



Java ile Nesne Merkezli ve Fonksiyonel Programlama

10. Bölüm: Torbalar



Eğitmen:

Akın Kaldıroğlu

Çevik Yazılım Geliştirme ve Java Uzmanı



Konular

- **Torbalara Giriş**
- **Java'da Torbalara Giriş**
- **Collection**
 - Generics
 - Iterable ve Iterator
- **Nesne Sıralama**
 - Comparable ve Comparator
- **Set ve Gerçekleştirmeleri**
 - HashSet ve TreeSet
- **List ve Gerçekleştirmeleri**
 - ArrayList ve LinkedList
- **Diğer Collection Nesneleri**
 - Stack, Vector, Queue ve Deque
- **Map ve Gerçekleştirmeleri**
 - HashMap ve TreeMap
- **Dönüşümler ve Algoritmalar**
- **Diğer Torba Kütüphaneleri**
- **Ödevler**

Torbalar Giriş



```
115     assertEquals(successfulCustomer, loggedCustomer);
116 }
117
118 @Test // Test2
119 public void testUnsuccessfulLoginWithNoSuchCustomerException() {
120     assertThrows(NoSuchCustomerFoundException.class, () -> {
121         Customer unfoundCustomer = service.login(unfoundCustomerTckn, properPassword);
122     });
123 }
124
125 @Test // Test3
126 public void testUnsuccessfulLoginWithCustomerLockedException() {
127     assertThrows(CustomerLockedException.class, () -> {
128         Customer lockedCustomer = service.login_lockedCustomerTckn, "qwerty123");
129     });
130 }
131
132 @Test // Test4
133 public void testUnsuccessfulLoginWithCustomerAlreadyLoggedException() {
134     assertThrows(CustomerAlreadyLoggedException.class, () -> {
135         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTckn, properPassword);
136     });
137 }
138
139 @Test // Test5
140 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForTckn() {
141     assertThrows(ImproperCustomerCredentialsException.class, () -> {
142         Customer customer = service.login(shortTckn, properPassword);
143     });
144 }
145
146 @Test // Test6
147 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForPassword() {
148     assertThrows(ImproperCustomerCredentialsException.class, () -> {
149         Customer customer = service.login(properTckn, shortPassword);
150     });
151 }
152
153 @Test // Test7
154 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForBoth() {
155     assertThrows(ImproperCustomerCredentialsException.class, () -> {
156         Customer customer = service.login(propertckn, shortPassword);
157     });
158 }
159
160 @Test // Test8
161 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForAll() {
162     assertThrows(ImproperCustomerCredentialsException.class, () -> {
163         Customer customer = service.login(propertckn, shortPassword);
164     });
165 }
```

Torba Nedir? - I



- Birden fazla değişkeni veya veriyi (kısaca elemanı) bir arada tutan yapılara **torba (collection)** denir.
- Torbalar, ister ilkel tiplerden ister ise karmaşık tiplerden olsun, birden fazla değişkeni grup olarak tek bir yapı altında tutar ve yönetir.
- Torbalar, dilde diğer veriler gibi değişkenlerle ifade edilir ve metodlara geçilir, metodlardan döndürülür, vs.
- Programlama dillerinde farklı amaç ve özelliklerle oluşturulmuş torbalar vardır.

Torba Nedir? - II



- Üniversitelerde **Veri Yapıları (Data Structures)** adıyla öğreten dersler, temelde torbaların iç yapıları, dizilim, ulaşım vb. algoritmaları ile bellek, depolama vs. gibi mekanizmlara odaklanırlar.
- Bu anlamda torbalar, **Soyut Veri Yapıları'nın (Abstract Data Types)** bir parçasıdır.
- Torbalar, bir veri yapısı olarak, veri soyutlamasıdır (data abstraction).
- Torbalar, iç yapıları dışarıdan saklanmış (encapsulated) yapılardır ve sağladıkları API ile elemanların saklanması, ulaşılmasına ve işlenmesine izin verir.

Torba Nedir? - III



- C++ gibi bazı farklı dillerde **container**, diğer bazlarında **user/programmer-defined types** olarak da adlandırılır.

Torba Çeşitleri - I



- Yapısal olarak torbaların üç tipinden bahsedilebilir:
- **Doğrusal (linear) torbalar:** Elemanlarını dizili (ordered) tutan torbalardır.
 - Torbaların iç yapısı farklı olsa bile dışarıya açılan arayüzü, dizi (array, sequence) görünümündedir.
 - Dolayısıyla torbadaki elemanlar ardışıl (sequential) bir yapıdadır; elemanların pozisyonları vardır ve aralarında öncelik ve sonralık söz konusudur.
 - Dizi (array), **List**, **Vector**, **Queue**, ve **Dequeue** doğrusal torbalardandır.

Torba Çeşitleri - II



- **İlişkilendiren (associative) torbalar:** Bir girdiye karşılık bir çıktı üreten (mapping), fonksiyon gibi davranışlı torbalardır.
 - Torba dizi olarak görünmez ve torbadaki elemanlar farklı nesnelerle ilişkilendirilerek tutulur.
 - Bazen eleman, verilen bir anahtarla (key) ilişkilendirilerek saklanır,
 - Bu tür torbalara **sözlük (dictionary)** ya da **ilişkilendiren (ilişkili) dizi (associative array)** de denir çünkü elemanları **anahtar-değer ikilisi (key-value pair)** yapısındadır.
 - **Map** ve **Hashtable** ilişkilendiren, sözlük yapısındaki torbalardandır.

Torba Çeşitleri - III



- İlişkilendiren torbalarda bazen eleman, dışarıdan verilmeyen, torba tarafından üretilen bir veri ile ilişkilendirilerek tutulur.
- Bu tür yapılarda en sık kullanılan veri hash kodudur.
- Torba her eleman için tekil (unique) olarak ürettiği hash kodu kullanarak elemanı saklar.
- Saklanan elemansa bu şekilde erişim çok hızlı olur.
- **Set** hash kod kullanarak ilişkilendiren torbalardandır.

Torba Çeşitleri - IV



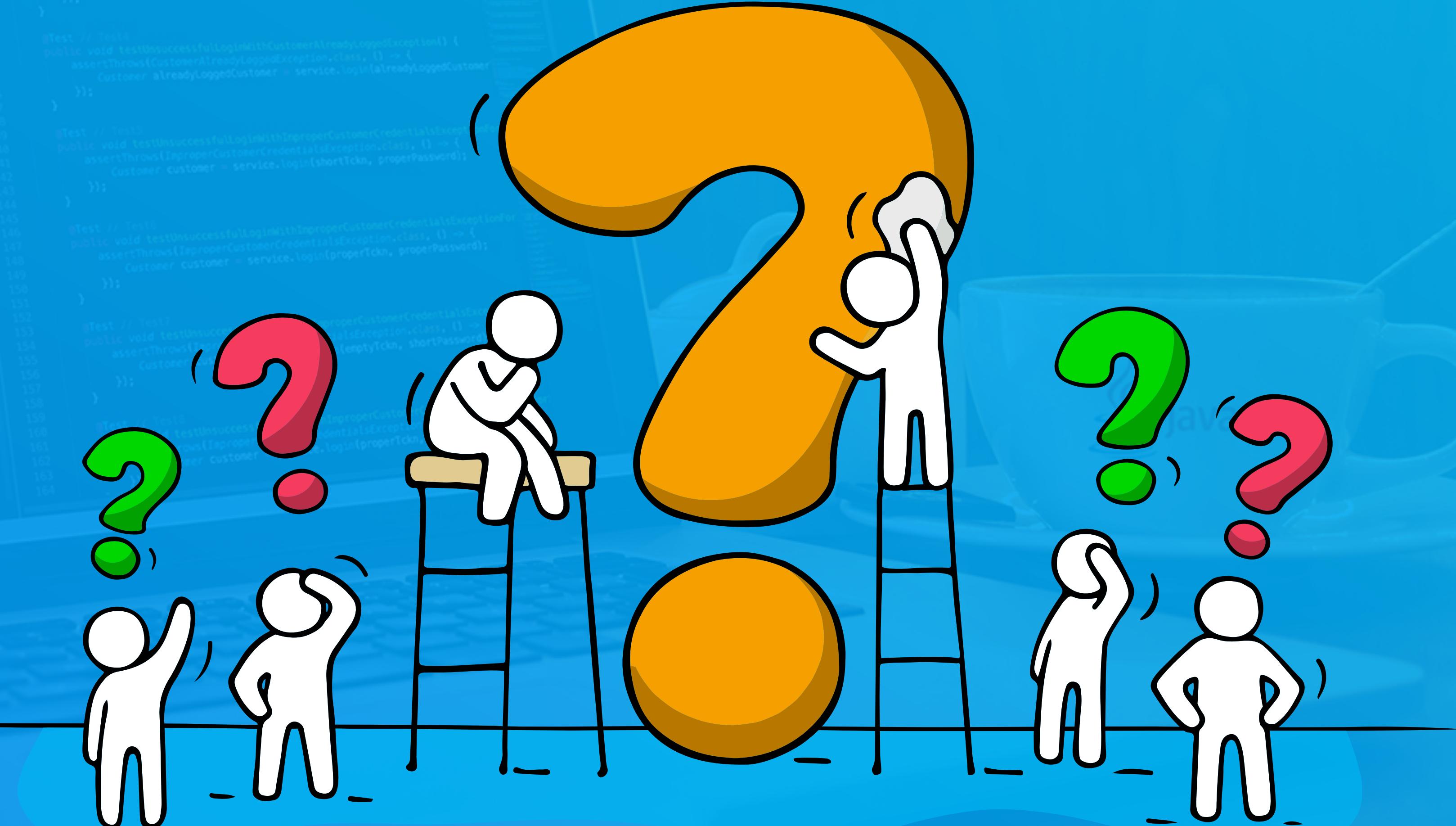
- **Graf (graph) torbalar:** Bir elemanı birden fazla elemanla ilişkilendiren torbalardır.

Torbaların Özellikleri



- Torbalarda farklı özellikler söz konusudur:
- Elemanları tekil olan torbalar ile aynı elemandan birden fazlasına izin veren torbalar,
- Elemanlarına erişimi dizideki yeri ile yapan torbalar ile anahtar kullanarak erişim sağlayan torbalar,
- Elemanlarını sıralı (sorted) tutan torbalar ile elemanlarını sıralamayan torbalar.
- Sadece aynı tipten elemanları tutan, homojen (homogeneous) torbalar ile farklı tipten elemanları tutan, heterojen (heterogeneous) torbalar.

Soru ve Cevap Zamani!



Java'da Torbalara Giriş

Java'da Torbalar



- Java, tüm diller gibi, pek çok torbaya sahiptir.
- Java SE'nin ilk sürümünde 3 tane torba vardı:
 - Dizi (array), boyutu değişmeyen ve homojen dizi,
 - **java.util.Vector**, dinamik ve heterojen dizi,
 - **java.util.Hashtable**, dinamik ve heterojen, sözlük (dictionary) yapısı.

Torba Çerçevesi - I



- Java ilk çıkışında sahip olduğu torba yapılarını 1.2 sürümünde genişletmiş ve bir **torba çerçevesi (collection framework)** haline getirilmiştir.
- Torba çerçevesi, Java'nın ilk sürümündeki **Vector** ve **Hashtable** yerine çok daha yetkin ve sistemli yeni yapılar getirmiştir.
- Diziler, kendine has kısıtlarıyla (özellikleriyle) hala yaygın bir şekilde kullanılmaktadır.

