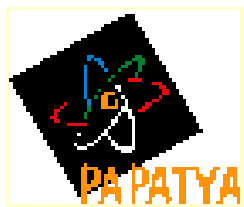
Nesneler için torbalar

(Collections)



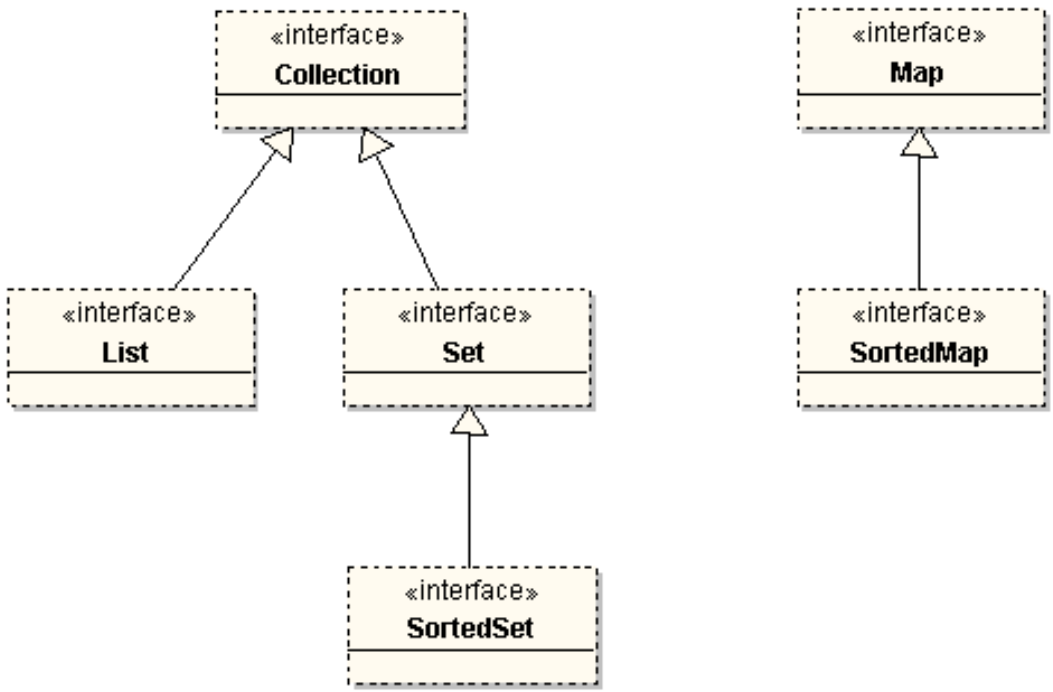
NESNELER İÇİN TORBALAR

- Torbalar birden çok nesneyi aynı çatı altında toplamak için kullanılır.
- Bunun faydası torba içerisinde bulunan nesnelerin daha kolay taşınmasıdır.
- En basit torba dizilerdir.

DiziOrnekBir.java

Torba Sistemleri

- Bir uygulama yazarken çoğu zaman ne kadarlık bir verinin dizi içerisine konacağı kestirilemez.
- Bu probleme çözüm olarak *java.util* paketinin altındaki arayüzler ve sınıflar kullanılabilir.

