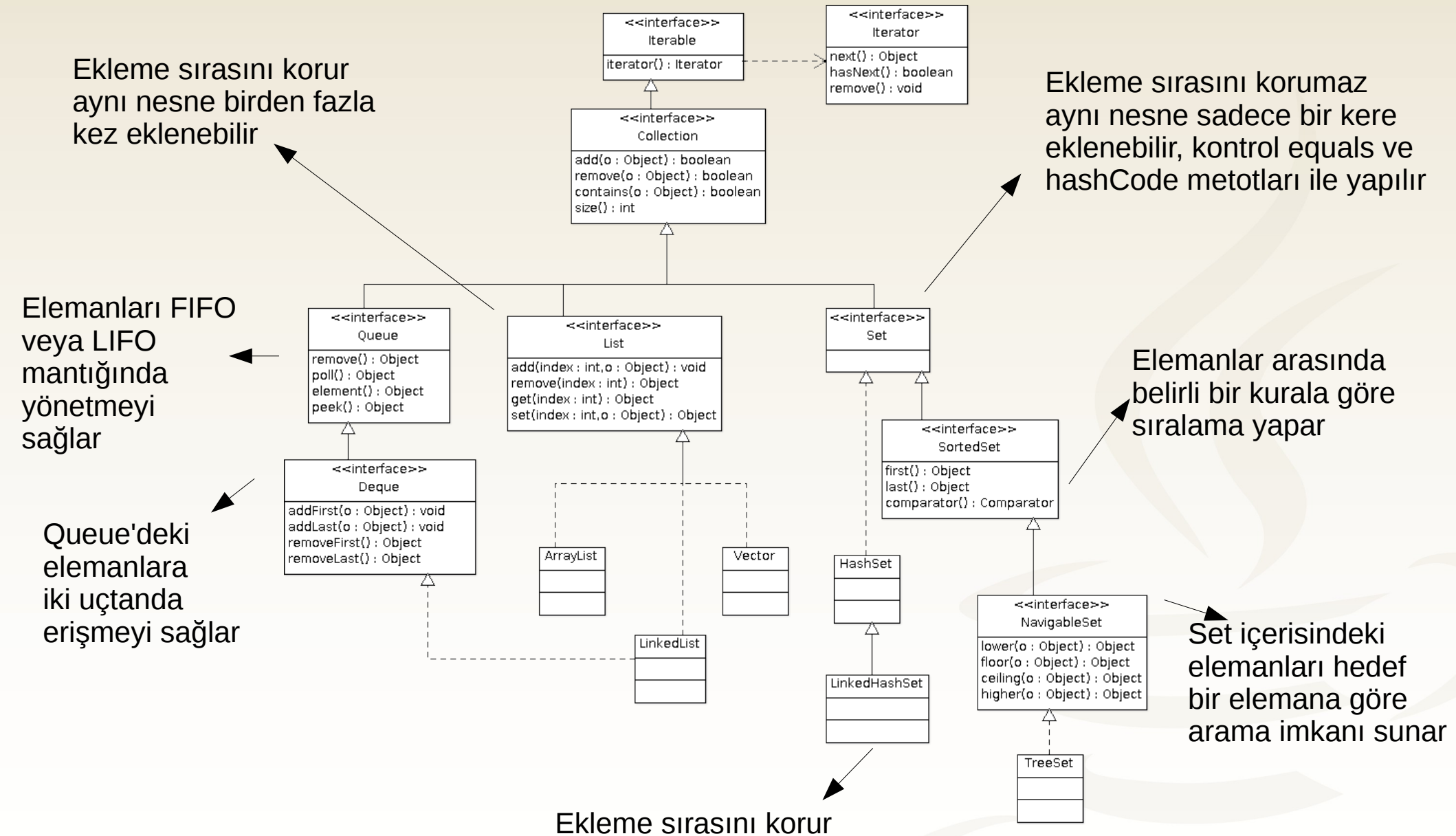
Java Collection API

harezmi bilişim