Torba Çerçevesi - II

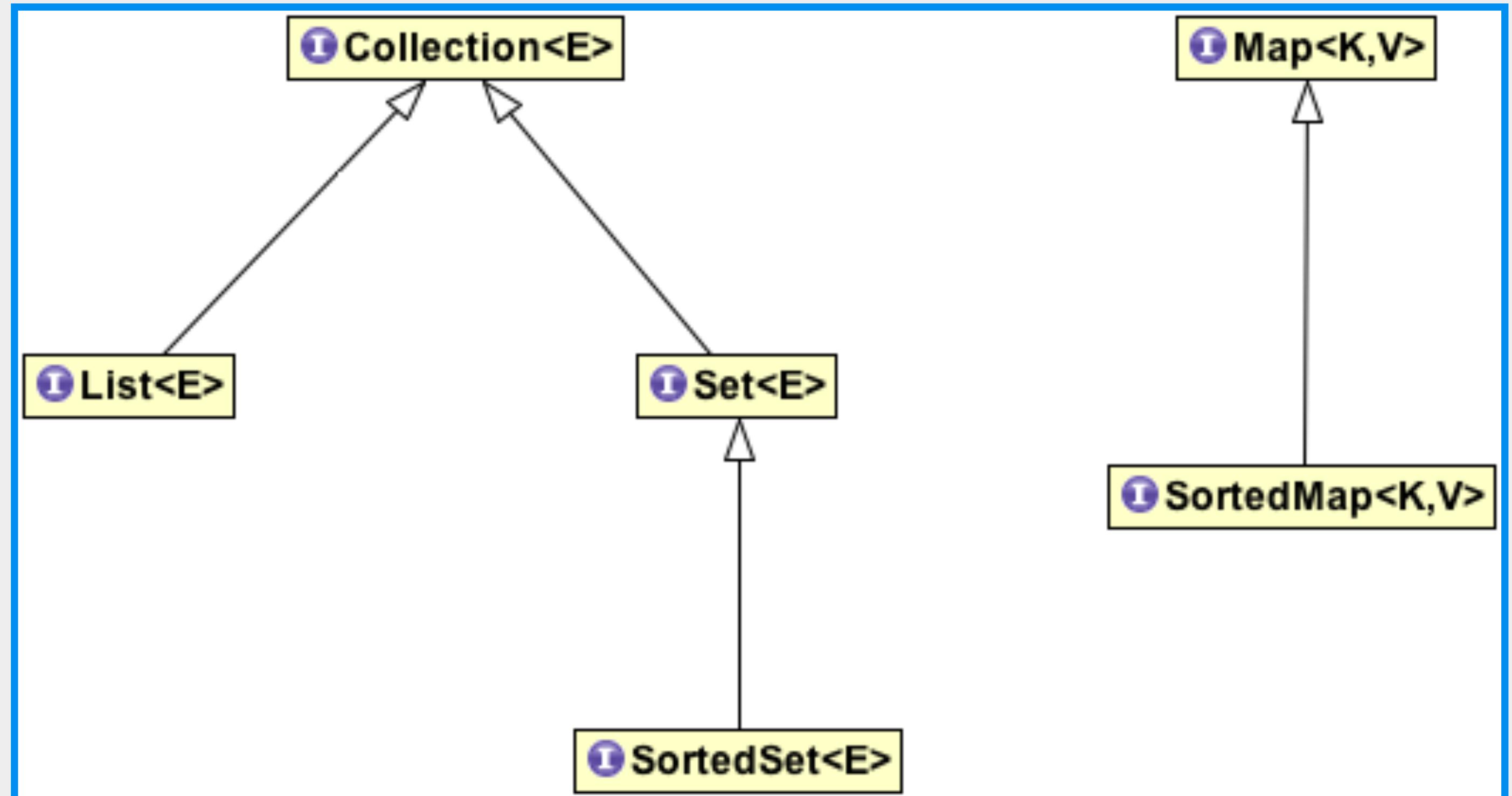


- Java'nın torba çerçevesi `java.util` paketindedir ve temelde şu yapılarından oluşmaktadır:
 - Arayüzler (interfaces)
 - Gerçekleştirmeleri (implementations)
 - Algoritmalar (algorithms) ve
 - yardımcı yapılar.

Arayüzler



- Java torbalarını iki farklı hiyerarşi ile ifade etmek mümkündür.
- İlk hiyerarşi **Collection** torbalarının ana arayüzlerini, ikinci hiyerarşi ise **Map** torbaların ana arayüzlerini göstermektedir.



Collection

Collection - I



- Collection arayüzü doğrusal torbaların kök arayüzüdür.
- Tuttuğu elemanlarla ilgili dizme, sıralama, teklik kontrolü vs. hiç bir kısıta sahip değildir.
- Doğrudan bir gerçekleştirmesi yoktur.
 - Alt arayuzlerinin gerçekleştirmeleri vardır.
- Tüm doğrusal torbalarda geçerli APIyi belirleyen arayüzdür.
- Tüm doğrusal torbalar aynı zamanda bir Collection nesnesidir.

Collection - II



- **Collection** arayüzü ilk halinde hepsi soyut olan 15 tane metoda sahipti.
- Java SE 8'de metot sayısı, eklenen 4 **default** metotla 19 olmuştur.
- Java SE 11'de metot sayısı, eklenen 1 **default** metotla 20 olmuştur.
- Dolayısıyla şu anda 15 soyut, 5 **default** metodu vardır.

Collection - III



- En genel torba davranışlarını bir araya getirmiştir:
 - Ekleme **add()**, **addAll()**, çıkarma **remove()**, **removeAll()**, **removeIf()**, **retainAll()** ve boşaltma **clear()**,
 - Elemanlara ulaşma **iterator()** ve **spliterator()**
 - **Collection**'da dizilim (öncelik-sonralık) ya da anahtar kavramı olmadığından, tuttuğu nesneleri almanın tek yöntemi **Iterator** arayüzüdür.

Collection<E>

- **add(E):boolean**
- **addAll(Collection<? extends E>):boolean**
- **clear():void**
- **contains(Object):boolean**
- **containsAll(Collection<?>):boolean**
- **equals(Object):boolean**
- **hashCode():int**
- **isEmpty():boolean**
- **iterator():Iterator<E>**
- **parallelStream():Stream<E>**
- **remove(Object):boolean**
- **removeAll(Collection<?>):boolean**
- **removeIf(Predicate<? super E>):boolean**
- **retainAll(Collection<?>):boolean**
- **size():int**
- **spliterator():Spliterator<E>**
- **stream():Stream<E>**
- **toArray():Object[]**
- **toArray(T[]):T[]**
- **toArray(IntFunction<T[]>):T[]**

Collection - IV



- Sorgulama **isEmpty()**, **contains()**, **containsAll()**, **size()**,
- Dönüşürme **toArray()** metotları,
- **equals()** ve **hashCode()**, **Object** metotları,
- **stream()**, **parallelStream()** gibi stream alma metotları.
- **Stream** ileride ayrı bir bölümde ele alınacaktır.

Collection<E>

- **add(E):boolean**
- **addAll(Collection<? extends E>):boolean**
- **clear():void**
- **contains(Object):boolean**
- **containsAll(Collection<?>):boolean**
- **equals(Object):boolean**
- **hashCode():int**
- **isEmpty():boolean**
- **iterator():Iterator<E>**
- **parallelStream():Stream<E>**
- **remove(Object):boolean**
- **removeAll(Collection<?>):boolean**
- **removeIf(Predicate<? super E>):boolean**
- **retainAll(Collection<?>):boolean**
- **size():int**
- **spliterator():Spliterator<E>**
- **stream():Stream<E>**
- **toArray():Object[]**
- **toArray(T[]):T[]**
- **toArray(IntFunction<T[]>):T[]**

Collection - V



- Tuttuğu elemanlarla ilgili dizme, sıralama, teklik kontrolü vs. hiç bir kısita sahip olmadığından ekleme metotları **add()** ve **addAll()** daima **true** döndürür.

CollectionExample.java



- org.javaturk.oofp.ch10.interfaces.CollectionExample



```
111 public void testSuccessfullLogin() {
112     // Given
113     // When
114     // Then
115     Customer loggedCustomer = service.login(properties, properPassword);
116     assertEquals(successfulCustomer, loggedCustomer);
117 }
118
119 @Test // Test 2
120 public void testSuccessfullLoginWithNoSuchCustomerException() {
121     // Given
122     // When
123     // Then
124     assertThrows(NoSuchCustomerNotFoundException.class, () -> {
125         Customer unfoundCustomer = service.login(unfoundCustomerTkn, properPassword);
126     });
127 }
128
129 @Test // Test 3
130 public void testSuccessfullLoginWithCustomerLockedException() {
131     // Given
132     // When
133     // Then
134     assertThrows(CustomerLockedException.class, () -> {
135         Customer lockedCustomer = service.login(lockedCustomerTkn, "password");
136     });
137 }
138
139 @Test // Test 4
140 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {
141     // Given
142     // When
143     // Then
144     assertThrows(CustomerAlreadyLoggedException.class, () -> {
145         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTkn, properPassword);
146     });
147 }
148
149 @Test // Test 5
150 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
151     // Given
152     // When
153     // Then
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(shortTkn, properPassword);
156     });
157 }
158
159 @Test // Test 6
160 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
161     // Given
162     // When
163     // Then
164     assertThrows(ImproperCustomerCredentialsException.class, () -> {
165         Customer customer = service.login(propertiesTkn, properPassword);
166     });
167 }
```

Generics

Heterojen Collection - I



- Java'ya generics özelliği gelmeden önce **Collection** heterojendi, en genel tip olan **java.lang.Object** ile çalışır ve her tipten nesnenin eklenmesine izin verirdi.
- Tüm metotları da **Object** alıp-verirdi:
 - **boolean add(Object o)** veya
 - **boolean addAll(Collection c)** gibi.
- Bu tabii olarak çok şekilliliğin (polymorphism) bir sonucudur.

Heterojen Collection - II



- Bunun negatif tarafı ise daha alt tiplerin nesnelerinin daima **Object** olarak görülmeleri ve gerçek tiplerini gösterememeleridir.
- Bu durumda **Collection**'dan alınan nesnenin gerçek tipine ulaşmak istendiğinde **instanceof** ve alçaltmaya (downcaste) ihtiyaç duyulur.

```
Collection col....  
Iterator it = col.iterator();  
while(it.hasNext()){  
    Employee s = (Employee) it.next();  
    ...  
}
```

Homojen Collection - I



- **Collection**, Java'ya 1.5 sürümünde gelen **generics** sayesinde sadece tek bir tipten olan nesneleri tutacak şekilde kullanılabilir.
- Bu şekilde **Collection**, **tek bir tipe has, özel (generic to a specified type ya da generic)** olur.
- Bu tek tip **E** ise, arayüz de **Collection<E>** olarak ifade edilir.
- Bu durumda **Collection<E>**, sadece **E** ve alt tiplerinin nesnelerini tuttuğundan homojen hale gelmiş olur.
- **Collection<E>**, **E** ve alt tipi dışındaki tiplerin nesneleri ile çalışmaz.