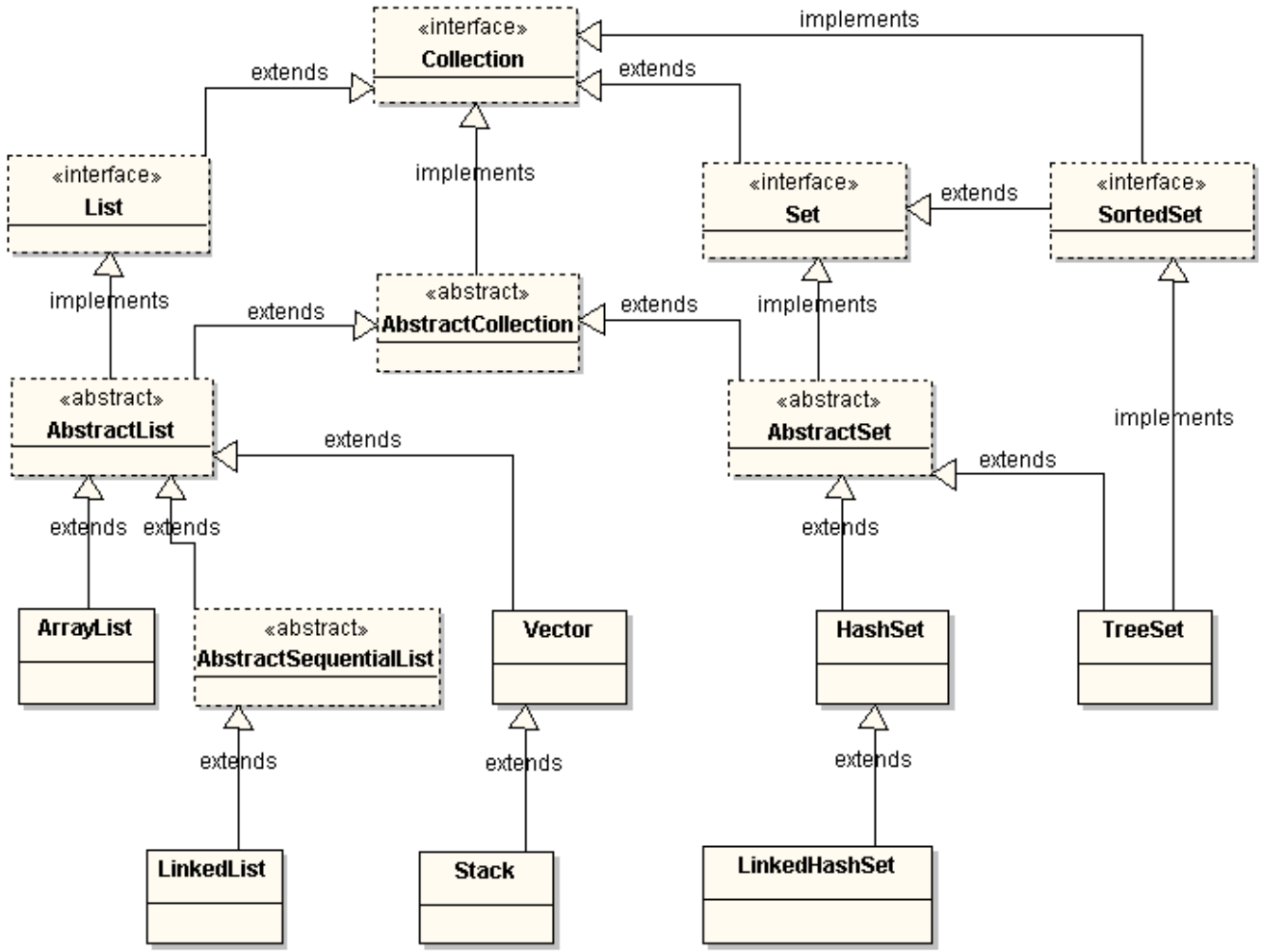


Collection Arayüzü

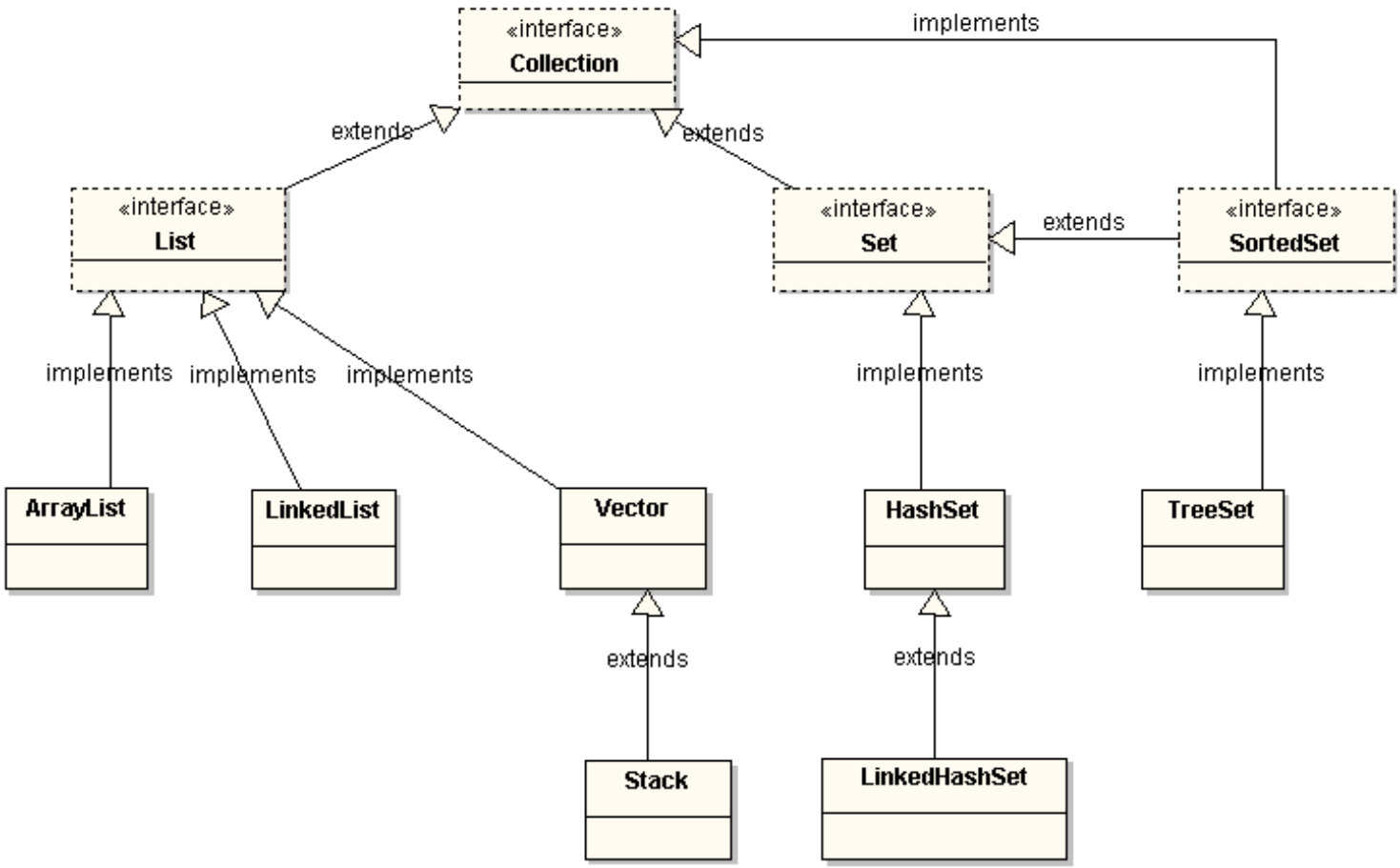
- *Collection* arayüzüne erişen sınıfların bir kısmı kendisine gelen tüm nesneleri (aynı olsalar dahi) kabul ederken, kimisi tamamen ayrı nesneler kabul etmektedir.
- Yine bu arayüze erişen bazı sınıflar, içerisindeki elemanları sıralı şekilde tutarken kimisi sırasız bir şekilde tutmaktadır.

CollectionTest.java

Detaylı Şema



Collection arayüzüne erişen diğer arayüzler ve sınıflar – Detaysız Şema (Soyut sınıflar çıkartılmış)



List Arayüzüne Erişen Sınıflar – ArrayList Sınıfı

- Genel olarak *List* arayüzüne erişen sınıflara ait nesnelerin kullanımı basittir.
- *List* arayüzüne erişen sınıflar, aynı diziler gibi sıfırıncı indeksten başlarlar.
- *ArrayList* nesnesinin içerisine eleman atmak için **`add()`**, içerideki bir elemanı almak için ise **`get()`** yordamı kullanılır.

```
yeniBoyut=(eskiBoyut*3) / 2 +1
```

ArrayList Sınıfı ve Iterator Arayüzü

- *ArrayList* sınıfı denince akla hemen *Iterator* arayüzüne erişmiş nesneler gelir.
- *Iterator* arayüzü tipindeki nesneler gerçekten çok basit ve kullanışlıdır.

Yordam İsmi	Açıklama
`boolean hasNext()`	İçeride hala eleman var ise **`true`** cevabını geri döner.
`Object next()`	Bir sonraki elemanı çağırır.
`void remove()`	**`next()`** yordamı ile çağrılmış olan elemanı siler. Bu yordam **`next()`** yordamından sonra çağrılmalıdır.

NufusCalismasi.java

Acaba Torbaya Ne Koymuştum?

- *ArrayList* nesnesinin içerisine atılan nesneleri almak için **`get()`** yordamı kullanılır.
- Bu yordam, içerideki nesneleri *Object* sınıfı tipinde bizlere geri döner.
- Gerçek tipi *Object* sınıfı tipinde olmayan bu nesnelerimizi daha sonradan aşağıya çevirim (*downcasting*) ile gerçek tiplerine çevirmemiz gereklidir.

