Collection Framework

Sınıf Düzenleri

Collections dediğimiz yapılar aslında bir grup nesneyi bir arada tutan yapılardan başka bir şey değildir. Bu yapılardan önce dizileri bu amaçla kulllanırdık. Ama dizilerin boyutları sabittir ve bir kere tanımlandıktan sonra değiştirilemez. Bundan dolayı collection frameworkü geliştirilmiş ve jdk 1.2den sonra hayatımıza girmiştir.

Aslında diziler gibi Dictionary,Vector,Stack ve Properties gibi dizilerden çok daha iyi olan yapılarımız vardı ama bu yapıların her biri farklı şekillerde çalışır ve kullanım şekil ve amaçları birbirinden farklıdır. Bu da sorunlara sebep olur. Collection frameworkü bize bir standart sağlar.

Java collection frameworkü bir grup nesneyi list,set,queue veya map olarak saklayabilir, her birinin detaylarını ayrı ayrı inceleyeceğiz.

Her birinin farklı önemleri ve özellikleri bulunsa da aynı interfaceleri implemente ettikleri için çoğu önemli metotları ortaktır.

Bu yapıları kullanarak dinamik olarak artan azalan elemanları saklayabilir, kullanabiliriz. Bunu yaparken de eleman sayısını düşünmemize gerek kalmaz.

Bunları ezberlemenize gerek yok ama collection frameworkü 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.

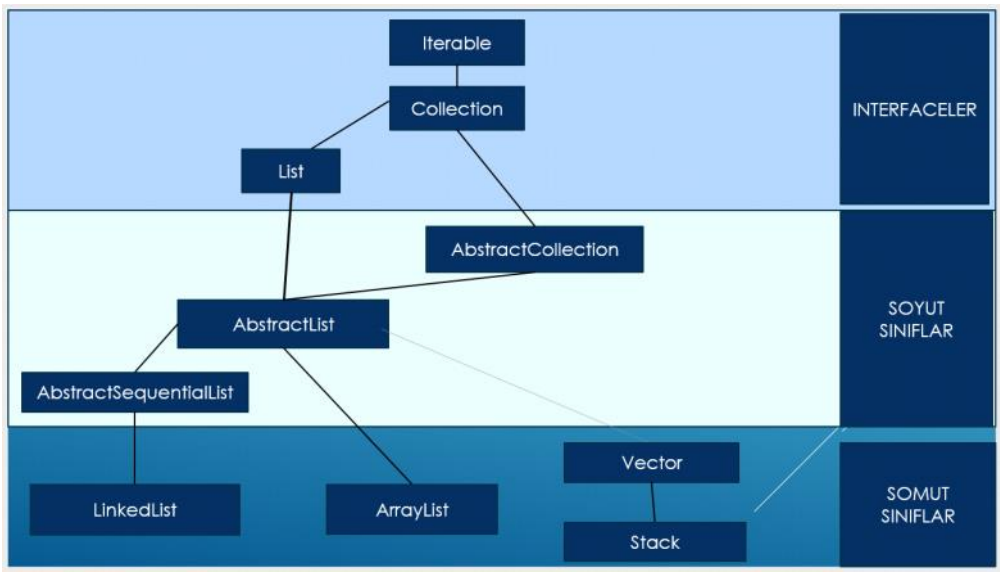
Tekrar söyleyelim List, Queue ve Set Collection interfacesinden extends edilen interfacelerken Map interfacesi Collection interfacesinden extends edilmemiştir.

Bunlar soyut yapılarken bu interfaceleri gerçekleştiren sınıflara ise somut sınıflar deriz ve bu yapıları kullanırken somut sınıflardan faydalanırız.