Heterojen Collection - II



- Bu durumda tüm metotları **E** tipi ile çalışır:
 - **boolean add(E e)** veya
 - **boolean addAll(Collection<? extends E> c)** gibi.
- Bu şekilde bir tipe has de **Collection<E>**'dan iterator yardımıyla alınan nesneler için **instanceof** ve alcaltmaya (downcasting) gerek kalmaz.

```
Collection<Employee> col = ...;
Iterator<Employee> it = col.iterator();
while(it.hasNext()){
    Employee e = it.next();
    ...
}
```



boolean	<code>add(E e)</code>
boolean	<code>addAll(Collection<? extends E> c)</code>
void	<code>clear()</code>
boolean	<code>contains(Object o)</code>
boolean	<code>containsAll(Collection<?> c)</code>
boolean	<code>isEmpty()</code>
<code>Iterator<E></code>	<code>iterator()</code>
boolean	<code>remove(Object o)</code>
boolean	<code>removeAll(Collection<?> c)</code>
default boolean	<code>removeIf(Predicate<? super E> filter)</code>
boolean	<code>retainAll(Collection<?> c)</code>
int	<code>size()</code>
<code>Object[]</code>	<code>toArray()</code>
<code><T> T[]</code>	<code>toArray(T[] a)</code>



```
default Stream<E>           stream()  
default Stream<E>           parallelStream()  
default Spliterator<E>       spliterator()  
default <T> T[]             toArray(IntFunction<T[]> generator)
```

İlkel Tipler ve Collection



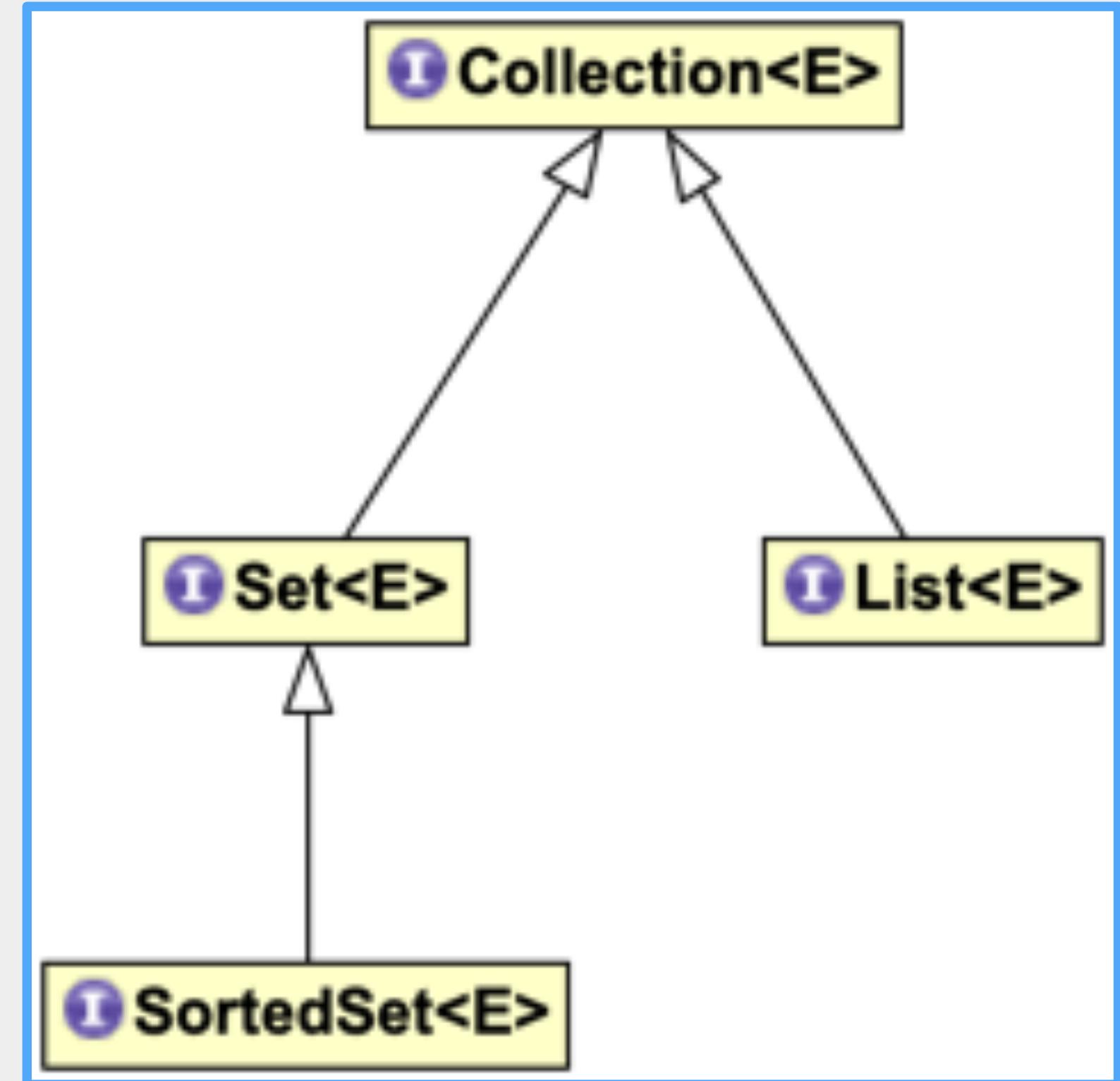
- **Collection** ilkel (primitive) tiplerle de çalışır.
- Çünkü ilkel tiplerle onların sarmalayan (wrapper) tipler arasındaki dönüşümler otomatik olarak yapılır.
- Dolayısıyla bir ilkel tip değişkeni ya da sabitesi **Collection** nesnesine eklenebilir ya da ondan geri alınabilir.

```
Collection<Integer> col = ...;
int i = 5;
col.add(i);
Iterator<Integer> it = col.iterator();
while(it.hasNext()){
    int j = it.next();
    ...
}
```

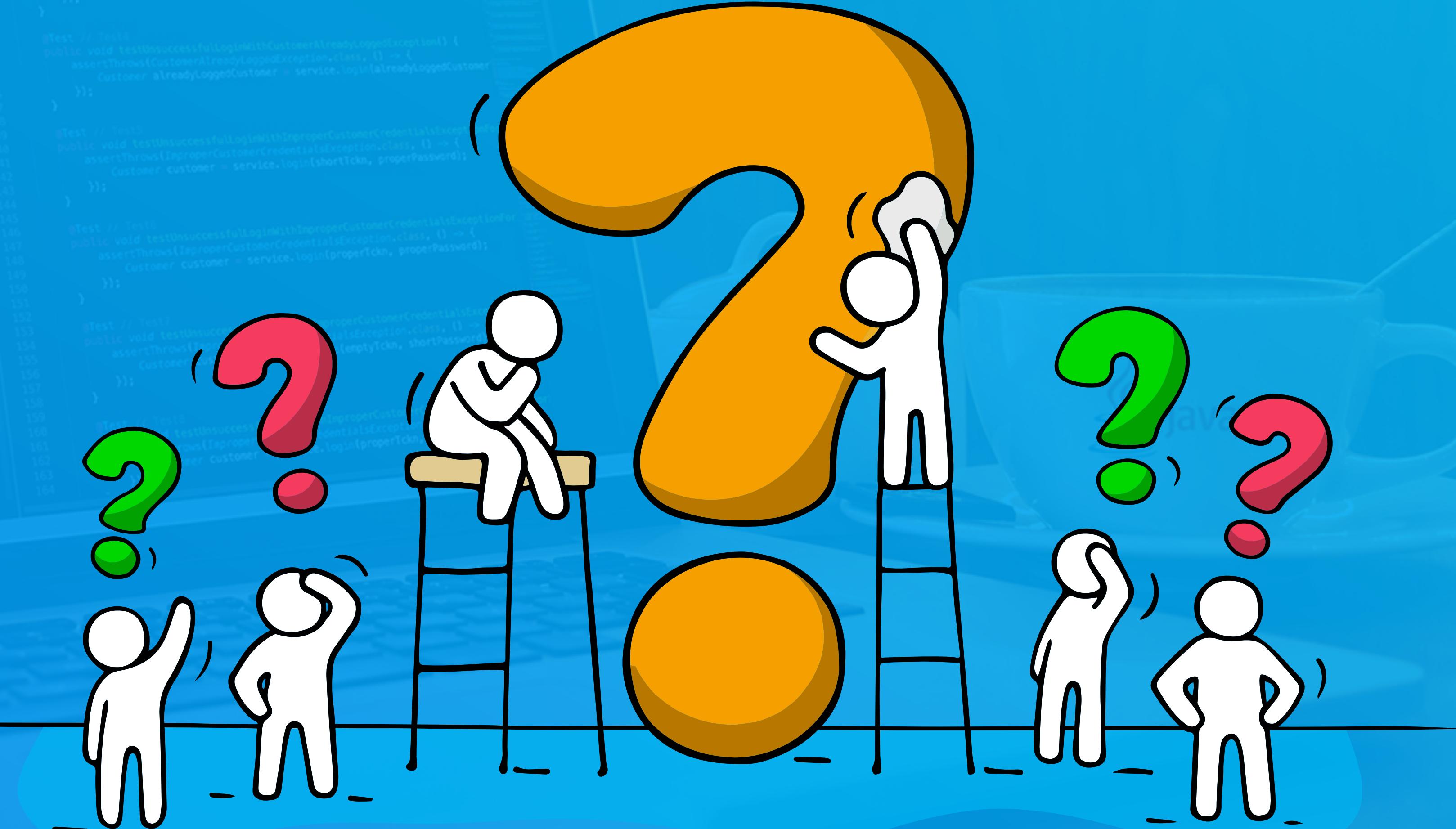
Collection'un Alt Tipleri



- Collection'un doğrudan bir gerçekleştirmesi yoktur ama iki önemli alt arayüzü vardır:
 - **Set:** Kümedir, tuttuğu nesneler tekildir, aynı nesneden birden fazla tutmaz.
 - Elemanları hash kod ile ilişkilendirerek saklar.
 - **List:** Doğrusal bir torbadır, dinamik, büyüyüp küçülebilen dizidir.
- Bu iki arayüzün gerçekleştirmeleri vardır.



Soru ve Cevap Zamani!





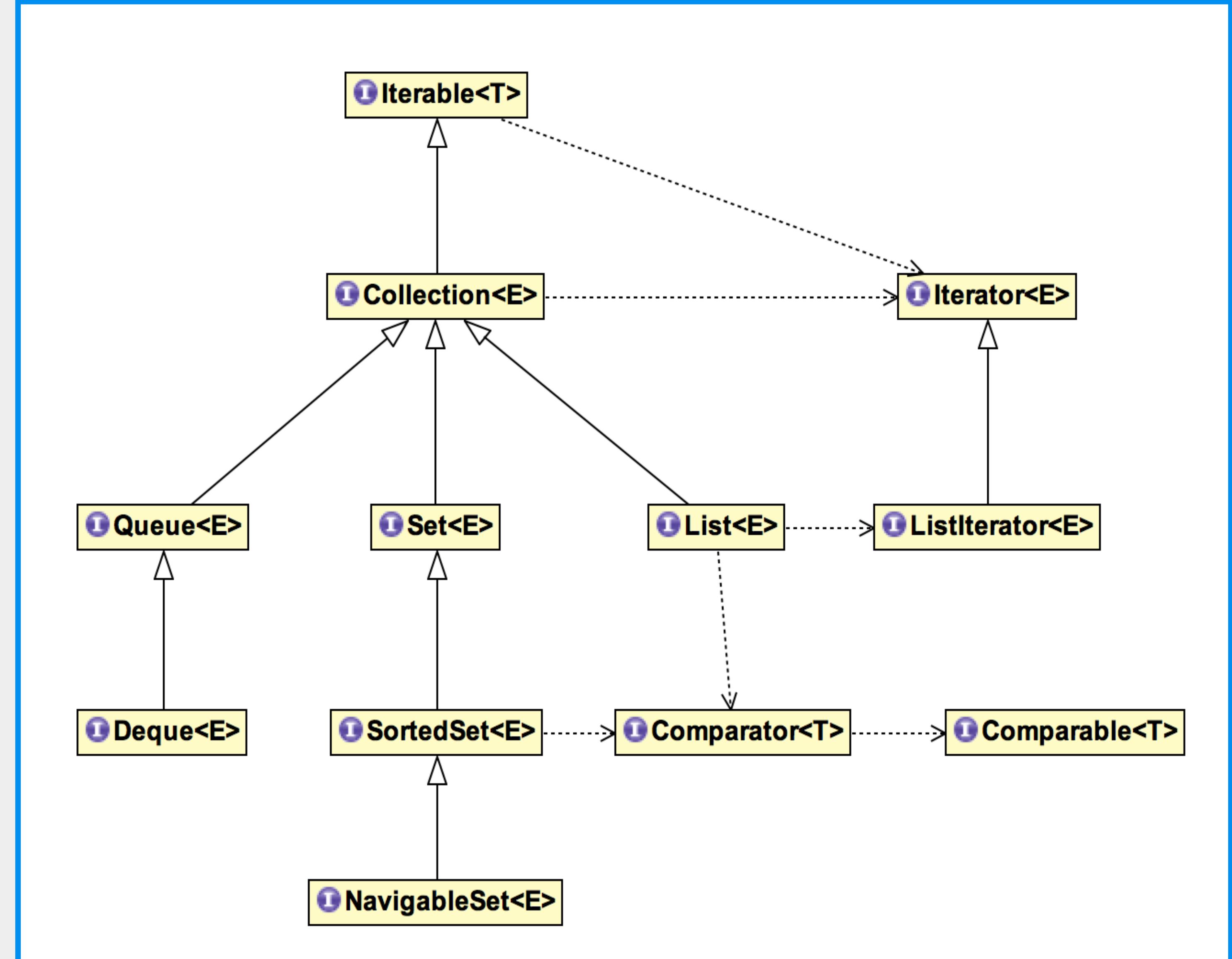
```
111 public void testSuccessfullLogin() {
112     assertThrows(NoSuchCustomerException.class, () -> {
113         Customer loggedCustomer = service.login(properties, properPassword);
114     });
115 }
116
117 @Test // Test 1
118 public void testSuccessfullLoginWithNoSuchCustomerException() {
119     assertThrows(NoSuchCustomerException.class, () -> {
120         Customer unfoundCustomer = service.login(unfoundCustomerToken, properPassword);
121     });
122 }
123
124 @Test // Test 2
125 public void testSuccessfullLoginWithCustomerLockedException() {
126     assertThrows(CustomerLockedException.class, () -> {
127         Customer lockedCustomer = service.login(lockedCustomerToken, "password");
128     });
129 }
130
131 @Test // Test 3
132 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {
133     assertThrows(CustomerAlreadyLoggedException.class, () -> {
134         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerToken, "password");
135     });
136 }
137
138 @Test // Test 4
139 public void testSuccessfullLoginWithInproperCustomerCredentialsException() {
140     assertThrows(InproperCustomerCredentialsException.class, () -> {
141         Customer customer = service.login(shortToken, properPassword);
142     });
143 }
144
145 @Test // Test 5
146 public void testSuccessfullLoginWithInproperCustomerCredentialsException() {
147     assertThrows(InproperCustomerCredentialsException.class, () -> {
148         Customer customer = service.login(propertiesToken, properPassword);
149     });
150 }
151
152 @Test // Test 6
153 public void testSuccessfullLoginWithInproperCustomerCredentialsException() {
154     assertThrows(InproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(propertiesToken, shortPassword);
156     });
157 }
```