SuperMarket.java

Garantili Torbalar

```
import java.util.ArrayList;
import java.util.Iterator;

public class StringArrayList {

  private ArrayList al = new ArrayList();

  public void add(String s) {
    al.add(s);
  }

  public String get(int indeks) {
    return (String)al.get(indeks);
  }

  public Iterator iteratorAl() {
       return al.iterator();
  }
}
```

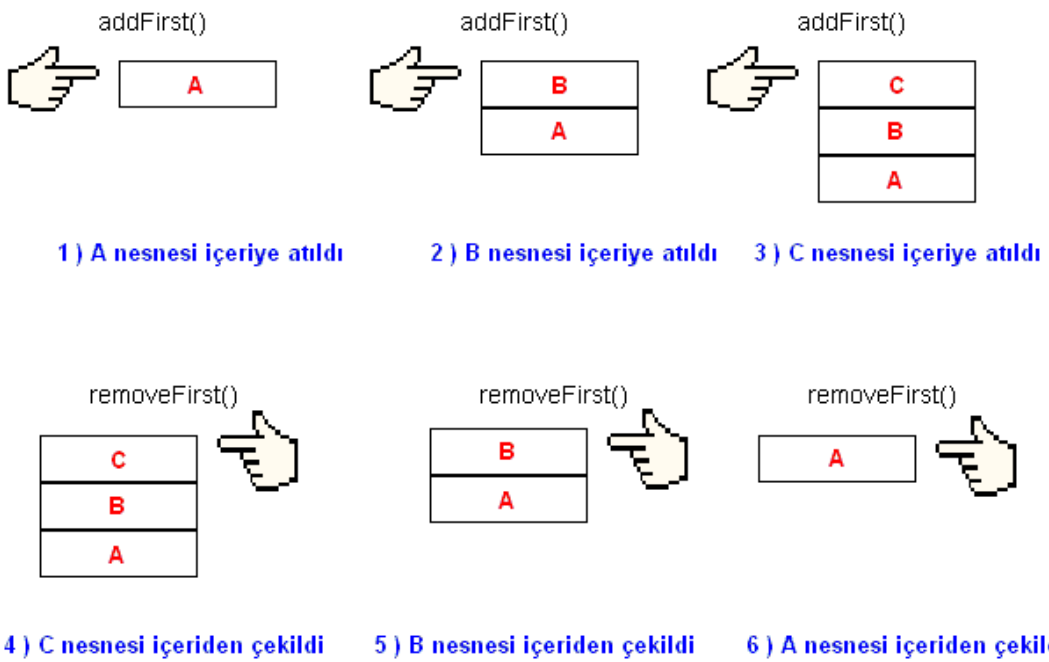
LinkedList Sınıfı

- *List* arayüzüne erişen bir başka sınıf ise *LinkedList* sınıfıdır.
- Bu sınıf da aynı *ArrayList* sınıfı gibi nesnelerin toplu olarak taşınmasında görev alır.
- *LinkedList* sınıfının *ArrayList* sınıfına göre bazı gelişmiş özellikleri bulunur.

LinkedListTestBir.java

LinkedList Sınıfı Kullanarak Yığın Yapısı Oluşturmak

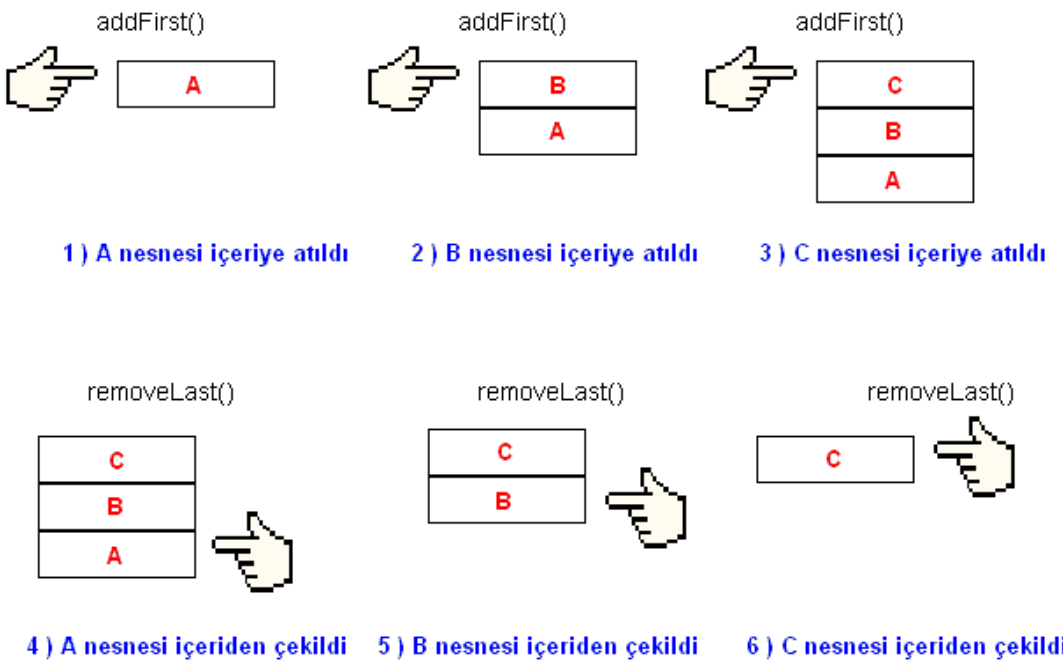
- *LinkedList* sınıfı tipindeki nesneye ait olan **`addFirst()`** ve **`removeFirst()`** yordamları kullanılarak veri yapılarındaki yığın yapısını tasarlamak mümkündür. (LIFO- Last in first out).