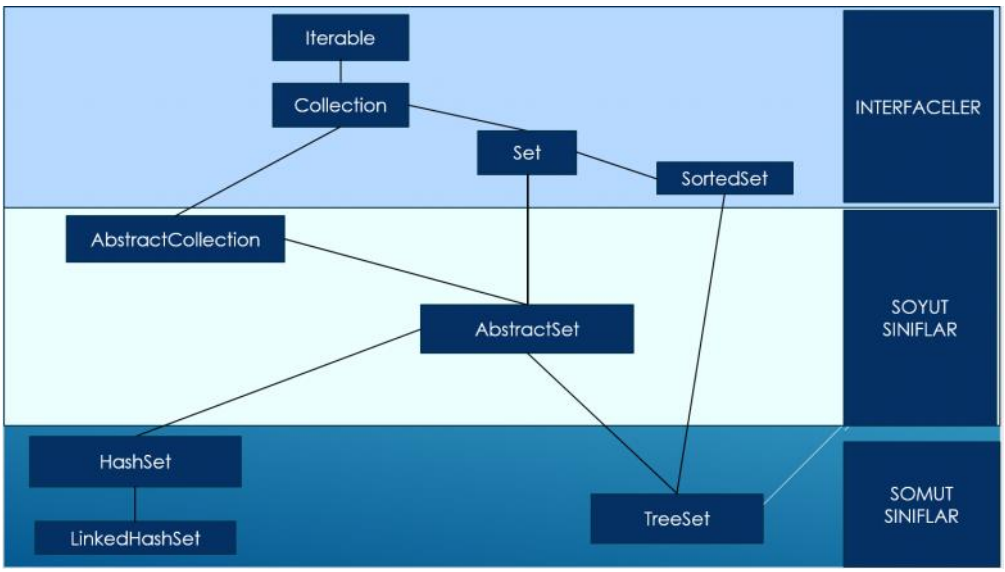
Liste Map ve Setlerin Sınıf Hiyerarşileri

Emre Altunbilek Java Dersleri

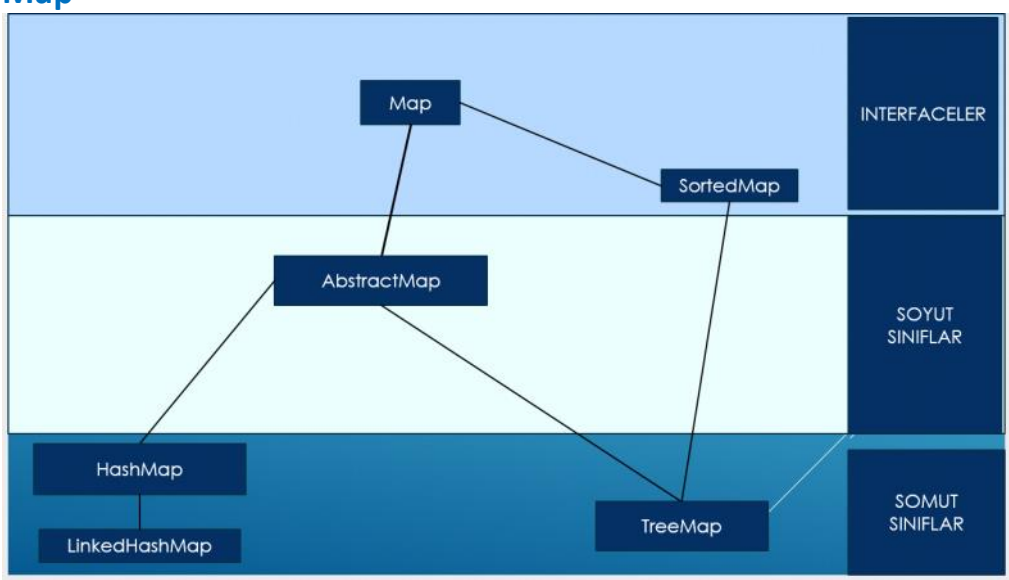
Liste



Set



Map



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.

No	Metot Adı	Tanımı
1	int size()	Eleman sayısını döner.
2	boolean isEmpty()	Koleksiyon yapısı boş ise true, değilse false döner
3	boolean contains(Object o)	Aranılan eleman varsa true döner
4	Iterator<E> iterator()	Koleksiyon için iterator nesnesi döner
5	Object[] toArray	Koleksiyonu diziye dönüştürür.
6	<T> T[] toArray(T[] a)	Koleksiyondaki sadece belirtilen tipteki elemanları diziye dönüştürür.
7	boolean add(E e)	Verilen elemanı koleksiyona ekler, başarılıysa true döner
8	boolean remove(Object o)	Verilen elemanı koleksiyondan çıkarır.
9	boolean containsAll(Collection<?> c)	Koleksiyon parametre olarak gönderilen koleksiyondaki elemanları içeriyor mu
10	boolean addAll(Collection<? extends E> c)	Parametre olarak gelen koleksiyondaki tüm elemanları ekler.
11	boolean removeAll(Collection<?> c)	Parametre olarak gelen koleksiyonda bulunan tüm elemanlar varsa koleksiyondan kaldırılır.
12	boolean retainAll(Collection<?> c)	İki koleksiyonda da ortak olan elemanları saklar, diğer elemanları siler
13	void clear()	Koleksiyondaki tüm elemanları siler
14	boolean equals(Object o)	Koleksiyonla gelen parametredeki değeri karşılaştırır.
15	int hashCode()	Koleksiyonun hashcodeunu döner.