Diger Bazi Arayuzler

Tüm Arayüzler



- Önceki sınıf diyagramını diğer doğrusal torbalarla ilgili arayüzleri kullanarak zenginleştirebiliriz.
- Sırayla, **Iterator**, **Iterable**, **Comparable** ve **Comparator**'ü ele alalım.





```
111 public void testSuccessfullLogin() {
112     throws NoSuchCustomerException, CustomerLockedException, CustomerInproperCustomerCredentialsException, MaxImposedRateExceededException;
113     Customer loggedCustomer = service.login(properties, properPassword);
114     assertEquals(successfulCustomer, loggedCustomer);
115 }
116
117 @Test // Test 1
118 public void testSuccessfullLoginWithNoSuchCustomerException() {
119     assertThrows(NoSuchCustomerException.class, () -> {
120         Customer unfoundCustomer = service.login(unfoundCustomerToken, properPassword);
121     });
122 }
123
124 @Test // Test 2
125 public void testSuccessfullLoginWithCustomerLockedException() {
126     assertThrows(CustomerLockedException.class, () -> {
127         Customer lockedCustomer = service.login(lockedCustomerToken, "password");
128     });
129 }
130
131 @Test // Test 3
132 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {
133     assertThrows(CustomerAlreadyLoggedException.class, () -> {
134         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerToken, properPassword);
135     });
136 }
137
138 @Test // Test 4
139 public void testSuccessfullLoginWithInproperCustomerCredentialsException() {
140     assertThrows(InproperCustomerCredentialsException.class, () -> {
141         Customer customer = service.login(shortToken, properPassword);
142     });
143 }
144
145 @Test // Test 5
146 public void testSuccessfullLoginWithInproperCustomerCredentialsException() {
147     assertThrows(InproperCustomerCredentialsException.class, () -> {
148         Customer customer = service.login(propertiesToken, properPassword);
149     });
150 }
151
152 @Test // Test 6
153 public void testSuccessfullLoginWithInproperCustomerCredentialsException() {
154     assertThrows(InproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(propertiesToken, shortPassword);
156     });
157 }
```

Iterator

Iterator - I



- Collection tipindeki torbalarda elemanlara ulaşmanın tek yöntemi `java.util.Iterator` arayüzüdür.
- Kelime anlamı, yineleyen, yineleyici demektir.
- Collection tipindeki torbalardan elemanlarını tek tek almakta kullanılır.
- Iterator arayüzü Java'ya 1.2 ile birlikte katılmıştır.

<code>boolean</code>	<code>hasNext()</code>
<code>E</code>	<code>next()</code>
<code>default void</code>	<code>remove()</code>
<code>default void</code>	<code>forEachRemaining(Consumer<? super E> action)</code>

Iterator - II



- İki soyut metodu vardır:

boolean	hasNext()
E	next()

- `Collection<E>`'dan `iterator()` metoduyla alınan `Iterator` nesnesinde `hasNext()` metodu `true` döndüğü müddetçe `next()` ile bir sonraki eleman alınabilir.

```
Collection<Integer> col = ...;  
...  
Iterator<Integer> it = col.iterator();  
while(it.hasNext()){  
    int j = it.next();  
    ...  
}
```



- Eğer iterator sona gelmiş dolayısıyla alttaki `Collection<E>`'da verecek hiç eleman kalmamış yani `hasNext()` çağrıSİ `false` döndürüyorsa, `next()` çağrıSİ `java.util.NoSuchElementException` fırlatır.
- Dolayısıyla `next()` çağrıSİ daima `hasNext()` çağrısınınONdan sonra, `true` dönerse yapılmalıdır.

Iterator - IV



- İki **default** metodu vardır:
 - **remove()** işlediği **Collection**'dan o sırada, **next()** çağrısının döndürdüğü elemanı siler.
 - **forEachRemaining()** işlediği **Collection**'daki elemanlara verilen **Consumer**'ı uygular.
 - **Collection**'daki **forEach()** 'ten farkı yoktur.

```
default void remove()
```

```
default void forEachRemaining(Consumer<? super E> action)
```

Iterator - V



- **Iterator**, elemanlarını işlediği **Collection**'dan **remove ()** ile silme de yapar.
- **remove ()** her bir **next ()** çağrısı için sadece bir defa çağrılmalıdır.
- **Collection** nesnesi, kendisinden **Iterator** alındıktan sonra üzerindeki **add ()** ya da **remove ()** ile değiştirilirse, iteratorun ilk **next ()** çağrısında **java.util.ConcurrentModificationException** sıra dışı durumu fırlatılır.
- Buna **fail-fast** denir, yani iterator durumu algılar algılamaz hata verir.



- Bu yüzden **Iterator** oluşturulduktan sonra **Collection**'dan silme **Iterator** üzerindeki **remove ()** metodu ile yapılmalıdır.
- Ekleme ise **Collection**'dan henüz **Iterator** alınmadan önce tamamlanmalıdır.

IteratorExample.java



- org.javaturk.oofp.ch10.interfaces.IteratorExample



```
111 public void testSuccessfullLogin() {
112     // Given
113     // When
114     // Then
115     // ...
116 }
117
118 @Test // Test 1
119 public void testSuccessfullLoginWithNoSuchCustomerException() {
120     // Given
121     // When
122     // Then
123     // ...
124 }
125
126 @Test // Test 2
127 public void testSuccessfullLoginWithCustomerLockedException() {
128     // Given
129     // When
130     // Then
131 }
132
133 @Test // Test 3
134 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {
135     // Given
136     // When
137     // Then
138 }
139
140 @Test // Test 4
141 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
142     // Given
143     // When
144     // Then
145 }
146
147 @Test // Test 5
148 public void testSuccessfullLoginWithImproperCustomerCredentialsSpecificationException() {
149     // Given
150     // When
151     // Then
152 }
153
154 @Test // Test 6
155 public void testSuccessfullLoginWithImproperCustomerCredentialsSpecificationException() {
156     // Given
157     // When
158     // Then
159 }
160
161 @Test // Test 7
162 public void testSuccessfullLoginWithImproperCustomerCredentialsSpecificationException() {
163     // Given
164     // When
165     // Then
166 }
```

Iterable

Iterable - I



- `java.lang.Iterable` arayüzüünü gerçekleştiren torbalar, hem `Iterator` sağlar hem de `Consumer` alan `forEach()` ile elemanlarının işlenebilmesine imkan verir.
- `Collection`’ın üst arayüzüdür dolayısıyla tüm `Collection` nesneleri `Iterable`’dır.
- Ama diziler (arrays) `Iterable` değildir. Neden?

`default void`

`forEach(Consumer<? super T> action)`

`Iterator<T>`

`iterator()`

`default Spliterator<T>`

`spliterator()`

Iterable - II



- Iterable arayüzü, `forEach()` dışında iterator veren iki metoda daha sahiptir.
- Ama `forEach` kullanım açısından `Iterator` kullanımından çok daha sık ve rahattır.

```
default void           forEach(Consumer<? super T> action)  
Iterator<T>          iterator()  
default Spliterator<T> spliterator()
```

IterableExample.java



- org.javaturk.oofp.ch10.interfaces.IterableExample

Nesne Siralama



```
115     assertEquals(successfulCustomer, loggedCustomer);
116 }
117
118 @Test // Test2
119 public void testUnsuccessfulLoginWithNoSuchCustomerException() {
120     assertThrows(NoSuchCustomerFoundException.class, () -> {
121         Customer unfoundCustomer = service.login(unfoundCustomerTckn, properPassword);
122     });
123 }
124
125 @Test // Test3
126 public void testUnsuccessfulLoginWithCustomerLockedException() {
127     assertThrows(CustomerLockedException.class, () -> {
128         Customer lockedCustomer = service.login(lockedCustomerTckn, "qwerty123");
129     });
130 }
131
132 @Test // Test4
133 public void testUnsuccessfulLoginWithCustomerAlreadyLoggedException() {
134     assertThrows(CustomerAlreadyLoggedException.class, () -> {
135         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTckn, properPassword);
136     });
137 }
138
139 @Test // Test5
140 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForProperCustomer() {
141     assertThrows(ImproperCustomerCredentialsException.class, () -> {
142         Customer customer = service.login(shortTckn, properPassword);
143     });
144 }
145
146 @Test // Test6
147 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForImproperCustomer() {
148     assertThrows(ImproperCustomerCredentialsException.class, () -> {
149         Customer customer = service.login(properTckn, properPassword);
150     });
151 }
152
153 @Test // Test7
154 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForImproperCustomerAndPassword() {
155     assertThrows(ImproperCustomerCredentialsException.class, () -> {
156         Customer customer = service.login(properTckn, shortPassword);
157     });
158 }
159
160 @Test // Test8
161 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForProperCustomerAndShortPassword() {
162     assertThrows(ImproperCustomerCredentialsException.class, () -> {
163         Customer customer = service.login(propertTckn, shortPassword);
164     });
165 }
```

Nesnelerin Sıralanması



- Java'da nesnelerin sıralanmasıyla ilgili iki arayüz vardır:
 - `java.lang.Comparable`
 - `java.util.Comparator`
- **Comparable** arayüzü onu gerçekleştiren nesnelerin sıralanabilmesini sağlar.
- **Comparable** arayüzü ile sağlanan sıralanmaya **tabi sıralanma (natural ordering)** denir.



```
111 public void testSuccessfullLogin() {  
112     assertEquals(CustomerNotFoundException.class, customer.  
113         login("customer", "password").getClass());  
114     assertEquals("Customer logged in successfully.", customer.  
115         login("customer", "password").getLogInMessage());  
116 }  
117  
118 @Test // Test 1  
119 public void testSuccessfullLoginWithUnfoundCustomer() {  
120     assertEquals(UnfoundCustomerException.class, () -> {  
121         Customer unfoundCustomer = service.login("unfoundCustomer", "proper  
122             password");  
123     }).getClass();  
124 }  
125  
126 @Test // Test 2  
127 public void testSuccessfullLoginWithLockedCustomer() {  
128     assertEquals(LockedCustomerException.class, () -> {  
129         Customer lockedCustomer = service.login("lockedCustomer", "proper  
130             password");  
131     }).getClass();  
132 }  
133  
134 @Test // Test 3  
135 public void testSuccessfullLoginWithAlreadyLoggedCustomer() {  
136     assertEquals(AlreadyLoggedCustomerException.class, () -> {  
137         Customer alreadyLoggedCustomer = service.login("alreadyLoggedCustomer",  
138             "proper password");  
139     }).getClass();  
140 }  
141  
142 @Test // Test 4  
143 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
144     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
145         Customer customer = service.login("shortTckn", "properPassword");  
146     }).getClass();  
147 }  
148  
149 @Test // Test 5  
150 public void testSuccessfullLoginWithImproperCustomerCredentialsAndPassword() {  
151     assertEquals(ImproperCustomerCredentialsAndPasswordException.class, () -> {  
152         Customer customer = service.login("properTckn", "properPassword");  
153     }).getClass();  
154 }  
155  
156 @Test // Test 6  
157 public void testSuccessfullLoginWithImproperCustomerCredentialsAndTckn() {  
158     assertEquals(ImproperCustomerCredentialsAndTcknException.class, () -> {  
159         Customer customer = service.login("properTckn", "shortPassword");  
160     }).getClass();  
161 }
```

Comparable

Comparable - I



- `java.lang.Comparable` arayüzü, onu gerçekleştiren nesnelerin sıralanabilmesini sağlar.
- Java'ya 1.2 ile birlikte katılmıştır ve fonksiyonel bir arayüzdür..
- Üzerindeki tek soyut metot:

`int compareTo(T o)`

- Eğer dönen değer negatif ise üzerinde `compareTo()` metodу çağrılan nesne önce gelmelidir, pozitif ise geçilen nesne önce gelmelidir, sıfır ise bu iki nesne de aynı sıradadır.

Comparable Arayüzü - II



- Java API'sindeki şu sınıflar **Comparable** arayüzü gerçekleştirirler, dolayısıyla da **tabi sıralamaya (natural ordering)** sahiptirler:
 - Tüm sarmalayan tipler (**Integer**, **Long**, **Double**, **Boolean** vs.)
 - Örneğin **Boolean.FALSE < Boolean.TRUE**
 - **BigInteger**, **BigDecimal**,
 - **String**, **Date**, **File**.

ComparableExample.java



- org.javaturk.oofp.ch10.interfaces.ComparableExample



```
111 public void testSuccessfullLogin() {  
112     assertThrows(NoSuchCustomerFoundException.class, () -> {  
113         Customer loggedCustomer = service.login(properties, properPassword);  
114     });  
115 }  
116  
117 @Test // Test 1  
118 public void testSuccessfullLoginWithNoSuchCustomerException() {  
119     assertThrows(NoSuchCustomerFoundException.class, () -> {  
120         Customer unfoundCustomer = service.login(unfoundCustomerTkn, proper  
121             Password);  
122     });  
123 }  
124  
125 @Test // Test 2  
126 public void testSuccessfullLoginWithCustomerLockedException() {  
127     assertThrows(CustomerLockedException.class, () -> {  
128         Customer lockedCustomer = service.login(lockedCustomerTkn, "short  
129             Password");  
130     });  
131 }  
132  
133 @Test // Test 3  
134 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {  
135     assertThrows(CustomerAlreadyLoggedException.class, () -> {  
136         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomer  
137             Tkn, "shortPassword");  
138     });  
139 }  
140  
141 @Test // Test 4  
142 public void testSuccessfullLoginWithImproperCustomerCredentialsException  
143 () {  
144     assertThrows(ImproperCustomerCredentialsException.class, () -> {  
145         Customer customer = service.login(shortTkn, properPassword);  
146     });  
147 }  
148  
149 @Test // Test 5  
150 public void testSuccessfullLoginWithImproperCustomerCredentialsException  
151 () {  
152     assertThrows(ImproperCustomerCredentialsException.class, () -> {  
153         Customer customer = service.login(propertiesTkn, properPassword);  
154     });  
155 }  
156  
157 @Test // Test 6  
158 public void testSuccessfullLoginWithImproperCustomerCredentialsException  
159 () {  
160     assertThrows(ImproperCustomerCredentialsException.class, () -> {  
161         Customer customer = service.login(properTkn, shortPassword);  
162     });  
163 }
```

Comparator

Comparator - I



- Eğer sıralanacak nesneler **Comparable** arayüzüünü gerçekleştirmiyorsa ne yapılabilir?
 - Ya da birden çok kıyaslama söz konusuysa?
- Bu durumlarda **java.util.Comparator** arayüzü, nesnelerin sıralamasında kullanılabilir.
- Foknsiyonel bir arayüzdür dolayısıyla üzerindeki, iki tane, kıyaslanacak nesneleri argüman olarak alan tek bir soyut metot vardır.

```
int compare(T o1, T o2)
```

Comparator - II



- Comparator arayüzü Java'ya 1.2 ile birlikte katılmıştır.
- Java SE 1.8'e kadar tek bir metodu vardı.
- Java SE 1.8'de pek çok **default** ve **static** metot eklenmiştir ve şu anda 18 metodu vardır.

Comparator<T>

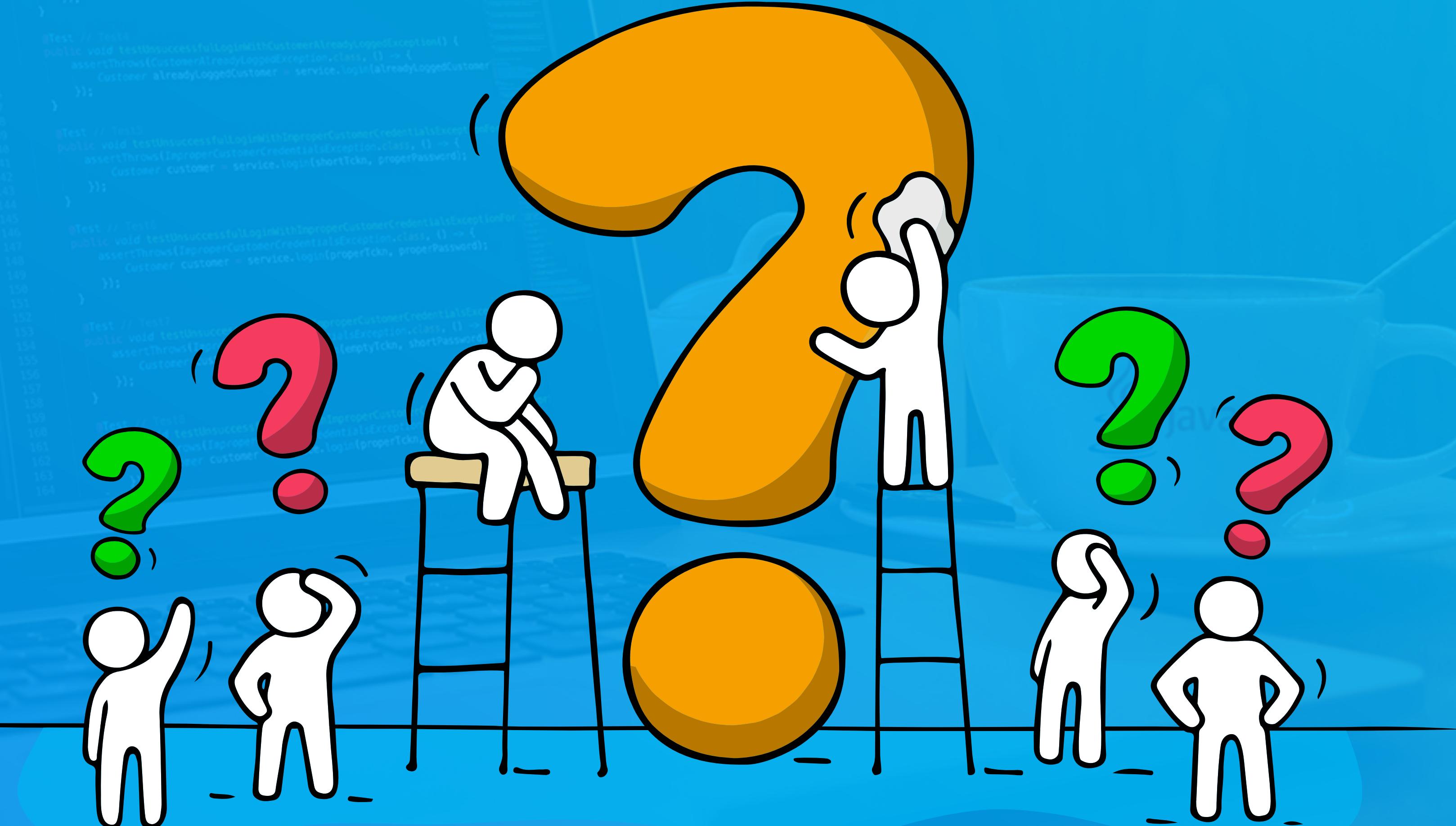
- `s comparing(Function<? super T,? extends U>):Comparator<T>`
- `s comparing(Function<? super T,? extends U>,Comparator<? super U>):Comparator<T>`
- `s comparingDouble(ToDoubleFunction<? super T>):Comparator<T>`
- `s comparingInt(ToIntFunction<? super T>):Comparator<T>`
- `s comparingLong(ToLongFunction<? super T>):Comparator<T>`
- `s naturalOrder():Comparator<T>`
- `s nullsFirst(Comparator<? super T>):Comparator<T>`
- `s nullsLast(Comparator<? super T>):Comparator<T>`
- `s reverseOrder():Comparator<T>`
- `compare(T,T):int`
- `equals(Object):boolean`
- `reversed():Comparator<T>`
- `thenComparing(Comparator<? super T>):Comparator<T>`
- `thenComparing(Function<? super T,? extends U>):Comparator<T>`
- `thenComparing(Function<? super T,? extends U>,Comparator<? super U>):Comparator<T>`
- `thenComparingDouble(ToDoubleFunction<? super T>):Comparator<T>`
- `thenComparingInt(ToIntFunction<? super T>):Comparator<T>`
- `thenComparingLong(ToLongFunction<? super T>):Comparator<T>`

ComparatorExample.java



- org.javaturk.oofp.ch10.interfaces.ComparatorExample
- org.javaturk.oofp.ch09.functions.composition.ComparatorComposition

Soru ve Cevap Zamani!



Collection Torbalari

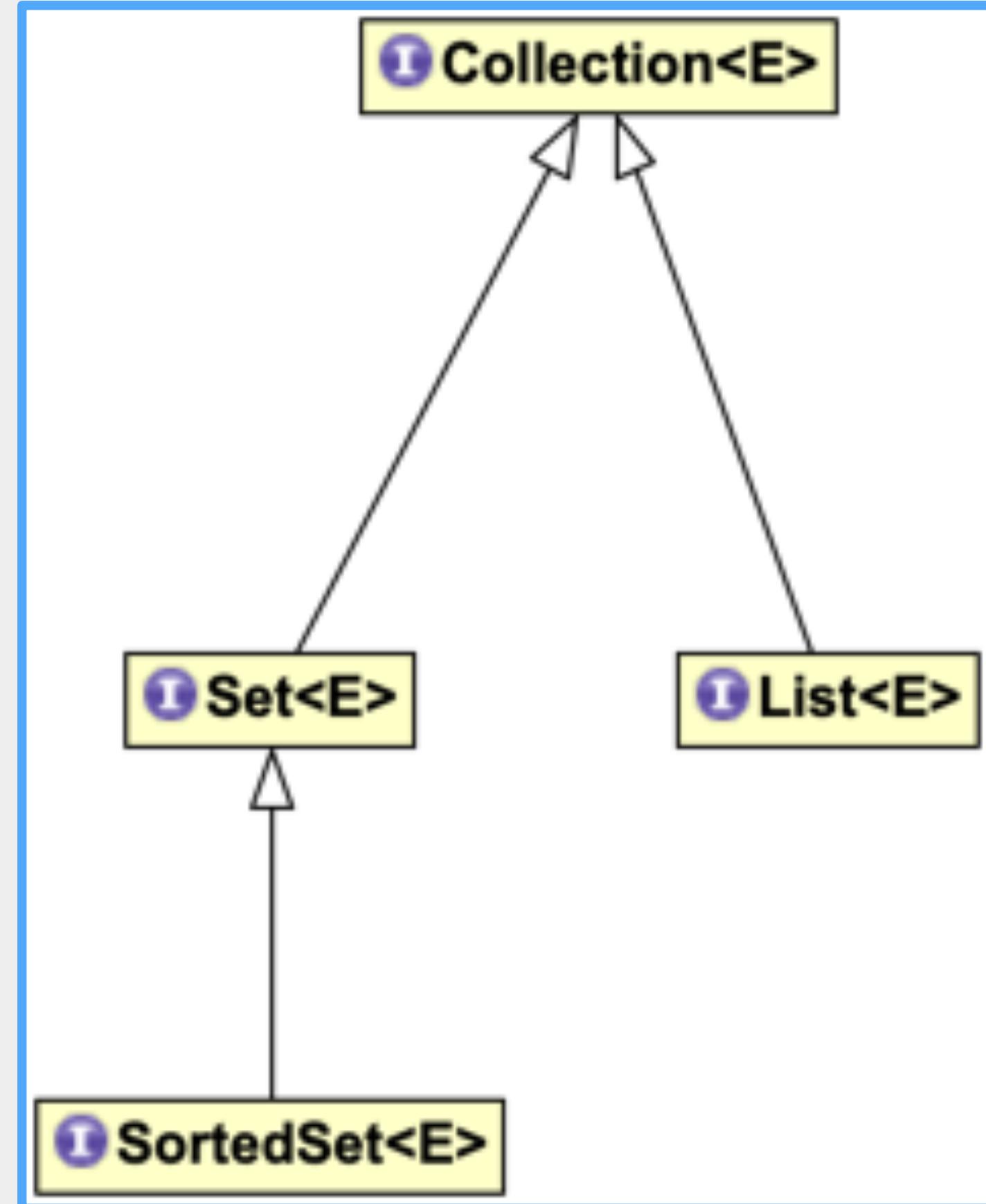


```
115     assertEquals(successfulCustomer, loggedCustomer);
116 }
117
118 @Test // Test2
119 public void testUnsuccessfulLoginWithNoSuchCustomerException() {
120     assertThrows(NoSuchCustomerFoundException.class, () -> {
121         Customer unfoundCustomer = service.login(unfoundCustomerTckn, proper
122     });
123 }
124
125 @Test // Test3
126 public void testUnsuccessfulLoginWithCustomerLockedException() {
127     assertThrows(CustomerLockedException.class, () -> {
128         Customer lockedCustomer = service.login_lockedCustomerTckn, "qwerty1
129     });
130 }
131
132 @Test // Test4
133 public void testUnsuccessfulLoginWithCustomerAlreadyLoggedException() {
134     assertThrows(CustomerAlreadyLoggedException.class, () -> {
135         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomer
136     });
137 }
138
139 @Test // Test5
140 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionFor
141     assertThrows(ImproperCustomerCredentialsException.class, () -> {
142         Customer customer = service.login(shortTckn, properPassword);
143     });
144 }
145
146 @Test // Test6
147 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionFor
148     assertThrows(ImproperCustomerCredentialsException.class, () -> {
149         Customer customer = service.login(properTckn, properPassword);
150     });
151 }
152
153 @Test // Test7
154 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionFor
155     assertThrows(ImproperCustomerCredentialsException.class, () -> {
156         Customer customer = service.login(properTckn, shortPassword);
157     });
158 }
159
160 @Test // Test8
161 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionFor
162     assertThrows(ImproperCustomerCredentialsException.class, () -> {
163         Customer customer = service.login(propertckn, shortPassword);
164     });
165 }
```

Collection'un Alt Tipleri



- Collection'un gerçekleştirmesi olmadığı ama iki önemli alt arayüzünün olduğu, bu arayüzlerin gerçekleştirmelerinin olduğu daha önce söylemişmiş:
 - **Set**: Kümedir, tuttuğu nesneler tekildir, aynı nesneden birden fazla tutmaz.
 - **List**: Doğrusal bir torbadır, dinamik, büyüyüp küçülebilen dizidir.
 - Şimdi bunları arayüzleri ve gerçekleştirmelerini görelim.



Set ve Gerekleştirmeleri



```
115     assertEquals(successfulCustomer, loggedCustomer);
116 }
117
118 @Test // Test2
119 public void testUnsuccessfulLoginWithNoSuchCustomerException() {
120     assertThrows(NoSuchCustomerFoundException.class, () -> {
121         Customer unfoundCustomer = service.login(unfoundCustomerTckn, properPassword);
122     });
123 }
124
125 @Test // Test3
126 public void testUnsuccessfulLoginWithCustomerLockedException() {
127     assertThrows(CustomerLockedException.class, () -> {
128         Customer lockedCustomer = service.login_lockedCustomerTckn, "qwerty123");
129     });
130 }
131
132 @Test // Test4
133 public void testUnsuccessfulLoginWithCustomerAlreadyLoggedException() {
134     assertThrows(CustomerAlreadyLoggedException.class, () -> {
135         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTckn, properPassword);
136     });
137 }
138
139 @Test // Test5
140 public void testUnsuccessfulLoginWithImproperCustomerCredentialsException() {
141     assertThrows(ImproperCustomerCredentialsException.class, () -> {
142         Customer customer = service.login(shortTckn, properPassword);
143     });
144 }
145
146 @Test // Test6
147 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForEmptyTckn() {
148     assertThrows(ImproperCustomerCredentialsException.class, () -> {
149         Customer customer = service.login(emptyTckn, properPassword);
150     });
151 }
152
153 @Test // Test7
154 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForEmptyPassword() {
155     assertThrows(ImproperCustomerCredentialsException.class, () -> {
156         Customer customer = service.login(emptyTckn, shortPassword);
157     });
158 }
```



- **Set** arayüzü, matematikteki kümeyi temsil eder.
- En temel iki özelliği:
 - Elemanları tekil olmalıdır, aynı elemandan birden fazla tutmaz.
 - Dizilim-sıra (order) kavramı yoktur.
 - **Set**, tüm **Collection** nesneleri gibi **Iterable**'dır ve devraldığı **iterator()** metoduna sahiptir.
 - Dolayısıyla elemanlarına ancak **Iterator** ile ulaşılabilir.

Set - II



- **Set**, **Collection**'dan devraldığı soyut davranışlara bir ekleme yapmaz, sadece kendine özgü statik metotlar ekler.
- **Collection**'dan en bariz farkı, ekleme (**add()** ve **addAll()**) metotlarına bir kısıt koymasıdır,
 - Tekil nesneler tutmak için **Set** her eleman ya da torba eklenmesinde, yani **add()** ve **addAll()** metodlarında aynılık kontrolü yapar.
 - Dolayısıyla bu metodlar **false** döndürebilir.

Set - III



- Eğer bir **Set**'e bir **Collection** nesnesi **add()** metodu ile eklenirse, **Set**, **Collection** nesnesini ayrı bir eleman olarak görür ve **Collection** nesnesi üzerinden aynılık kontrolü yapar.
- Aynı türden ekleme **add()** yerine **addAll()** metoduyla yapılrsa bu durumda **Collection** nesnesinin elemanları ayrı ayrı, aynılık kontrolüne tabi tutulur ve geçenler **Set**'e eklenir.

Set - IV



- Torbanın içindeki elemanların daima tekil olmaları sebebiyle **Set** gerçekleştirmeleri daha çok
 - bir kişinin kredi kartları, evleri, araçları ya da
 - bir ailenin fertleri
- gibi tekilik gerektiren durumlarda kullanılır.
- **Set** en fazla bir tane **null** eleman tutabilir.

FindDuplicates.java



- `org.javaturk.oofp.ch10.set.FindDuplicates`
- **Set'in add()** metodunun davranışına dikkat edin.
- **Set'ten elemanların iterator yoluyla alınmasına** dikkat edin.

Set - V



- Set'e Java'nın 9. sürümünde 12 tane `of` isimli, overloaded `static` üretici (factory) metot eklenmiştir.

```
static <E> Set<E> of()  
  
static <E> Set<E> of(E ... elements)  
  
static <E> Set<E> of(E e1)  
  
static <E> Set<E> of(E e1, E e2)  
...
```

- Ayrıca Set'e Java'nın 10. sürümünde, geçen bir `Collection`'daki nesnelerle Set oluşturan `static` kopyalama metodu da eklenmiştir.

```
<E> Set<E> copyOf(Collection<? extends E> coll)
```



- Bu metodlar değiştirilemez (unmodifiable) Set nesnesi oluşturur.
- Oluşturulan Set değiştirilmeye çalışıldığında `java.lang.UnsupportedOperationException` fırlatır.
- Ayrıca bu şekilde oluşturulan Set, null elemanlara izin vermez, null ile oluşturulmaya çalışıldığında `NullPointerException` fırlatır.