çözümleri

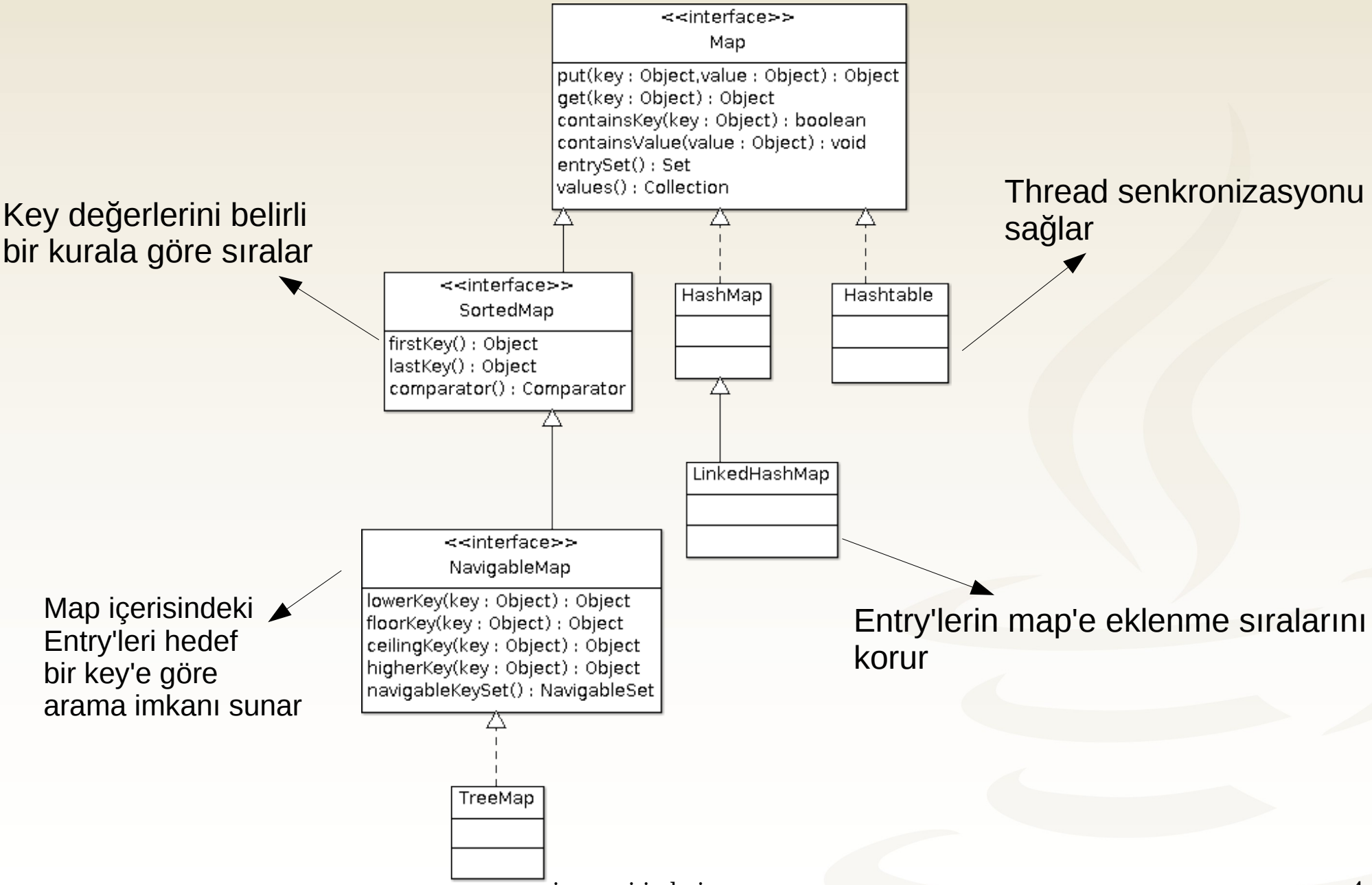
Collection Nedir?