Yigin.java

LinkedList Sınıfı ile Kuyruk Yapısı Oluşturmak

- Kuyruk yapılarındaki kural, içerisine atılan ilk elemanın yine ilk olarak çıkmasıdır (FIFO-First in first out).



Kuyruk.java

Collections Sınıfı

- *Collections* sınıfının *Collection* arayüzü ile kalıtımsal herhangi bir bağı yoktur.
- *Collections* sınıfının içerisinde bir çok faydalı statik yordam bulunur.
- Bu yordamlar sayesinde *Collection* veya *Map* arayüzüne erişmiş sınıflara ait nesnelerin içerinde bulunan elemanları sıralama, arama, en büyük elemanı ve en küçük elemanı bulma, v.b. işlemleri gerçekleştirmemiz mümkün olur.

Collections.sort

- Bir torba (ör : *ArrayList* nesnesi) içerisindeki elemanları küçükten büyüğe doğru sıralamak (veya tam ters sırada) için *Collections* sınıfına ait statik **`sort()`** yordamını kullanabiliriz.

 ***SiralamaBir.java***

 ***TerstenSiralama.java***

 ***NesneSiralama.java***

Soru ?

- Peki *String* veya *Integer* sınıf tipindeki nesnelere referansları *ArrayList* nesnesinin içerisine atmasak da bunun yerine kendi oluşturduğumuz ayrı bir sınıfa ait nesnenin referanslarını *ArrayList* nesnesinin içerisine atsak ve **`Collections.sort()`** yordamı ile sıralatmaya çalışırsak ne olur?

NesneSiralama.java

Kitap.java

OzgunSiralama.java

java.lang.Comparable

- **`compareTo()`** yordamının döndürmesi gereken sonuçlar.

Durum	**Döndürülen sonuç**
O anki nesne *(this)*, parametre olarak gelen nesneden küçükse	O anki nesne *(this)*, parametre olarak gelen nesneden küçükse
O anki nesne *(this)*, parametre olarak gelen nesneye eşitse	sıfır
O anki nesne *(this)*, parametre olarak gelen nesneden büyükse	pozitif tamsayı

 ***Kitap2.java***

 ***OzgunSiralama2.java***

Collections.min() ve Collections.max()

- Torba (ör:*ArrayList*) içerisindeki elemanların en büyüğünü ve en küçüğünü bulan *Collections* sınıfının statik olan **`max()`** ve **`min()`** statik yordamlarıdır.

MinMaxBulma.java

Collections.binarySearch()

- *Collections* sınıfı içerisinde bulunan statik **`binarySearch()`** yordamı ile arama işlemleri kolaylıkla yapılabilir.
- **`binarySearch()`** yordamı iki adet parametre alır.
  1. Birincisi arama yapılacak olan torba (ör: *ArrayList*)nesnesine ait referans.
  2. İkincisi ise aratılan nesneye ait referans.

AramaTestBir.java

Hangisi Daha Hızlı ArrayList Sınıfı mı, LinkedList Sınıfı mı?

- Şu ana kadar olan örneklerimizin bazılarında *ArrayList* sınıfı bazılarında ise *LinkedList* sınıfı kullanılmıştır.
- Bu iki sınıfın amacı diğer nesnelerin toplanması için torba görevi görmektir.