SetExample.java

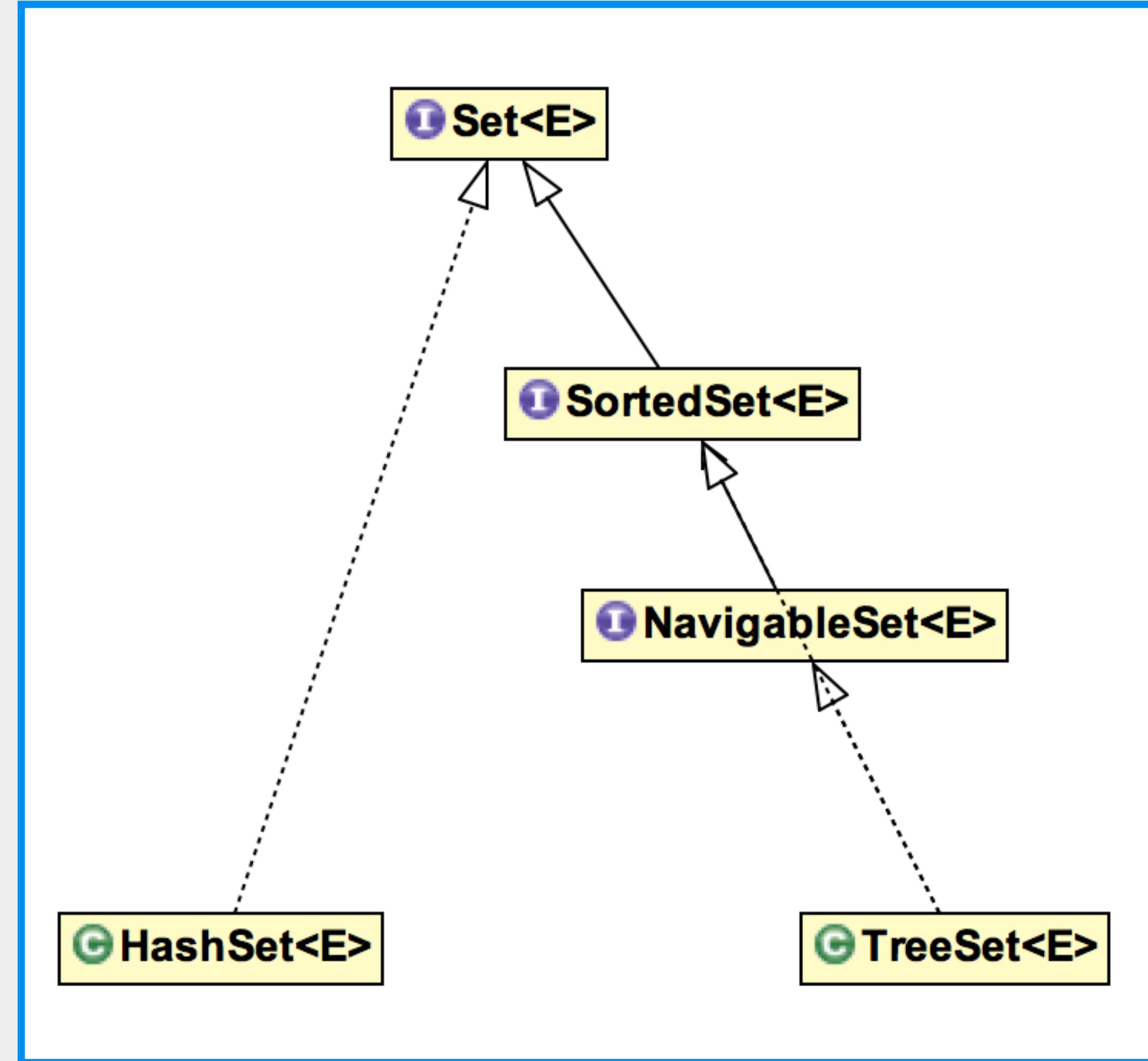


- org.javaturk.oofp.ch10.set.SetExample

Set - VII



- Set arayüzünün en temel iki gerçeklestirmesi vardır:
 - HashSet, Set'i doğrudan gerçekleştirir ve en sık kullanılır.
 - TreeSet ise Set'in alt arayüzleri olan SortedSet ve NavigableSet arayüzlerini de gerçekleştirir ve genelde sadece sıralama gereğinde kullanılır.





```
111 public void testSuccessfullLogin() {  
112     assertEquals(CustomerNotFoundException.class, customer.  
113         login("customer", "password").getClass());  
114     assertEquals("Customer logged in", service.loggedCustomer);  
115     assertEquals(true, successfulCustomer.loggedCustomer);  
116 }  
117  
118 @Test // Test 1  
119 public void testSuccessfullLoginWithUnfoundCustomer() {  
120     assertEquals(UnfoundCustomerException.class, () -> {  
121         Customer.unfoundCustomer = service.login(unfoundCustomerTkn, proper  
122             .password());  
123     }).getClass();  
124 }  
125  
126 @Test // Test 2  
127 public void testSuccessfullLoginWithCustomerLocked() {  
128     assertEquals(CustomerLockedException.class, () -> {  
129         Customer.lockedCustomer = service.login(lockedCustomerTkn, "password");  
130     }).getClass();  
131 }  
132  
133 @Test // Test 3  
134 public void testSuccessfullLoginWithCustomerAlreadyLogged() {  
135     assertEquals(CustomerAlreadyLoggedException.class, () -> {  
136         Customer.alreadyLoggedCustomer = service.login(alreadyLoggedCustomer  
137             .Tkn, properPassword());  
138     }).getClass();  
139 }  
140  
141 @Test // Test 4  
142 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
143     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
144         Customer.customer = service.login(shortTkn, properPassword());  
145     }).getClass();  
146 }  
147  
148 @Test // Test 5  
149 public void testSuccessfullLoginWithImproperCustomerCredentialsAndPassword() {  
150     assertEquals(ImproperCustomerCredentialsAndPasswordException.class, () -> {  
151         Customer.customer = service.login(properTkn, shortPassword());  
152     }).getClass();  
153 }  
154  
155 @Test // Test 6  
156 public void testSuccessfullLoginWithImproperCustomerCredentialsAndPasswordAndTkn() {  
157     assertEquals(ImproperCustomerCredentialsAndPasswordAndTknException.class, () -> {  
158         Customer.customer = service.login(properTkn, properPassword());  
159     }).getClass();  
160 }
```

HashSet

HashSet - I



- **HashSet**, arka tarafta bir **HashMap** tarafından desteklenen bir **Set** gerçekleştirmesidir.
- **HashSet**, herhangi bir dizilim sözü vermez, var olan dizilim zaman içinde değişimdir.
- **HashSet**, sadece bir tane **null** referansa izin verir.
- **HashSet**, temel fonksiyonlarda (**add()**, **remove()**, **contains()**, ve **size()**) sabite yakın, **O(1)**, bir performans sağlar.

HashSet - II



- **HashSet**, bir **Set** olduğundan elemanlarına ancak **Iterator** ile erişilebilir.
- Bu da eleman sayısıyla orantılı, linear bir performans verir.

HashSet - III



- HashSet'in 4 tane kurucusu vardır.
 - Varsayılan kurucu başlangıç kapasitesi 16 ve doluluk oranı 0,75 olan boş bir HashSet oluşturur.
 - Arzu edilirse HashSet oluşturulurken başlangıç kapasitesi ve doluluk oranı verilebilir,

`HashSet()`

`HashSet(Collection<? extends E> c)`

`HashSet(int initialCapacity)`

`HashSet(int initialCapacity, float loadFactor)`

HashSet - IV



- Ya da bir başka **Collection** nesnesinden **HashSet** oluşturulabilir,
- Bu durumda kaynak torbadaki elemanlar **HashSet**'e aktarılırken yine aynılık kontrolünden geçirilir.

HashSet()

HashSet(Collection<? extends E> c)

HashSet(int initialCapacity)

HashSet(int initialCapacity, float loadFactor)

FindDuplicates.java



- `org.javaturk.oofp.ch10.set.FindDuplicates`
- **HashSet**'ten içindeki elemanları iterator yoluyla alınmasına dikkat edin.
 - Elemanlar hangi sırayla geliyorlar?
 - Eleman eklenirken dizilim değişiyor mu?

HashSet - V



- **Set** arayüzü için “elemanları tekil olmalıdır, dolayısıyla aynı elemandan birden fazla tutmaz.” dendi.
- **HashSet**, iki elemanın aynı olup olmadığını nasıl anlar?

SetWithDuplicatedObjects.java



- org.javaturk.oofp.ch10.set.SetWithDuplicatedObjects
- Bu örneği HashSet ile çalıştırın.
- Örneği Employee sınıfının aşağıda belirtilen farklı şekilleriyle çalıştırın:
 - equals() ve hashCode() override edilmemiş,
 - Sadece equals() override edilmiş,
 - Sadece hashCode() override edilmiş,
 - Hem equals() ve hashCode() override edilmiş.

SetWithDuplicatedObjects.java



- Hangi durumda **HashSet**'in birbirinin aynı olan nesnelerden sadece bir tane tuttuğunu gözlemlediniz?

HashSet - VI



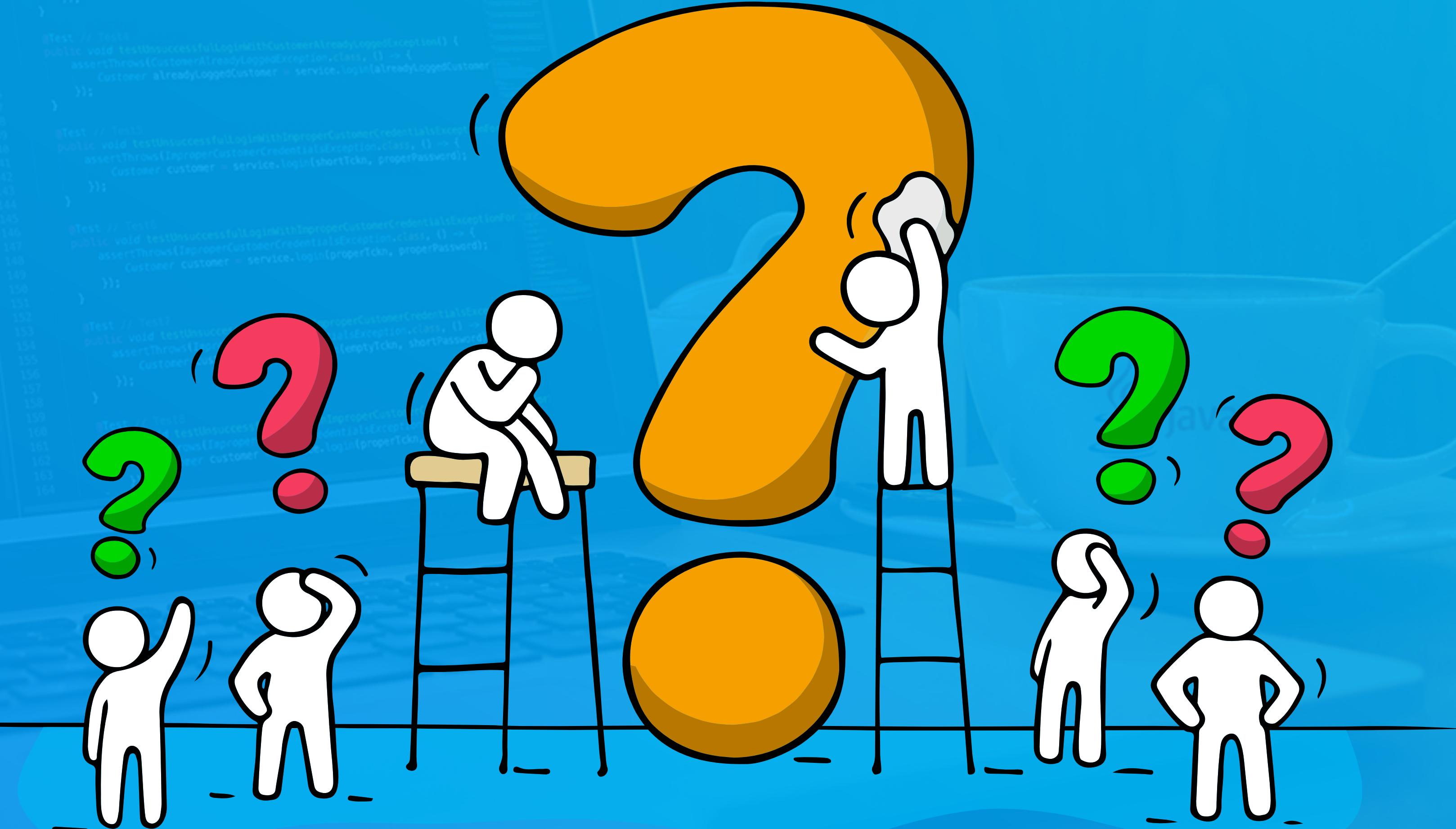
- HashSet, iki elemanın aynı olup olmadığını belirlemek için eklenen nesnelerin **equals ()** ve **hashCode ()** metodlarını çağrıır ve ilkinden **true** ikincisinden aynı **int** dönerse iki nesnenin aynı olduğuna karar verir.
- İki nesne üzerinde yapılan çağrıların sırası şöyledir:
 - Önce **hashCode ()** çağrıır, dönen sonuçlar aynı ise **equals ()** çağrıır,
 - İki **equals ()** da **true** döndürürse nesnenin zaten torbada olduğuna karar verilir ve **add ()** metodu ekleme yapmadan **false** döndürür.
 - Bu iki çağrıdan birisi için aksi durum olursa eleman eklenir ve **add ()** metodu **true** döndürür.

HashSet - VII



- Java API'sindeki aşağıdaki tiplerin **equals()** ve **hashCode()** metotları olması gereği gibi override edildiğinden bu tiplerin nesneleri tabi olarak aynılık-farklılık kontrolünden geçer:
- Tüm sarmalayan tipler (**Integer**, **Long**, **Double**, **Boolean** vs.)
 - Örneğin **Boolean.FALSE < Boolean.TRUE**
 - **BigInteger**, **BigDecimal**,
 - **String**, **Date**, **File**.

Soru ve Cevap Zamani!





```
111 public void testSuccessfulLogin() {
112     assertThrows(NoSuchCustomerException.class, () -> {
113         Customer loggedCustomer = service.login(properties, properPassword);
114     });
115 }
116
117 @Test // Test 1
118 public void testSuccessfulLoginWithNoSuchCustomerException() {
119     assertThrows(NoSuchCustomerException.class, () -> {
120         Customer unfoundCustomer = service.login(unfoundCustomerToken, properPassword);
121     });
122 }
123
124 @Test // Test 2
125 public void testSuccessfulLoginWithCustomerLockedException() {
126     assertThrows(CustomerLockedException.class, () -> {
127         Customer lockedCustomer = service.login(lockedCustomerToken, "password");
128     });
129 }
130
131 @Test // Test 3
132 public void testSuccessfulLoginWithCustomerAlreadyLoggedException() {
133     assertThrows(CustomerAlreadyLoggedException.class, () -> {
134         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerToken, properPassword);
135     });
136 }
137
138 @Test // Test 4
139 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {
140     assertThrows(ImproperCustomerCredentialsException.class, () -> {
141         Customer customer = service.login(shortToken, properPassword);
142     });
143 }
144
145 @Test // Test 5
146 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {
147     assertThrows(ImproperCustomerCredentialsException.class, () -> {
148         Customer customer = service.login(propertiesToken, properPassword);
149     });
150 }
151
152 @Test // Test 6
153 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(propertiesToken, shortPassword);
156     });
157 }
```

Hashing ve Hash Fonksiyonları

Hashing ve Hash Fonksiyonları - I



- **HashSet** arka planda bir **HashMap** tarafından desteklenir.
- Bu yapılarda adı geçen “hash” nedir?
- **Hash**, girdilerine karşılık bir tam sayı üreten fonksiyonlara verilen genel bir isimdir.
- Hash fonksiyonlarının aynı girdi için aynı değeri, her bir farklı girdi için de farklı bir değer üretmesi beklenir.
- Böyle hash fonksiyonlarına **evrensel hash fonksiyonları (universal hash functions)** denir.