- Bir grup Java nesnesinin **bir arada** tutulması ve **beraber** ele alınmalarını sağlayan yapılardır
- Array'e benzerler, ancak eleman sayıları **dinamik** olarak değişebilir
- Bu yapılar **iki temel kategori**de ele alınabilir
  - Collection
  - Map
- **Container** sınıflar olarak da bilinirler

Java'da Collection Hiyerarşisi



Map Hiyerarşisi



Java Collection API Örnekleri


```
public class Pilot {

   private String adi;
   private String soyadi;

   public Pilot(String adi, String soyadi) {
      this.adi = adi;
      this.soyadi = soyadi;
   }

   public String toString() {
      return adi + " " + soyadi;
   }
}
```

Java Collection API Örnekleri

```
public boolean equals(Object o) {
   if(o == null) return false;
   if(this == o) return true;
   if(!Pilot.class.isAssignableFrom(o.getClass()))
      return false;
   Pilot p = (Pilot)o;
   return this.adi != null
         && this.adi.equals(p.adi)
         && this.soyadi != null
         && this.soyadi.equals(p.soyadi);
}

public int hashCode() {
   int result = 1;
   result *= 37 + adi != null ? adi.hashCode() : 0;
   result *= 37 + soyadi != null ? soyadi.hashCode() : 0;
   return result;
}
```

List

```
List pilotListesi = new ArrayList();

pilotListesi.add(new Pilot("Jenson","Button"));
pilotListesi.add(new Pilot("Mark","Webber"));
pilotListesi.add(1,new Pilot("Sebastian","Vettel"));
pilotListesi.add(new Pilot("Jarno","Trulli"));
pilotListesi.add(new Pilot("Mark","Webber"));

Iterator itr = pilotListesi.iterator();
while(itr.hasNext()) {
   System.out.println(itr.next());
}

Pilot pilot = (Pilot)pilotListesi.remove(3);
pilotListesi.set(2, pilot);
int size = pilotListesi.size();
```

```
Jenson Button
Sebastian Vettel
Mark Webber
Jarno Trulli
Mark Webber
```

Set

```
Set pilotSeti = new HashSet();

pilotSeti.add(new Pilot("Jenson","Button"));
pilotSeti.add(new Pilot("Mark","Webber"));
pilotSeti.add(new Pilot("Sebastian","Vettel"));
pilotSeti.add(new Pilot("Jarno","Trulli"));
pilotSeti.add(new Pilot("Mark","Webber"));

Iterator itr = pilotListesi.iterator();
while(itr.hasNext()) {
   System.out.println(itr.next());
}

boolean result = pilotSeti.remove(new Pilot("Jarno","Trulli"));

int size = pilotSeti.size();
```

```
Jarno Trulli
Sebastian Vettel
Jenson Button
Mark Webber
```

LinkedHashSet

```
Set pilotSeti = new LinkedHashSet();

pilotSeti.add(new Pilot("Jenson","Button"));
pilotSeti.add(new Pilot("Mark","Webber"));
pilotSeti.add(new Pilot("Sebastian","Vettel"));
pilotSeti.add(new Pilot("Jarno","Trulli"));
pilotSeti.add(new Pilot("Mark","Webber"));

Iterator itr = pilotSeti.iterator();
while(itr.hasNext()) {
   System.out.println(itr.next());
}

boolean result = pilotSeti.remove(new
Pilot("Jarno","Trulli"));

int size2 = pilotSeti.size();
```

```
Jenson Button
Mark Webber
Sebastian Vettel
Jarno Trulli
```

TreeSet

```
SortedSet pilotSeti = new TreeSet();
pilotSeti.add(new Pilot("Jenson","Button"));
pilotSeti.add(new Pilot("Mark","Webber"));
pilotSeti.add(new Pilot("Sebastian","Vettel"));
pilotSeti.add(new Pilot("Jarno","Trulli"));
pilotSeti.add(new Pilot("Mark","Webber"));

Iterator itr = pilotSeti.iterator();
while(itr.hasNext()) {
   System.out.println(itr.next());
}

boolean result = pilotSeti.remove(new
Pilot("Jarno","Trulli"));

int size2 = pilotSeti.size();
```