 ***HizTesti1.java***  ***HizTesti2.java***

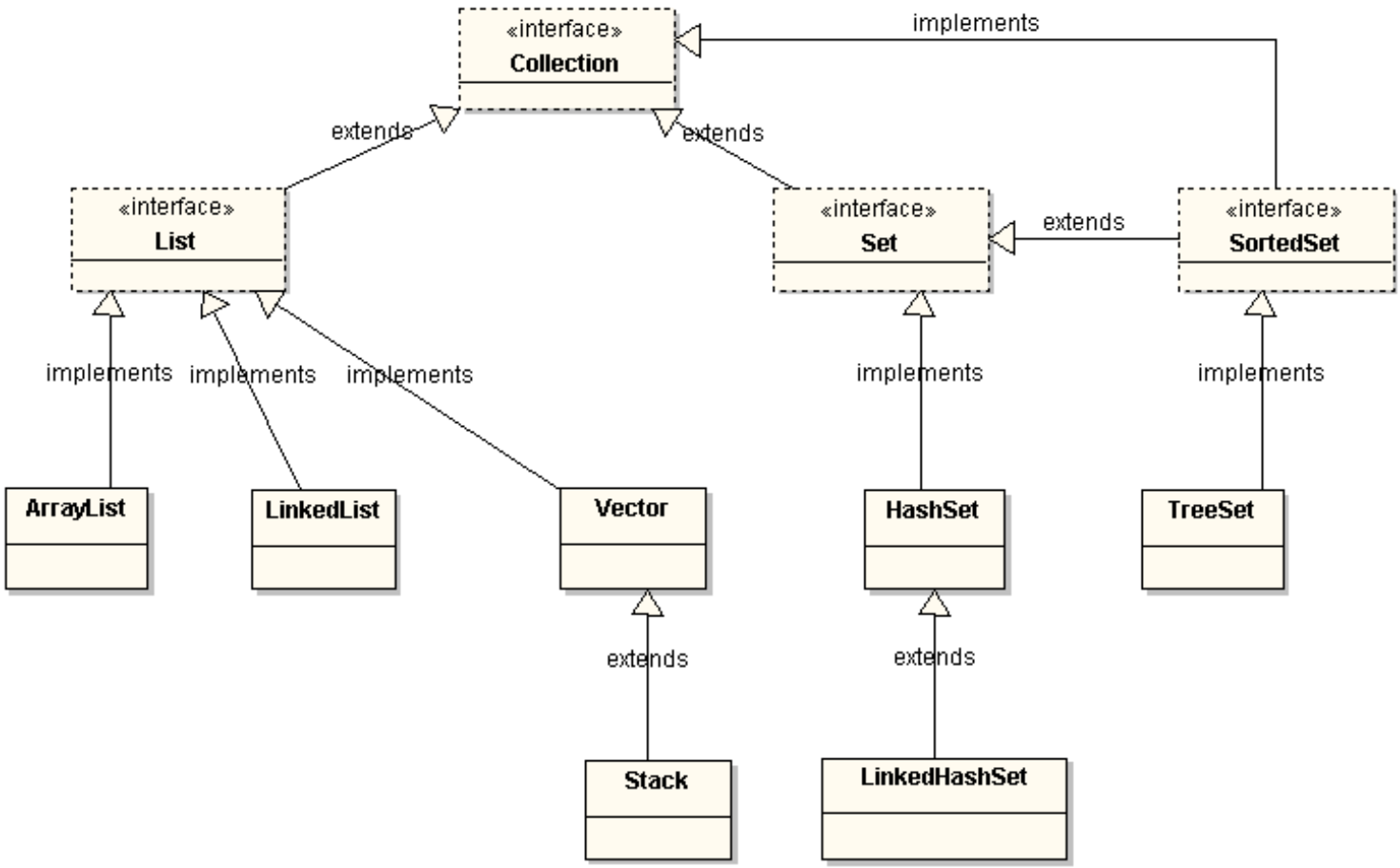
Sonuç

- Arama işlemlerinde *ArrayList* sınıfı en iyi performası verir.
  - *ArrayList* sınıfı, *RandomAccess* arayüzüne erişir ama *LinkedList* erişmez ve bu yüzden **`Collections.binarySearch()`** yordamı *ArrayList* üzerinde en iyi performası verir.
- LinkedList sınıfınında iyi olduğu yerler vardır.
  - Örneğin ters çevirme işlemi -ki bu işlem için `Collections.reverse()` yordamı kullanılır;
  - Ayrıca elemanlar arasında baştan sona veya sondan başa doğru sıralama *(iteration)* işlemlerinde de
  - Eleman ekleme çıkartma işlemlerinde LinkedList sınıfı kullanılır

Set Arayüzü

- Set arayüzü *Collection* arayüzünden türetilmiş.
- Set arayüzüne erişen sınıflara ait nesnelerin içerisine aynı elemanı iki kere atamayız.
- Birbirine eşit iki nesneye ait referansı, *Set* arayüzüne erişen bir nesnenin içerisine atamayız.