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

Bir sınıf interfaceyi implemente ettiğinde kendine özgü olarak bu metotları yazmalıdır. Veya object sınıfından kalıtıldı için yazmasa da olur ama yanlış çalışmalar olabilir.

Collection bir interface iken Collections koleksiyonlarla ilgili güzel metotları bulunan bir sınıftır.
Array ve Arrays gibi -> Array bir veri yapısı iken Arrays dizilerle ilgili metotlar barındıran bir sınıftır.

List Interface

Genel Özellikleri

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.

No	Metot	Açıklama
1	E get(int index)	verilen indeksteki elemanı döndürür.
2	E set(int index, E element)	verilen indeksteki elemana parametredeki elemanı atar.
3	void add(int index, E element)	verilen indekse parametredeki elemanı ekler.
4	E remove(int index)	verilen indeksteki elemanı siler
5	int indexOf(Object o)	parametredeki nesnenin varsa koleksiyondaki ilk görüldüğü yerin indeksini döndürür.
6	int lastIndexOf(Object o)	parametredeki nesnenin varsa koleksiyondaki son görüldüğü yerin indeksini döndürür.
7	ListIterator<E> listIterator()	listeyi gezebileceğiniz bir listiterator döndürür.
8	ListIterator<E> listIterator(int index)	verilen indeksten itibaren elemanları gezebileceğiniz bir iterator döndürür.
9	List<E> subList(int fromIndex, int toIndex)	parametrede verilen indekslerin arasındaki elemanları içeren bir liste döndürür.

Arraylistin Array'e Göre Artıları

Neden Arraylist Kullanmalıyız ?

- 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.

Bunun için video derse bakabilirsiniz.

Bir listeyi baştan sona sondan basa kolayca gezebiliriz.

```
ArrayList<String> liste = new ArrayList<String>();

liste.add("EMRE");

liste.add("HASAN");

liste.add("ALI");

liste.add("AYSE");

ListIterator iterator = list.listIterator();

System.out.println("Baştan sona");

while (iterator.hasNext())
{
    System.out.println(iterator.next());
}

System.out.println("Sondan başa");

while (iterator.hasPrevious())
{
    System.out.println(iterator.previous());
}
```

Listemizi Gezmenin Bazı Yöntemleri

Normal for döngüsü ile

```
for (int i = 0; i < liste.size(); i++)
{
        elemanın_veri_tipi element = list.get(i);
}
```

Iterator ile

```
Iterator<String> it = list.iterator();
while (iterator.hasNext())
{
    System.out.println(iterator.next());
    iterator.remove();
}
```

ListIterator ile

```
ListIterator<String> listIt = list.listIterator();
while (listIterator.hasNext() or listIterator.hasPrevious())
{
    System.out.println(listIterator.next());

        System.out.println(listIterator.previous());
}
```

Gelişmiş for döngüsü ile

```
for (eleman_veri_tipi element : liste)
{
    System.out.println(element);
}
```

Iterator ile ListIterator Farkları

List iterator interfacesi Iterator interfacesinden türetilmiştir. Iterator ile listeler,setler ve queueler gezilebilir(sadece bastan sona dogru)

Oysaki listiterator ile baştan sonra sonra basa gezilebilir. O yüzden eğer amacımız liste yapılarını gezmekse bu daha mantıklıdır.

Iterator metotları
hasNext, next ve remove

ListIterator metotları
hasNext, hasPrevious, next , previous, nextIndex, previousIndex, remove,set,add ..

Görüldüğü gibi listeler için çok daha fazla metot barındırıyor. Ayrıca listeyi gezerken bir önceki ve bir sonraki elemanın index değerlerini de elde edebileceğimiz metotlar var. O yüzden listiterator kullanmak daha mantıklı.

Ayrıca listiterator ile ekleme yerdeğiştirme ve silme yapabilirken iterator ile sadece silme yapılabilir.

Listiterator ile belli bir indexten başlayıp listemizi gezebilirken iterator ile bunu yapamayız.

Arraylist ile LinkedList Farkları

Bazı önemli farklar

Her ne kadar iki sınıfta List interfacesini implemente etse de bazı ortak yanları ve farkları vardır. Çünkü çalışma şekilleri farklıdır. Aşağıdaki tablo bu benzerlik ve farklılıkları göstermektedir.