Jarno Trulli
Jenson Button
Mark Webber
Sebastian Vettel

Pilot sınıfının **Comparable** interface'ini implement etmesi ya da SortedSet'e bir **Comparator** nesnesi verilmesi gerekiyor

Map

```
Map pilotMap = new HashMap();

pilotMap.put(new Pilot("Jenson","Button"),"Brawn GP");
pilotMap.put(new Pilot("Mark","Webber"),"Red Bull Racing");
pilotMap.put(new Pilot("Sebastian","Vettel"),"Red Bull
Racing");
pilotMap.put(new Pilot("Jarno","Trulli"),"Toyota");
pilotMap.put(new Pilot("Mark","Webber"),"Honda");

for(Object p:pilotMap.keySet()) {
   System.out.println(p + " : " +
         pilotMap.get(p));
}

Object result = pilotMap.remove(new
Pilot("Jarno","Trulli"));

int size2 = pilotMap.size();
```

```
Jarno Trulli : Toyota
Sebastian Vettel : Red
Bull Racing
Jenson Button : Brawn GP
Mark Webber : Honda
```

TreeMap

```
Map pilotMap = new TreeMap();

pilotMap.put(new Pilot("Jenson","Button"),"Brawn GP");
pilotMap.put(new Pilot("Mark","Webber"),"Red Bull
Racing");
pilotMap.put(new Pilot("Sebastian","Vettel"),"Red Bull
Racing");
pilotMap.put(new Pilot("Jarno","Trulli"),"Toyota");
pilotMap.put(new Pilot("Mark","Webber"),"Honda");

for(Object p:pilotMap.keySet()) {
   System.out.println(p + " : "
      + pilotMap.get(p));
}

Object result = pilotMap.remove(new
Pilot("Jarno","Trulli"));

int size2 = pilotMap.size();
```

```
Jarno Trulli : Toyota
Jenson Button : Brawn GP
Mark Webber : Honda
Sebastian Vettel : Red Bull
Racing
```

Pilot sınıfının **Comparable** interface'ini implement etmesi Ya da TreeMap'e bir **Comparator** nesnesi verilmesi gerekiyor

LinkedHashMap

```
Map pilotMap = new LinkedHashMap();


pilotMap.put(new Pilot("Jenson","Button"),"Brawn GP");
pilotMap.put(new Pilot("Mark","Webber"),"Red Bull Racing");
pilotMap.put(new Pilot("Sebastian","Vettel"),"Red Bull Racing");
pilotMap.put(new Pilot("Jarno","Trulli"),"Toyota");
pilotMap.put(new Pilot("Mark","Webber"),"Honda");

for(Object p:pilotMap.keySet()) {
    System.out.println(p +
        " : " + pilotMap.get(p));
}
```

```
Jenson Button : Brawn GP
Mark Webber : Honda
Sebastian Vettel : Red Bull Racing
Jarno Trulli : Toyota
```

```
Object result = pilotMap.remove(new Pilot("Jarno","Trulli"));

int size2 = pilotSeti.size();
```

Java Collection API'deki Sınıf Türleri

- Java Collection API/Framework içerisinde üç farklı türde sınıf mevcuttur
  - **Interface**: Collection, List, Set, Map, ...
  - **Abstract class**: AbstractList, AbstractSet, AbstractMap...
  - **Concrete class**: ArrayList, HashSet, HashMap...
- Abstract sınıfları kullanarak **kendi container implemantasyonlarımızı** da geliştirebiliriz

