleman eklemeye çalışırsan java hata vermez ancak her iki öğeyide map’e eklemez. Sadece son eklediğin öğe map’e eklenir. Yani hashMap’de key değerleri unique – tekrarsızdır.
* Key değeri null olan öğeleri hashMap’e ekleyebilirsin.
* Öğeler map içerisinde hash code’larına göre sıralanarak tutulur. Yani map’i print edersen, ilk eklediğini ilk sırada görmek yerine hash code’u küçük olanı ilk sırada görürsün.
* Key ve Value değerlerinin her ikisine birden Entry denir.

Constructors:

* 1. HashMap (): Başlangıç kapasitesi 16 ve yük faktörü 0.75 olan bir map oluşturur.
  2. HashMap (int initialCapacity): Başlangıç kapasitesi parametre olarak giden değer ve yük faktörü 0.75 olan bir map oluşturur.
  3. HashMap (int initialCapacity, float loadFactor): Başlangıç kapasitesi ve yük faktörü parametre olarak giden değer olan bir map oluşturur.
  4. HashMap (Map map): Load factor’ü 0.75 olan ve kapasitesi argüman olarak gönderilen map objesini tutabilecek düzeyde olan bir map oluşturur.

Not: HashMap’lerde, eşit olmayan objelerin hashcode’larının farklı üretilmesi, eleman arama konusunda performansı daha iyi yapar.

Not: key değeri null olan öğeler, arkaplandaki dizinin 0. indexinde tutulur.

Not: Eğer yeni bir class oluşturur ve bu classın equals methodunu override edersek hashCode methodunuda override etmemiz gerekir. Aksi halde eşit nesnelerin hashcode’ları eşit olmayacaktır. Buna dikkat etmezsek hashMap

doğru çalışmayacaktır.

HashMap Sınıfı ArkaPlanda Calisma Mantığı(ÖNEMLİ):

1. <https://www.geeksforgeeks.org/internal-working-of-hashmap-java/>
2. https://medium.com/javarevisited/internal-working-of-hashmap-in-java-97aeac3c7beb#:~:text=Internally%20HashMap%20uses%20a%20hashCode,entries%20(nodes)%20are%20stored.
3. https://medium.com/java-m%C3%BClakat-sorular%C4%B1-2-4-y%C4%B1l-tecr%C3%BCbeli-adaylar/java-hashmap-nas%C4%B1l-%C3%A7al%C4%B1%C5%9F%C4%B1r-24ebafbd3450

LinkedHashMap:

* HashMap classını extends etmiş bir classtır.
* HashMap’in neredeyse aynısıdır. Aradaki fark: LinkedHashMap’de elemanlar ekrana basıldığında eklendiği sırada görülür.

Arka Planda Nasıl Çalışır?

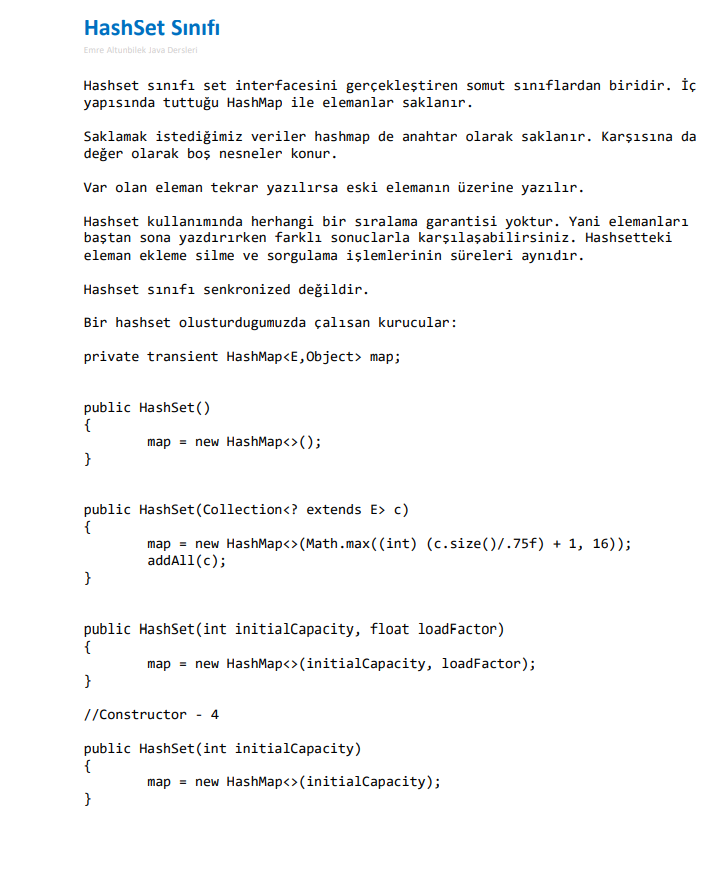
* HashMap’in çalışma mantığının aynısıdır. Ancak burada bir fark vardır. O da elemanları eklediğimiz sırada almamızdır. Arka plandaki çalışma farkı sadece bundan kaynaklanır.
* HashMap’den farklı olarak Head ve Tail adlı iki pointer ve bucket’lara yerleşen node’ların içinde before, after adında iki pointer daha olması.
* LinkedHashMap’e ilk eleman eklendiğinde head ve tail bu elemanı gösterir. İlk elemanın before ve after’ı null’u gösterir.
* İkinci eleman eklendiğinde tail bu elemanı gösterirken head ilk elemanı göstermeye devam eder. İlk elemanın after’ı yeni eklenen elemanı gösterir. İkinci elemanın before’u ilk elemanı ve after’ı null’u gösterir.
* Üçüncü eleman elendiğinde tail bu bu elemanı gösterirken head ilk elemanı göstermeye devam eder. İkinci elemanın after’ı üçüncü elemanı gösterirken üçüncü elemanın before’u ikinci elemanı gösterir. Ve bu işlem böyle devam eder….
* Elemanları okumak istediğimizde de head pointer’ın işaret ettiği düğümden başlayarak node’lar üzerindeki after pointer’ının işaret ettiği alanlar sırasıyla okunur ve bu sayede elemanlar sıralı bir şekilde alınmış olur.
* Bunun dışında bütün her şey HashMap’dekiyle aynıdır.

TreeMap:

* Arka planda Red Black Tree kullanır.
* Null key’lere izin vermez.
* Elemanları bizim istediğimiz sırada alırız.
* Elemanları istediğimiz sırada almak için Comparable ve Comparator interface’lerini kullanır.

Set Interface:

* Elemanları tekrarsız olarak saklamak için kullanılır.
* Collection interface’inden gelmiştir.
* Arka planda HashMap’i kullanır.
* Set yapıları key-value şeklinde değer tutmaz ancak arka planda HashMap kullanıyorsa value olarak ne atıyor? Cevap: Boş bir obje.





metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu