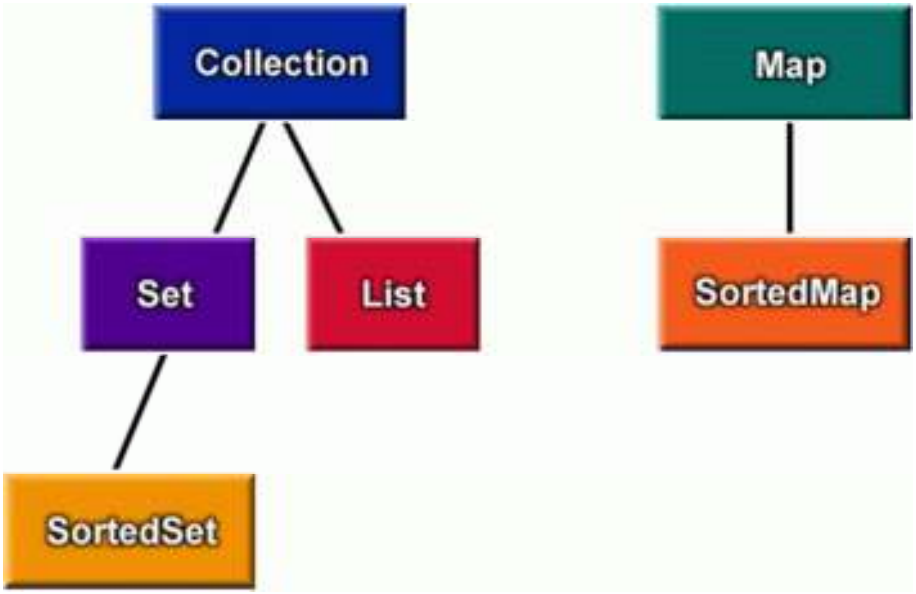
Java Koleksiyonları

(Java Collections)



Giriş

Bu bölümde, java standart kütüphanesinde yer alan `Collections` topluluğunu ele alacağız.

Amaçlar

Bu dersin sonunda öğrenci şunları biliyor olacaktır:

- Java Collctions Framework adıyla bilinen topluluğu tanıyacak,
- Collections arayüzlerinde yer alan öğeleri tanıyacak,
- iterators (tekrarlayıcılar, döngüler) ile ilgili kavramları öğrenecektir.

Bu bölüm şu altbölümlere ayrılmıştır:

1) Collections topluluğu nedir?

2) Collection arayüzleri

3) Eski ve yeni Collections

4) Lists

5) Sets

6) Maps

7) Collection kılgılama (implementations)

Collections Nedir?

Çoğu yazılım tek tek öğeler yerine öğelerden oluşan toplulukları depolar ve onlar üzerinde işlem yapar. *Array*'ler onlardan birisidir. *Java Collections Framework*, arraylerle yapılan işleri daha kolay yaptığı gibi, daha fazlasını da yapar.

Java'da bir koleksiyon (collection - bazen container, ambar diye adlandırılır) nesnelerden oluşan bir topluluğu bir arada tutan bir yapıdır. *'Collections Framework'* ise arayüzler ve onların kurgularından (implementations) oluşur.

- Olabilir işlevleri arayüzler tanımlar
- Kılgı (implementation) ise, onarı hayata geçirir.

Java Collection Framework denilen çatı altında biraraya getirilen arayüz (interface) ve sınıfların (class) kullanılışlarını ele almadan önce bu çatının avantajlarını ve dezavantajlarını bilmek yararlı olabilir. Adından da anlaşılacağı gibi bir koleksiyon (collection) içinde bir çok öğeyi barındıran bir nesnedir (object). Öğeler, veri gruplarından oluşur; sıralı ya da sırasız olabilirler. Bazı bazı koleksiyonlarda aynı öğe birden çok kez (dublikasyon) koleksiyonda yer alabilir, bazılarında yer alamaz. Veri koleksiyonu yapmaktaki amacımız verilere erişim sağlamak, veriler üzerinde işlem yapmak, verileri sıralamak, yeni veri eklemek ya da mevcut bir veriyi silmek, bir verinin koleksiyon içinde olup olmadığını aramak gibi eylemleri gerçekleştirebilmektir.

Java Collection Framework'un Avantajları:

1. Veri kollleksiyonları üzerinde yukarıda sıralanan eylemeleri yapmaya yarayan ve API[1] adını alan arayüzleri öğrenmek zorunda kalmadan, *Java Collection Framework* yardımıyla istediğimiz eylemleri gerçekleştirebiliriz.
2. Yazılımın tekrar kullanılmasını (software reuse) sağlayan arayüzleri ve veriler üzerinde işlem yapmaya yarayan algoritmaları içerir.
3. Özel amaçlar için bile olsa, programcı yeniden algoritma yazmak zorunda kalmaz.
4. İçerdiği veri yapıları onlara uygulanan algoritmalar, programcının işini kolaylaştırır, üretim zamanını kısaltır, programın güvenirliğini sağlar.
5. İçerdiği veri yapıları onlara uygulanan algoritmalar, programın performansını artırır.
6. API!lerin kullanılışı konusunda ortak bir dil oluşturur.
7. Veri ekleyip çıkardıkça, koleksiyonların uzunluğu (büyüklüğü) kendi kendine değişir; programcının o değişimi ayarlaması gerekmez.

Java Collection Framework'un Dezavantajları:

1. Derleme anında veri tipi denetimi yapamaz.
2. Veri tipini doğru seçmek gerekir.

[1] **Application Programming Interface** (API), java tarafından hazırlanan kütüphanelere verilen addır. Her birisi farklı amaçlar için kullanılabilecek çok sayıda API vardır ve onlar bir programcının gerekseme duyacağı hemen hemen her şeyi içerirler.

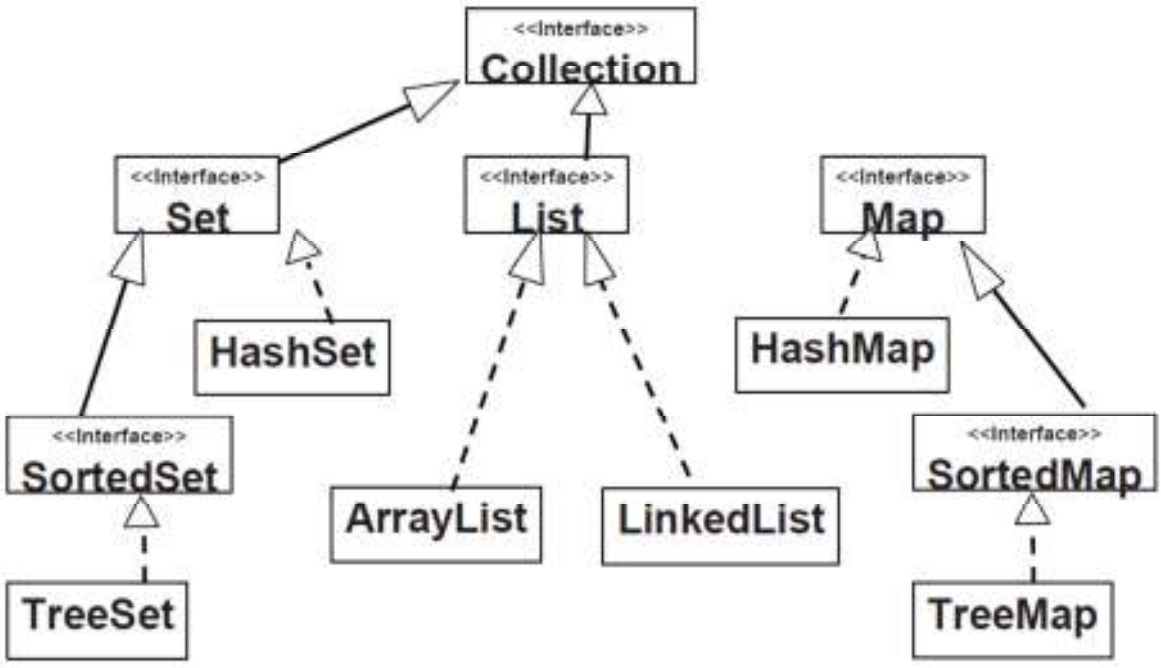
Collection Arayüzleri

Collections framework'un özünü oluşturan şey '`Collection`' denen arayüzdür. Bu arayüz framework'un temeli olan metotları tanımlar. `List` ve `Set` arayüzleri `Collection` arayüzünde olmayan metotları tanımlayarak, *framework*'un uygulama alanını genişletirler.

Arayüzler yalnızca metotların imzalarını taşırlar; sınıflarda olduğu gibi metotların gövdelerini (işlevi gerçekleştiren kodları) içermezler.

Önemli bir başka arayüz `Map` adını alır. Ancak Map arayüzü Collection arayüzünün bir genişlemesi değildir. İleride göreceğimiz nedenlerle, `Map` arayüzü `Collection` hiyerarşisine dahil değildir; ama *Collections framework*'un bir parçasıdır.

Söylediklerimiz özetleyerek, arayüzler için şu sınıflandırmayı yazabiliriz:



Java Collection Framework

• **Collection:** en genel grup

– **List:** nesnelerden oluşan topluluk. Topluluk içinde dublikasyon olabilir, topluluğun belirli bir sıralaması vardır.

– **Set**: Sırası olmayan ve duplikasyonu olmayan nesneler topluluğu.

– **SortedSet**: Artan sırada sıraya dizilmiş nesneler kümesi.

• **Map**: her öğesine bir anahtar atanmış nesneler topluluğu

– **SortedMap**: Anahtarlarına göre artan sırada dizilmiş nesneler topluluğu.

Şekilden görüldüğü gibi, `List` ve `Set` arayüzleri `Collection` arayüzünü genişletir. *SortedSet* arayüzü *Set* arayüzünü genişletir. *SortedMap* arayüzü *Map* arayüzünü genişletir.

Bunları ilerideki alt bölümlerde ayrıntılı olarak inceleyeceğiz.

Eski ve Yeni 'Collections'

Java SDK 1.4 sürümüne kadar, koleksiyonları *Object* sınıfının altsınıfları olarak ele aldı. Böyle olması doğaldı, çünkü Java'nın bütün nesneleri `Object` sınıfından elde edilir. Bu demektir ki, bütün veri tipleri `Object` sınıfının altsınıflarıdır. Bu yaklaşım Java'da halen geçerlidir. Ancak, nesneler, bir nesneler topluluğundan (koleksiyon) alındığında derleyici onların hangi sınıfa ait olduğunu ayıramayabiliyor ve bu durum bazen sorun yaratıyor.

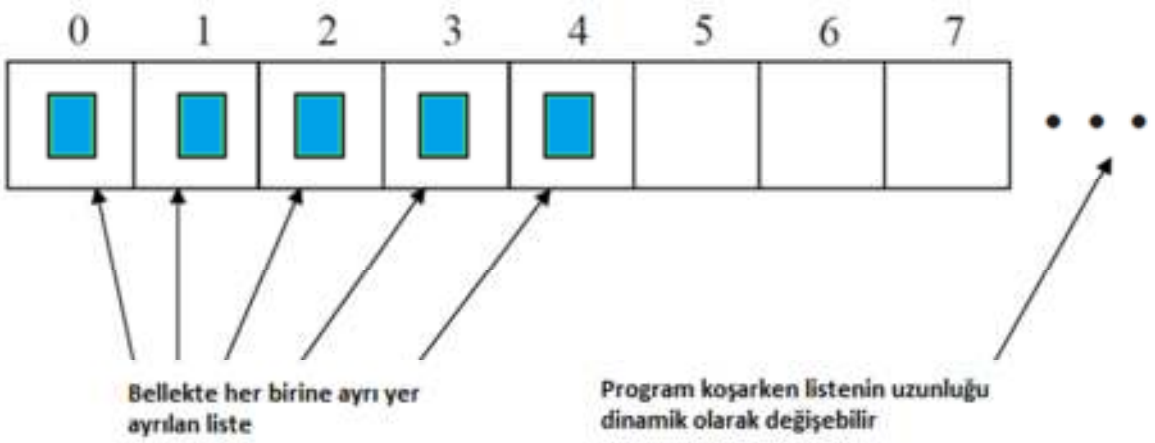
Pratikte, bir koleksiyondaki belirli özeliği olan nesnelerle işlem yaparız. Bu sorunu çözmek için Java JDK 5.0 sürümünde '`Generics`' kavramını getirdi. '`Generics`', içerdiği veri tiplerini belirlemeksizin sınıfın tanımlanmasına izin verdi. Sınıfa ait nesne yaratılınca (instantiate), sınıfın içerdiği veriler belirlenmiş oluyor.

Böylece, '`Generics`' yardımıyla, yalnızca belirlenen tipten nesneleri içeren koleksiyonlar oluşturmak mümkün olmaktadır.

Listeler (Lists)

`Listeler` koleksiyonların yaygın olarak kullanılan türüdür. `Array` tipinin kullanıldığı her yerde kullanılırlar. Ama veri işleme eyleminde, `array`'in sağladığından daha çok şeye izin verirler. Listelerin her öğesi (terim) bellekte kendine özgü bir yer tutan veri yapılarıdır ve çok genel işlerin yapılmasına olanak sağlarlar. Listeler, bir çok bakımdan arraylere benzemekle birlikte, yeni öğe eklendikçe uzunlukları kendiliğinden artar; dolayısıyla arraylere göre daha kullanışlıdır. Bunun yanında, veri işlemeye yarayan çok sayıda metot içerdikleri için, programcıya, arraylerin sağladığından daha büyük kolaylıklar sağlar.

Listeler, öğelerini bir dizi halinde depolar. Dizinin her hangi bir sırada olması gerekmez; ama istendiğinde kolayca sıralanabilirler. Ayrıca, listede, aynı öğeler birden çok kez yer alabilir (duplicate).



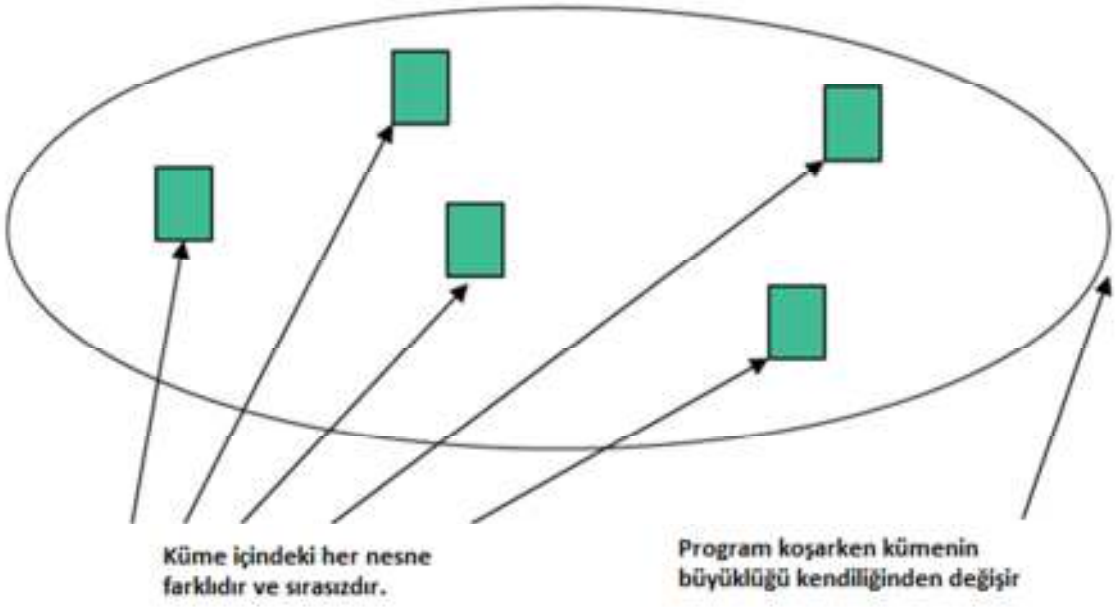
Kümeler (Sets)

Küme (set) içindeki öğeler, bir dizi biçiminde değil, bir torbaya doldurulmuş biçimdedirler. Matematikteki küme kavramından gelir. Hiç öğesi olmayan küme *boş* kümedir. Kümenin bir, iki, ya da çok sayıda öğesi olabilir. Ancak öğe sayısı *sonlu* olmalıdır. Matematikte *sonsuz* öğeli kümeleri tanımlayabilir ve onlarla işlem yapabiliriz. Ancak, bilgisayarlarda öğeler ve işlemler sonlu sayıda olmak zorundadır. Bilgisayarlarda sonsuz sayıda öğe tanımlanamaz, sonsuz sayıda işlem yapılamaz. O nedenle, listeler için olduğu gibi, kümeler içinde öğe sayısının sonlu olma koşulu vardır.

Matematikte bir kümede aynı öğe birden çok kez yer alamaz. *Collections* içindeki *Set* topluluğu bu kurala uyar. Set içinde aynı öğe ancak bir kez yer alabilir (duplikasyon olamaz).

Matematikte bir kümenin öğeleri sıralı olmak zorunda değildir. *Collections* içindeki *Set* topluluğu bu kurala uyar. Eğer, öğelerin sıralanması gerekiyorsa, *SortedSets* altkoleksiyonu kullanılır.

Listelerde olduğu gibi, kümeye yeni öğeler eklendikçe, küme otomatik olarak büyür; programcının ayrı kod yazmasına gerek kalmaz.



Küme (set)

List için var olan işlemlerin (operations) çoğu kümeler için de geçerlidir. Ancak şu kısıtlar vardır:

- Kümenin öğeleri sıralı olmadığı için, yeni gelen bir öğeyi kümede belirli bir konuma yerleştiremeyiz.
- Aynı nedenle, bir öğe yerine başka bir öğe koyamayız (replacement olamaz). Ama, istenen öğe kümeden silinebilir ve istenen öğe kümeye eklenebilir.
- Kümedeki öğelere erişmek (retrieving) mümkündür, ama erişim sırası belirsizdir.
- Kümede bir öğenin yeri belirsizdir.

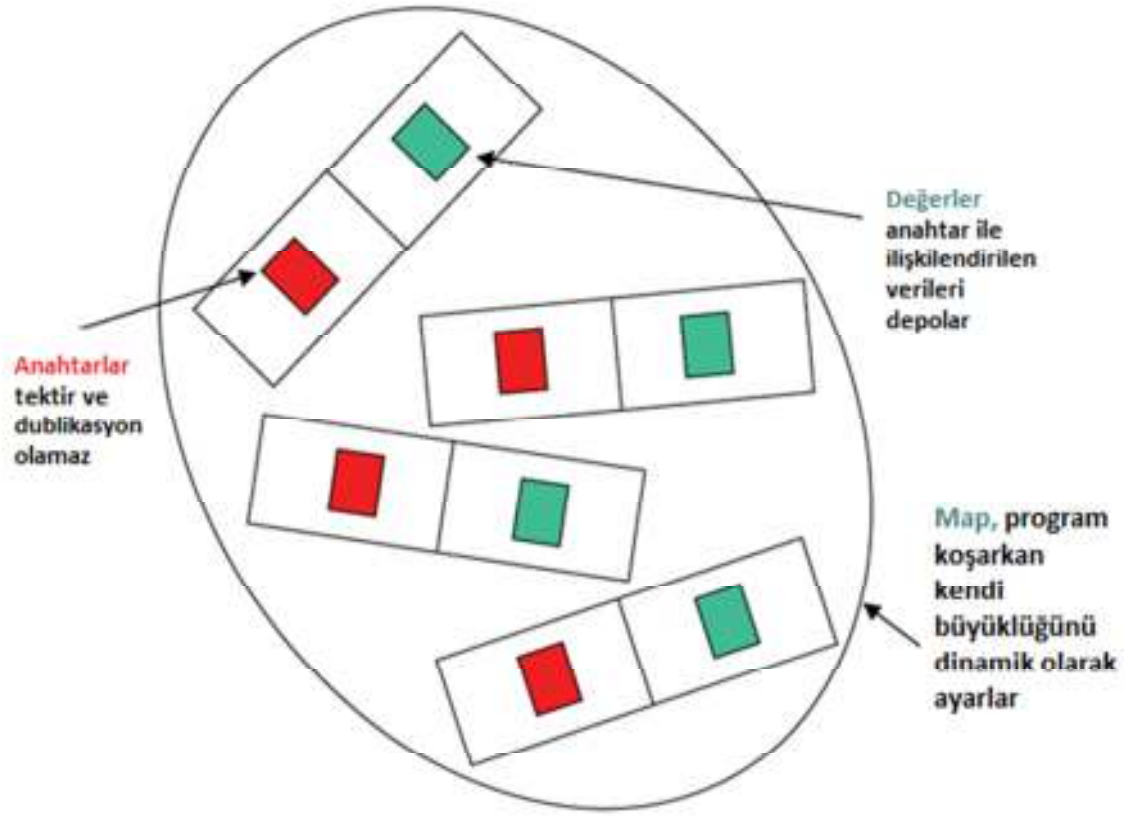
Maps (Dönüşümler)

Dönüşüm (Map)'ler, yapısal olarak, *Listeler* ve *Kümeler*'den çok farklıdır. Öğeleri tek tek depolamak yerine nesne çiftlerini depo ederler. Depolanmak istenen her öğeye bir *anahtar* verilir; böylece bir öğe yerine bir öğe çifti oluşur ve bu çiftler depo edilirler. Depodaki her öğeye kendi anahtarıyla erişilir. Öğe çifti "*anahtar*" ve "*değer*" olmak üzere iki nesneden oluşur. Anahtar, oluşan çifti belirleyen işaretçidir; değer ise anahtara ilişkilendirilen bilgiyi içeren bir nesnedir.

Örnek: Bir adres defteri düşünelim. Defter soyad sırasıyla düzenlenebilir. O zaman *soyad* anahtar olur. Her *soyad* ile ilişkilendirilen *ad, telefon_numarası, adres, doğum_günü* gibi bilgileri içeren bir nesne "*değer*" nesnesidir. Her anahtara karşılık böyle bir ve yalnız bir tane "değer" nesnesi vardır; ama değer nesnesi birden çok bilgi (veri) içerebilir.

Map içinde anahtarlar tektir; yani aynı anahtar birden çok değeri işaret edemez. Ancak, farklı iki anahtarın işaret ettiği değerlerde aynı veriler olabilir. Örneğin, adres defterinde farklı iki kişi aynı telefonu kullanabilir ya da aynı adreste kalabilirler. Tabii, telefon defterinde *soyad* anahtar alınırsa, aynı soyadı taşıyan iki kişi duplikasyon yaratır. Bu durumda *soyad* anahtar olamaz. Böyle durumlarda başka bir anahtar düşünmek gerekir. Örneğin, defterde kayıtlı herkese bir sıra numarası verilebilir.

Kümeler için olduğu gibi, bir *Map* öğeleri sıralamaz. Gerekiyorsa, anahtara göre sıralama yapan *SortedMap* arayüzü kullanılır. Yeni öğeler geldikçe, *Map* kendi kendisinin boyunu (öğe sayısını) artırabilir. Tersine olarak, öğe silinirse, *Map* kendi boyunu küçültür.



Map (dönüşüm)

Aktivite 1

Aşağıdaki toplulukların her birisi için en uygun yapının (list, set, map) ne olduğunu yazınız:

1) Bir dernek üyelerine ait bilgileri kaydetmek istiyoruz.
2) Harcamalarımızı kaydetmek istiyoruz.
3) Banka hesabımıza ait bilgileri (para hareketleri – yatan, çekilen- kaydetmek istiyoruz.

Geribildirim 1

1) Bunun için Set uygun bir yapıdır. Derneğe üye olanlar kaydedilir, üyeliği düşenler silinir; üyeler için bir sıralama yoktur.
2) Bunun için List yapısı uygundur. Alışverişlerimizi tarih sırasıyla tutabiliriz. Aynı cins malı birden çok kez alabiliriz.
3) Aynı numara ile iki banka hesabı olamayacağı için Set yapısı uygun olabilir. Daha iyisi, hesap numaralarını anahtar kabul eden Map yapısı kullanılabilir.

Koleksiyon Kılgılama

(Collection Implementations)

Şimdiye kadar koleksiyonu oluşturan arayüzlerden söz ettik. Ama onların kullanılabilmeleri için, ilgili arayüzün bir sınıf tarafından kılgılanması (implement) gerekir. Bu altbölümde bunun nasıl yapıldığını inceleyeceğiz. Başka bir deyişle, sözkonusu arayüzlerin kılgısını yapan sınıflara gerekseme vardır.

Aşağıdaki sınıflar sözü edilen arayüzlerin kılgısını yapan sınıflardır. Bu sınıflar *"Java platform library packages"* paketindedirler. Ancak, programcı, isterse arayüzlerin kılgısını yapan sınıflar tanımlayabilir.

• **ArrayList** sınıfı `List` arayüzünü kurgular

– sıralı listelerde hızlı erişim sağlanır

• **LinkedList** sınıfı da `List` arayüzünü kurgular

– bazı özel durumlarda ArrayList 'dekinden daha hızlı erişim sağlanır

• **HashSet** sınıfı `Set` arayüzünü kurgular

– hızlı erişim sağlar, ama öğeleri sırasızdır

• **TreeSet** sınıfı da `Set` arayüzünü kurgular

–`HashSet` 'den daha yavaştır, ama öğelerini sıraya koyar; bu demektir ki yapı bir `SortedSet` olur.

• **HashMap** sınıfı `Map` arayüzünü kurgular

– hızlı erişim sağlar, ama öğeleri sırasızdır

•**TreeMap** sınıfı da `Map` arayüzünü kurgular

–`HashSet` ‘den daha yavaştır, ama öğelerini sıraya koyar; bu demektir ki yapı bir `SortedSet` olur.

Uyarı

`List` bir arayüz olduğu için bir `List nesnesi` yaratılamaz. Ama bir `ArrayList` ya da `LinkedList` nesnesi yaratılabilir. Bunların her ikisi de `List` arayüzünün işlevselliğine sahip olur.