Collection arayüzüne erişen diğer arayüzler ve sınıflar – Detaysız Şema



HashSet Sınıfı

- *HashSet* sınıfı *Set* arayüzüne erişmiştir.
- Bunun doğal sonucu olarak da *Set* arayüzü içerisindeki gövdesiz yordamlara gövde yazmıştır.

HashSetTestBir.java

TreeSet Sınıfı

- *TreeSet* sınıfı *SortedSet* arayüzüne erişmiştir.
- *TreeSet* sınıfına ait bir nesnenin özelliği, içerisindeki elemanları sıralı (artan sırada) bir şekilde tutmasıdır.
- *TreeSet* sınıfına ait nesnenin içerisine atılacak olan referanslara bağlı nesnelere ait sınıfların kesin olarak *Comparator* arayüzüne erişmiş ve bu arayüzün içerisindeki gövdesiz yordamları iptal etmeleri gerekmektedir.

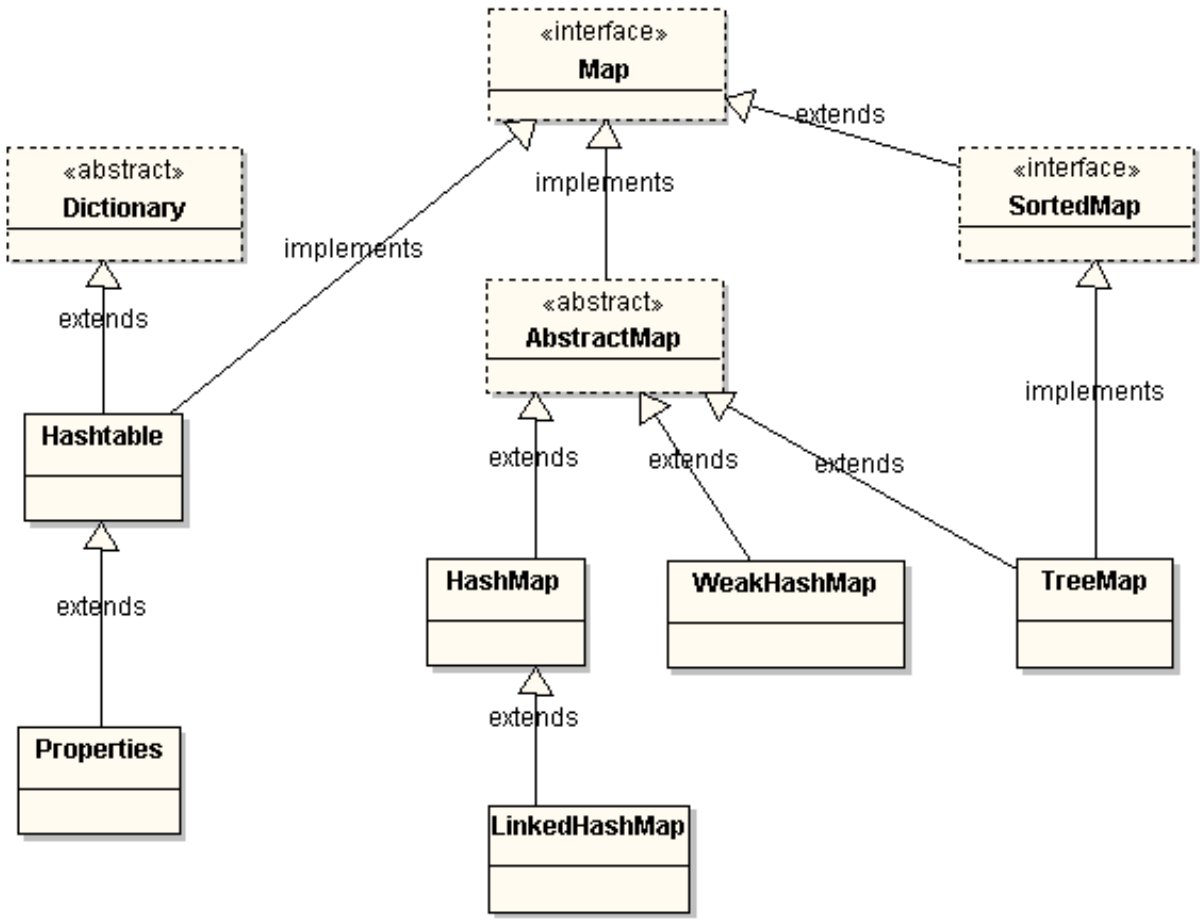
TreeSetTestBir.java

Map Arayüzü

- Uygulama yazarken ihtiyaç duyulan en büyük ihtiyaçlardan biri de anahtar-değer *(key-value)* ilişkisidir.
- Bu anahtar-değer ilişkisini ufak bir veritabanı gibi düşünebilirsiniz.

Plaka kodu (anahtar)	**İl (değer)**
06	Ankara
07	Antalya
41	Kocaeli
34	İstanbul
77	Yalova
....	....

Map arayüzüne erişen arayüz, soyut sınıflar ve sınıflar



HashMap Sınıfı

- *HashMap* sınıfı *Map* arayüzüne erişen sınıflarımızdan bir tanesidir.
- Bu sınıfın rolü, kodu yazan kişiye anahtar-değer ilişkisi oluşturabileceği bir ortam sunmaktır.

HashMapTestBir.java

HashMapTestIki.java

HashMap sınıfına ait bir nesneyi bir çok iş için kullanabiliriz.

- Bu tablodan Müşteri1, Müşteri2 ve Müşteri3'ün yaptıkları ödemelerin toplamını kolayca nasıl bulabiliriz ?

No	Müşteri adı	Ödenen tutar
1	Müşteri3	10
2	Müşteri2	5
3	Müşteri2	140
4	Müşteri3	5
5	Müşteri1	90
6	Müşteri2	15
7	Müşteri1	40
8	Müşteri3	65
9	Müşteri2	15
10	Müşteri1	25

Musteri.java

Kasa.java

Hangisi Daha Hızlı ArrayList Sınıfı mı, HashMap Sınıfı mı?