	ArrayList	LinkedList
Yapısı	Elemanlar indeks bazlı tutulur. Her elemanın bir indeks değeri vardır.	Burdaki elemanlara Node denir. Bir node 3 bilgi taşır. Önceki elemanın referansı, veri ve sonraki elemanın referansı
Ekleme ve Çıkarma	Eğer listenin ortasına eleman ekleme-cıkarma yapılırsa oldukça yavaştır. Çünkü bu işlemlerden sonra kaydırma yapılır.	Ekleme ve çıkarma nereye yapılacağına bakılmaksızın hızlıdır. Çünkü indeks bazlı değil, kaydırma yapılmaz.
Eleman bulma Eleman çağırma	Daha hızlıdır, çünkü elemanlar indeks bazlı tutulur.	Bir elemanın bulunması için tüm listenin gezilmesi gerekir. Indeks bazlı olmadığı için bu şekilde yapılmalı
Random Access	Rastgele olarak istenilen elemana ulaşılabilir.	Elemanlara rastgele ulasılamaz. Bir elemana ulaşmak için öncesindeki elemanları gezmek gerekir.
Kullanım	Arraylist Stack veya Quenue olarak kullanılamaz.	LinkedList Arraylist, Stack,Quenue, Tek Bağlı veya Çift Bağlı Liste olarak kullanılabilir.
Bellek Kullanımı	Arraylistte sadece eleman tutuldugundan daha az yer kaplar.	Burada data ile beraber önceki ve sonraki elemanın referans değerleri tutulduğu için daha fazla yer kaplar.
Ne zaman Kullanalım	Ekleme cıkarmadansa eleman saklama ve getirme daha çoksa arraylist daha mantıklı	Ekleme cıkarma çok fazla yapılıyorsa linkedlist daha mantıklı

- İki yapı da list interfacesini implemente eder,
- Ikisi de Cloneable ve Serializable interfacesini implemente eder,
- Ikisinde de elemanlar eklenme sırasına göre tutulur ve ikisi de non-synchronized'dır. Yani aynı anda birden fazla thread buradaki elemanlara müdahele edebilir.

Vector Sınıfı

Tanım

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.

Vector<Integer> vector = new Vector<Integer>(3); vectorumüzü 3 elemanlı olarak oluşturduk.

vector.capacity() ile vektörümüzün o anki boyutunu görebiliriz. Bu kapasite aşıldığı anda vector kapasitesini 2 katına çıkarır.

vector.setSize(20) diyerek vektörün boyutunu arttırabiliriz, bu durumda vektör boş olan elemanlara null değerleri atar.

Vektördeki elemanları gezmek için

```
Enumeration<Integer> en = vector.elements();
while (en.hasMoreElements())
{
    System.out.println(en.nextElement());
}
```

vector.firstElement() ve vector.lastElement() ile ilk ve son elemanlara ulaşabilirsiniz.

Bunların dışında vektör kullanımı çok önerilmez, peki neden ?

- Thread safe özelliğini arraylistlere de kazandırabiliriz.
  - Collections.synchronizedList(arraylist)
- Vektörlerdeki thread safe özelliği çok fazla zaman alır.

  Eski bir sınıftır, eksi özellikleri çoktur o yüzden arraylist kullanalım.

Stack Yığın Sınıfı

Tanım

Vektör sınıfından türetilmiştir, ilk giren son çıkar mantığı ile çalışır.

empty() -> stack boş ise true döner
peek() -> En son elemanı döner;
pop() -> En son elemanı döner ve yığından cıkarır.
push(eleman) -> yığının en sonuna parametredeki elemanı ekler
search(eleman) -> parametredeki elemanın yığındaki yerini döner.

Stack sınıfı yerine Linkedlist sınıfı kullanılabilir.

```
        LinkedList<Integer> stack = new LinkedList<Integer>();


        stack.push(10);

        stack.push(20);

        stack.push(30);

        stack.push(40);



        System.out.println(stack);      //Output : [40, 30, 20, 10]

        //Poping out the elements from the stack

        System.out.println(stack.pop());    //Output : 40

        System.out.println(stack.pop());    //Output : 30
```

Queue Interface ve Sınıfları

Queue Interface

Collection interfacesinden türetilmiştir. Ilk giren ilk çıkar mantığı ile çalışır. Priority queue ise biraz farklıdır. Burada elemanlar önceliğe göre sıralanır, en küçük değeri olan kuyruğun başına geçer.

Kuyruğun başı ve sonu vardır. Elemanlar sona eklenir, baş taraftan çıkartılır. Kuyruğun ortasına bir eleman eklenmez.

Burada null değerlere izin verilmez. Aynı elemanlar eklenebilir. Liste gibi random access erişim yoktur.

poll metotu ile elemanı kuyruktan çıkarırız, eleman yoksa null değer döner. remove metotu da kullanılabilir, yalnız eleman yoksa hata alırız.

peek ile en baştaki elemanı görebiliriz eğer eleman yoksa null değer döner. element metotu da kullanılabilir, yalnız eleman yoksa hata alırız.

offer ile kuyruğun sonuna eleman eklenir eğer başarılı olunmazsa null değer döner, add ile de ekleyebilirz yalnız başarısız durumda hata alırız.

PriorityQueue Sınıfı

Elemanları belli bir sıraya göre saklayan somut sınıftır.Wrapper ve string sınıflar için herhangi bir şey yapmaya gerek yoktur ama kendi sınıflarımızı kullanırken bu sınıfların Comparable interfacesini gerçekleştirmesi gerekir ki java sıralamayı nasıl yapacağını bilebilsin.

Eğer string ve wrapperlar için normal sıralama istemiyorsanız bu sınıftan nesne üretirken comparator interfacesini gerçekleştirerek isteğinizi belirtebilirsiniz.

Set Interface

Set Interface

Set yapısının en önemli özelliği tekrar eden elemanları barındırmamasıdır. Bu interface de Collection interfacesinden türetilmiştir. Yani collection interfacesindeki tüm metotlar burada da bulunur.

Tek fark add() metotundadır. Burdaki add metotu eğer zaten ekli bir eleman varsa false değer döndürür.

Tekrar eden eleman bulunmaz, sadece tek bir null değere izin verilir.

Rastgele erişim mümkün değildir, yani index ile erişim mümkün değildir. Bir elemana erişilecekse sırayla gezmek gerekir.

Interfaceden bir nesne üretilemez, o yüzden set mantığında bir yapı istiyorsak bu interfaceyi kullanan somut bir sınıftan nesne üretmek lazım. Eğer Hashset sınıfı kullanılırsa elemanlar hash code'a göre sıralanır, eğer TreeSet sınıfı kullanılırsa elemanlar Comparator da belirtilen kurallara göre sıralı saklanır Eğer LinkedHashSet kullanılırsa elemanlar eklenme sırasına göre saklanır.

Set interfacesinin kendine özgü metotları yoktur, collections interfacesindeki metotları tekrar eden eleman olmayacak sekilde kullanır.

Sınıflarımızı oluştururken equals ve hashcode metotlarını override edip, eşit olarak kabul ettiğimiz iki nesnenin aynı hashcode'u üretmesini sağlamalıyız.

HashSet Sınıfı

Hashset sınıfı set interfacesini gerçekleştiren somut sınıflardan biridir. İç yapısında tuttuğu HashMap ile elemanlar saklanır.

Saklamak istediğimiz veriler hashmap de anahtar olarak saklanır. Karşısına da değer olarak boş nesneler konur.

Var olan eleman tekrar yazılırsa eski elemanın üzerine yazılır.

Hashset kullanımında herhangi bir sıralama garantisi yoktur. Yani elemanları baştan sona yazdırırken farklı sonuclarla karşılaşabilirsiniz. Hashsetteki eleman ekleme silme ve sorgulama işlemlerinin süreleri aynıdır.

Hashset sınıfı senkronized değildir.

Bir hashset olusturdugumuzda çalısan kurucular:

```
private transient HashMap<E,Object> map;


public HashSet()
{
        map = new HashMap<>();
}


public HashSet(Collection<? extends E> c)
{
        map = new HashMap<>(Math.max((int) (c.size()/.75f) + 1, 16));
        addAll(c);
}


public HashSet(int initialCapacity, float loadFactor)
{
        map = new HashMap<>(initialCapacity, loadFactor);
}

//Constructor - 4

public HashSet(int initialCapacity)
{
        map = new HashMap<>(initialCapacity);
}
```

LinkedHashSet Sınıfı

Tanımı

Bu set yapısında ise elemanlar çift bağlı liste ile iç tarafta saklanır. Burda elemanlar eklenme sırasına göre tutulur. Tabiki de burda da elemanların hepsi birbirinden farklıdır.

LinkedHashSet extends HashSet implements Set….