Java Collection API'deki Sınıf Türleri

JAVA Eğitimleri

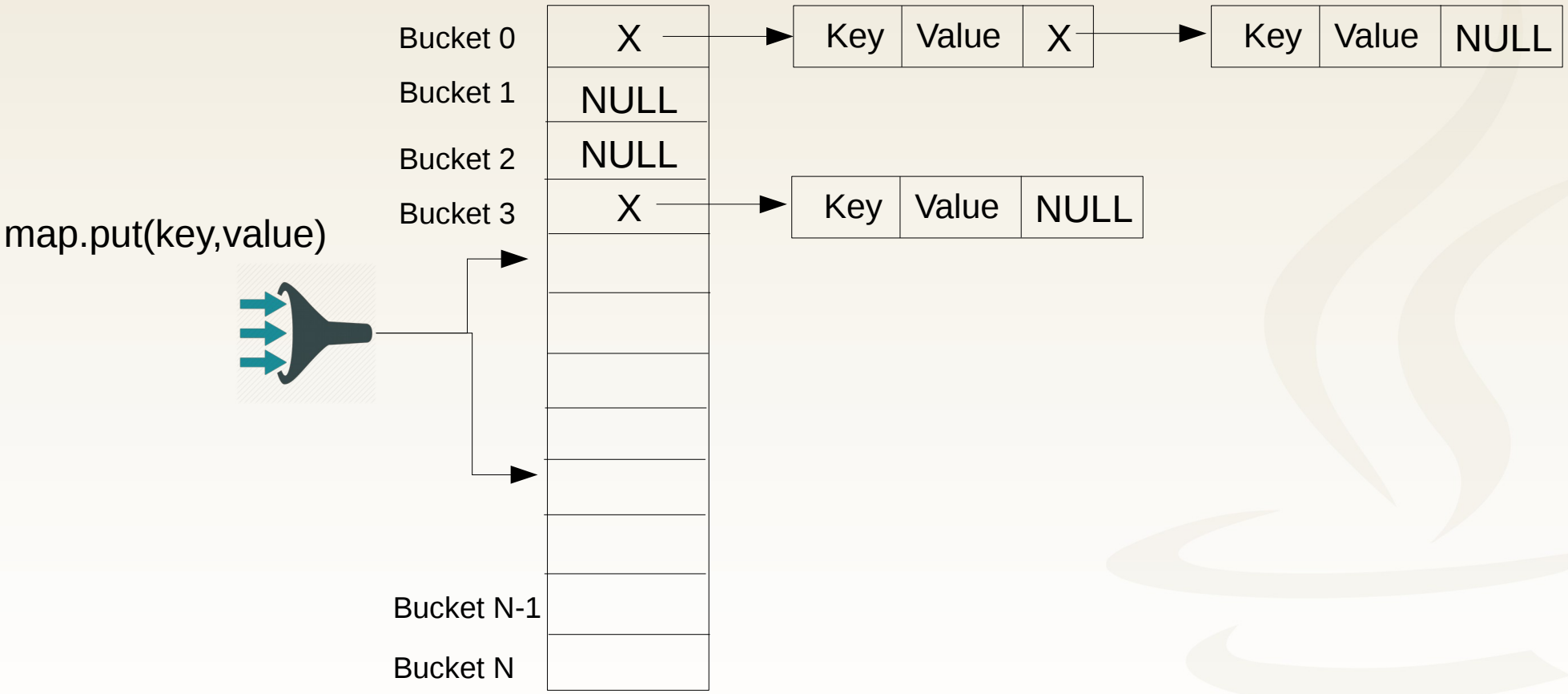
- ArrayXXX: implemantasyonunda **array** stratejisi kullanılmıştır
- LinkedXXX: implemenasyonunda **pointer link**ing stratejisi kullanılmıştır
- HashXXX: implementasyonunda **hash** stratejisi kullanılmıştır
- TreeXXX:implementasyonunda **binary tree** stratejisi kullanılmıştır

HashSet ve HashMap Nasıl Çalışır?

JAVA Eğitimleri

- HashMap'in'içerisine yeni bir key:value eklenirken **önce key'in hash'i** elde edilir
- Hash değerine göre **uygun bucket** tespit edilir
- Bu bucket içerisinde başka entry mevcut değilse, **ilk olarak bu entry** yerleştirilir
- Mevcut bir entry varsa key değerinin **equals metodu** ile entry'ler kontrol edilir
- Eşleşen varsa yeni entry ile değiştirilir
- Eşleşen yoksa yeni entry bucket'ın en sonuna eklenir

HashSet ve HashMap Nasıl Çalışır?



HashSet ve HashMap İlişkisi

JAVA Eğitimleri

- HashSet arka **tarafta HashMap** kullanarak çalışır
- HashMap değerleri **key:value** ikilisi olarak saklar
- HashSet elemanları Map içerisinde key olarak saklanır, value olarak da sabit bir nesne kullanılır

HashSet ve HashMap Performansı

- Map'in kapasitesi dolmaya yaklaştıkça **load factor'e göre** alan genişletilerek **yeniden hash hesaplaması** yapılır
- Bu işlem de **maliyetli**dir, yeniden hash yapılması mümkün olduğunca azaltılmalıdır
- Dolayısı ile Set ve Map için **initialCapacity** ve **loadFactor** değerleri önemlidir
- Default loadFactor **0.75**'tir ve çoğu senaryo için **optimal bir değer**dir

HashSet ve HashMap Performansı

- Default initialCapacity ise **16**'dır
- Set/Map'deki mevcut eleman sayısı **16 x 0.75** değerini aştığı vakit alan genişletmesi ve yeniden hash hesaplaması yapılır
- Set/Map içerisinde tutulacak elemanların sayısına göre **initialCapacity artırılabilir**
- Set ve Map içerisine konan elemanların hashCode metotlarının **benzersiz hash değerleri** üretir biçimde yazılması önemlidir
- Hash **çakışması** arttıkça performans düşecektir

Comparable ve Comparator Arayüzleri

- Java nesnelerini **sıralama amaçlı karşılaştırma**yı sağlayan yapılardır
- **Comparable** arayüzünü implement eden sınıf kendisini diğer bir nesne ile karşılaştırabilir
- Comparable arayüzü üzerinden karşılaştırmaya "**natural ordering**" adı verilmektedir
- **Comparator** nesnesi ise iki nesneyi birbirleri arasında karşılaştırmaya yarar

Comparable

```
public class Personel implements Comparable<Personel>{
   private int id;
   private String adi;
   private String soyadi;

   public Personel(int id, String adi, String soyadi) {
      this.id = id;
      this.adi = adi;
      this.soyadi = soyadi;
   }

   public int compareTo(Personel p) {
      if(id < p.id) return -1;
      else if(id == p.id) return 0;
      else return 1;
   }
}
```

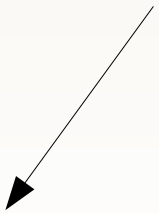
Negatif değer: this < p

Sıfır: this.equals(p)

Pozitif: this > p

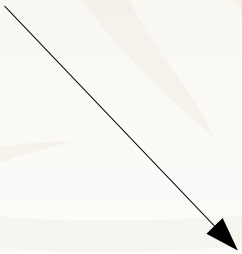
Comparator

```
public class PersonelDogumTarihiComparator
                implements Comparator<Personel>{

   public int compare(Personel p1, Personel p2) {
      long p1DTarih = p1.getDogumTarihi().getTime();
      long p2DTarih = p2.getDogumTarihi().getTime();

      if(p1DTarih < p2DTarih) return -1;
      else if (p1DTarih == p2DTarih) return 0;
      else return 1;
   }

}
```



Pozitif : p1 > p2

Negatif : p1 < p2



Sıfır : p1.equals(p2)

```
Set personelSet = new TreeSet(new PersonelDogumTarihiComparator());
```

İletişim

www.harezmi.com.tr

www.java-egitimleri.com

info@harezmi.com.tr

info@java-egitimleri.com

@HarezmiBilisim

@JavaEgitimleri