- *ArrayList* ve *HashMap* her ne kadar farklı yapıda olsalar da biri diğerinin yerine kullanılabilir.

HizTesti3.java

TreeMap Sınıfı

- *TreeMap* sınıfı *SortedMap* arayüzüne erişir.
- *TreeMap* sınıfına ait bir nesne kullanılarak aynı *HashMap* sınıfına ait nesnelerde olduğu gibi anahtar-değer ilişkilerini saklamak mümkündür.
- *TreeMap* sınıfına ait bir nesne kullanmanın avantajı anahtar-değer ilişkisindeki anahtarın sıralı bir biçimde tutulmasıdır.

TreeMapTestBir.java

Hangisi Daha Hızlı: HashMap Sınıfı mı, TreeMap Sınıfı mı?

- Bu sorunun cevabı şu şekilde olabilir: "anahtar-değer ilişkisindeki anahtarın sıralı olmasını istiyorsam *TreeMap*, aksi takdirde *HashMap* sınıfını kullanırım."
- Olaylara bu iki sınıfa ait nesnelerin içerisindeki anahtarın aratılma hızları açısından bakarsak, acaba olaylar nasıl değişir?

HizTesti4.java

Iterator Arayüzü ve Dikkat Edilmesi Gereken Hususlar

- Bu arayüz tipinde bir nesne elde etmek için *Collection* arayüzüne erişen sınıflara ait nesnelerin iterator() yordamını çağırmak yeterlidir.
- *Iterator* arayüzüne erişen bir nesnenin kullanılmasının sebebi kolaylıktır.
- *Iterator* arayüzüne ait nesneler ilgili torbanın elemanların belli bir andaki fotoğrafını çekip daha sonradan bu elemanları başka bir yere kopyalayıp, onların üzerinde mi işlem yapıyor?

IterationOrnek.java

Senkronize Torbalar

- Şu ana kadar incelediğimiz sınıflara ait nesnelerin hiç biri senkronize değildi.
- Bunun anlamı bu nesnelere (*ArrayList*, *LinkedList*, *HashSet*, *HashMap*...) aynı anda iki veya daha fazla sayıdaki iş parçacığının(*threads*) erişip istedikleri eklemeyi veya silme işlemini yapabileceğidir.

SenkronizeListTestBir.java

SenkronizeListTestIki.java

Sorular ...