Aslında bir linkedhashset nesnesi oluşturunca arka planda elemanlar bir linkedhashmap yapısı oluşturulur. Hashsetten en büyük farkı elemanların eklenme sırasına göre saklanmasıdır.

Ekleme çıkarma ve elaman okuma işlemleri aynı hızlıdır, ama tabiki hashsete göre biraz daha yavastır çünkü elemanların eklenme sırasına göre saklanması için çift bağlı liste oldugu için biraz daha yavastır.

Not: Bu set yapısını gezmek için iterator kullanabiliriz. Bu iterator olusturulduktan sonra eğer set yapısı değiştirilirse eleman eklenip çıkarılırsa hata alırız.

```
Iterator<String> it = set.iterator();

set.add("JSF");//set yapısı değişti

while (it.hasNext())
{
    //Fırlatılan hata: ConcurrentModificationException

    System.out.println(it.next());
}
```

TreeSet Sınıfı

Tanım

Diğer somut set sınıfları olan HashSet ve LinkedHashSetten farkı burada elemanların sıralanarak saklanmasıdır. Sakladıgımız ve sıralayarak saklamak istediğimiz sınıflarda Comparable interfacesini uygulayarak bu işlemi yapabiliriz.

Veya ilk treeset oluşturuldugunda comparator geçerek bu sıralamanın nasıl olacağını belirleyebiliriz.

Yalnız burada null değer saklayamıyoruz.

Treeset iç yapısında treemap kullanır.

Treeset ekleme silme ve eleman bulma değeri log(n) dir, n -> eleman sayısı

Set Sınıflarının Farklılıkları

Alt Başlık

Üç sınıf da set interfacesini gerçekleştirse de aralarında bazı farklar vardır. Aşağıdaki tablo ile konunun özetini yapalım

	HashSet	LinkedHashSet	TreeSet
İç yapısında ne kullanır?	HashMap	LinkedHashMap	TreeMap
Elemanları Nasıl Saklar	Herhangi bir sıralama yapmaz	Eklenme sırasına göre saklar	Comparator ile belirlenen kurala göre elemanları sıralayarak saklar.
Performans (işlemci performansı)	Diğerlerine göre daha hızlıdır. Çünkü bir sıralama yapılmaz	Hashsete göre daha yavaş, treesete göre daha hızlıdır. Elemanların eklenme sırasına göre saklanması için linkedlist kullanır.	En yavaş çalışan set sınıfıdır.
Ekleme, Çıkarma, Eleman bulma Operasyonları	O(1) yani hepsi aynı performanslıdır ve eleman sayısı önemli değildir.	O(1) yani hepsi aynı performanslıdır ve eleman sayısı önemli değildir.	O(log(n))
Elemanlarını nasıl karşılaştırırlar?	Hashcode ve equals metotlarını kullanırlar.	Hashcode ve equals metotlarını kullanırlar.	Compare veya compareto metotlarını kullanarak elemanları sıralar.
Null Eleman	1 tane null eklenebilir	1 tane null eklenebilir.	Boş bir treesete null eleman eklenebilir. Eleman olan bir treesete null değer eklenemez.
Bellek Kullanımı	Sadece hashmap kullandığı için en az bellek gerektiren sınıftır	Hashmap + Linkedlist kullandığı için biraz daha fazla bellek gerektirir.	Sıralama yaptığı için biraz daha fazla bellek gerektirir.
Ne zaman kullanırım?	Sıralama gerekli değil ama tekil elemanlar saklamak istiyorsak	Eklenme sırası önemli ise ve tekil eleman saklamak istiyorsak	Tekil eleman ve sıralama önemliyse

Map Interface

Tanım

List, Set ve Queue interfaceleri gibi Collections interfacesinden türememiştir. Üst interfacesi yoktur, elemanlarını key-value; anahtar-değer ilişkisinde tutar.

Haspmap, LinkedHashMap ve Treemap gibi somut sınıflar bu interfaceden türemiştir.

Map yapısında keyler tekildir yani tekrarlanmaz, aldıkları değerler tekrar edebilir.

Her anahtar-değer ikilisine bir entry denir. Map.Entry aslında map interfacesinin içinde tanımlı olan bir interfacedir.

Hashmap de elemanların sırası önemli değildir, linkedhashmap de elemanlar eklendikleri sırada tutulur ve son olarak treemap de ise elemanlar sıralı olarak saklanır.

keySet metotu ile anahtarları, values metotu ile değerleri ve entrySet ile de anahtar-deger ikililerini elde edebiliriz.