Hashing ve Hash Fonksiyonları - II



- Hash fonksiyonları tipik olarak, `HashSet` gibi, elemanlarının düzenlemesi görünüşte gelişmiş güzel olan yapıların其实 onları yapısal olarak düzenlenmesinde kullanılır.

Hashing ve Hash Code - I



- `java.lang.Object` üzerindeki `hashCode()` metodu, native olarak gerçekleştirilen ve nesnenin bellekteki adresini kullanarak `int` bir değer üreten bir hash fonksiyonu kullanır.
- `hashCode()` metodunun özelliği, tutarlılık açısından `equals()` metodunun `true` döndürdüğü nesneler için aynı `int` değeri hash olarak döndürmesidir.
 - Yani durumu aynı olan aynı tipten nesneler için aynı hash code, durumu aynı olmayan aynı tipten nesneler için farklı hash code söz konusudur.
 - Aynı tipten olmayan nesneler için tabii olarak farklı hash code söz konusudur.

Hashing ve Hash Code - II



- Java'nın bazı sınıflarında hash code şöyle hesaplanır:
 - **Integer sınıfı** `int` değerini hash code olarak çevirir,
 - **Long sınıfı** `(int)(value ^ (value >>> 32))` değerini,
 - **Double sınıfı** `(int)(bits ^ (bits >>> 32))` değerini,
 - **Character sınıfı** `(int)value` değerini,
 - **Boolean sınıfı**, `true` ve `false` için `1231` ve `1237` değerlerini,
 - **String sınıfı** `s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]` değerini döndürür.

HashCodeExample.java



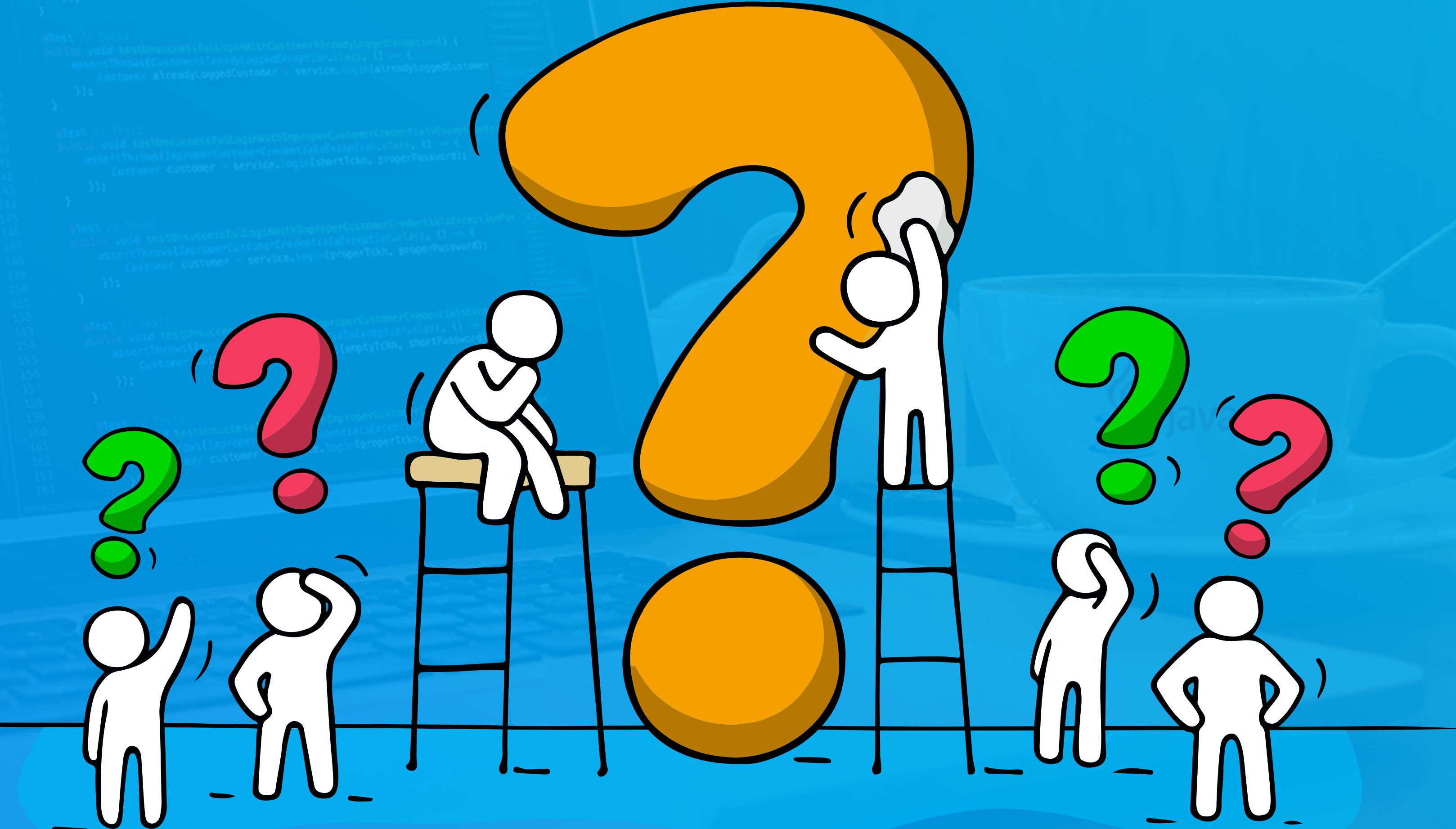
- org.javaturk.oofp.ch10.hash.HashCodeExample

HashingExample.java



- org.javaturk.oofp.ch10.hash.HashingExample

Soru ve Cevap Zamani!





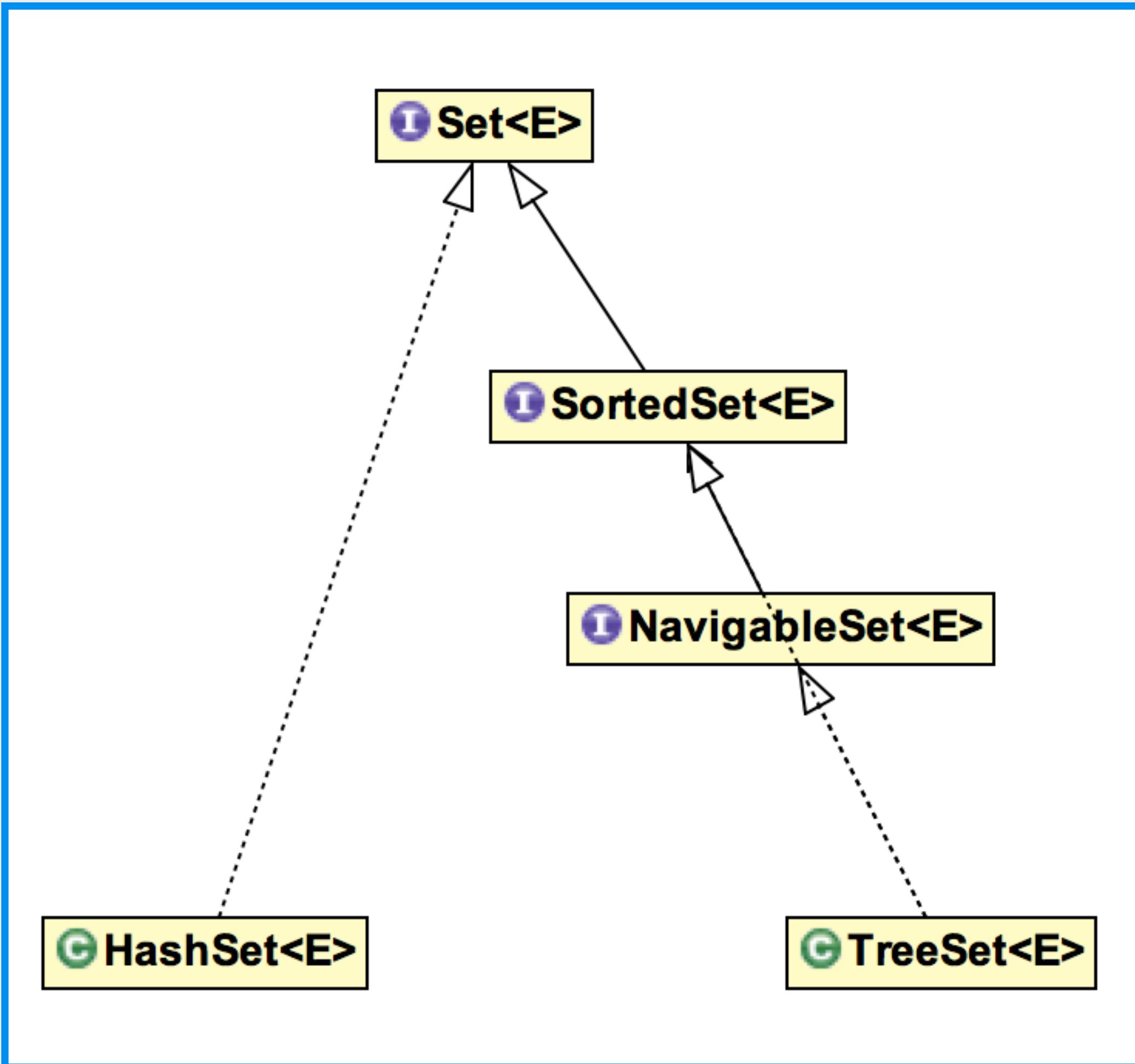
```
111 public void testSuccessfullLogin() {
112     assertThrows(NoSuchCustomerFoundException.class, () -> {
113         Customer loggedCustomer = service.login(properties, properPassword);
114     });
115 }
116
117 @Test // Test 1
118 public void testSuccessfullLoginWithNoSuchCustomerException() {
119     assertThrows(NoSuchCustomerFoundException.class, () -> {
120         Customer unfoundCustomer = service.login(unfoundCustomerToken, properPassword);
121     });
122 }
123
124 @Test // Test 2
125 public void testSuccessfullLoginWithCustomerLockedException() {
126     assertThrows(CustomerLockedException.class, () -> {
127         Customer lockedCustomer = service.login(lockedCustomerToken, "password");
128     });
129 }
130
131 @Test // Test 3
132 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {
133     assertThrows(CustomerAlreadyLoggedException.class, () -> {
134         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerToken, properPassword);
135     });
136 }
137
138 @Test // Test 4
139 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
140     assertThrows(ImproperCustomerCredentialsException.class, () -> {
141         Customer customer = service.login(shortToken, properPassword);
142     });
143 }
144
145 @Test // Test 5
146 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
147     assertThrows(ImproperCustomerCredentialsException.class, () -> {
148         Customer customer = service.login(propertiesToken, properPassword);
149     });
150 }
151
152 @Test // Test 6
153 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(propertiesToken, shortPassword);
156     });
157 }
158
159 @Test // Test 7
160 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
161     assertThrows(ImproperCustomerCredentialsException.class, () -> {
162         Customer customer = service.login(propertiesToken, properties);
163     });
164 }
```

TreeSet

TreeSet - I



- TreeSet, elemanlarını sıralı (sorted) tutan bir başka Set gerçekleştirmesidir.
- TreeSet, hem NavigableSet hem de SortedSet'tir.
- TreeSet, bir Red-Black tree gerçeklestirmesi olan TreeMap tarafından desteklenir.



TreeSet - II



- Dolayısıyla **HashSet** herhangi bir dizilim sözü vermezken, **TreeSet** için dizilim sıralamadır ve var olan dizilim zaman içinde değişebilir.
- **TreeSet**, temel fonksiyonlarda (**add()**, **remove()**, **contains()**) **O(lgn)** bir performans sağlar.

FindDuplicates.java

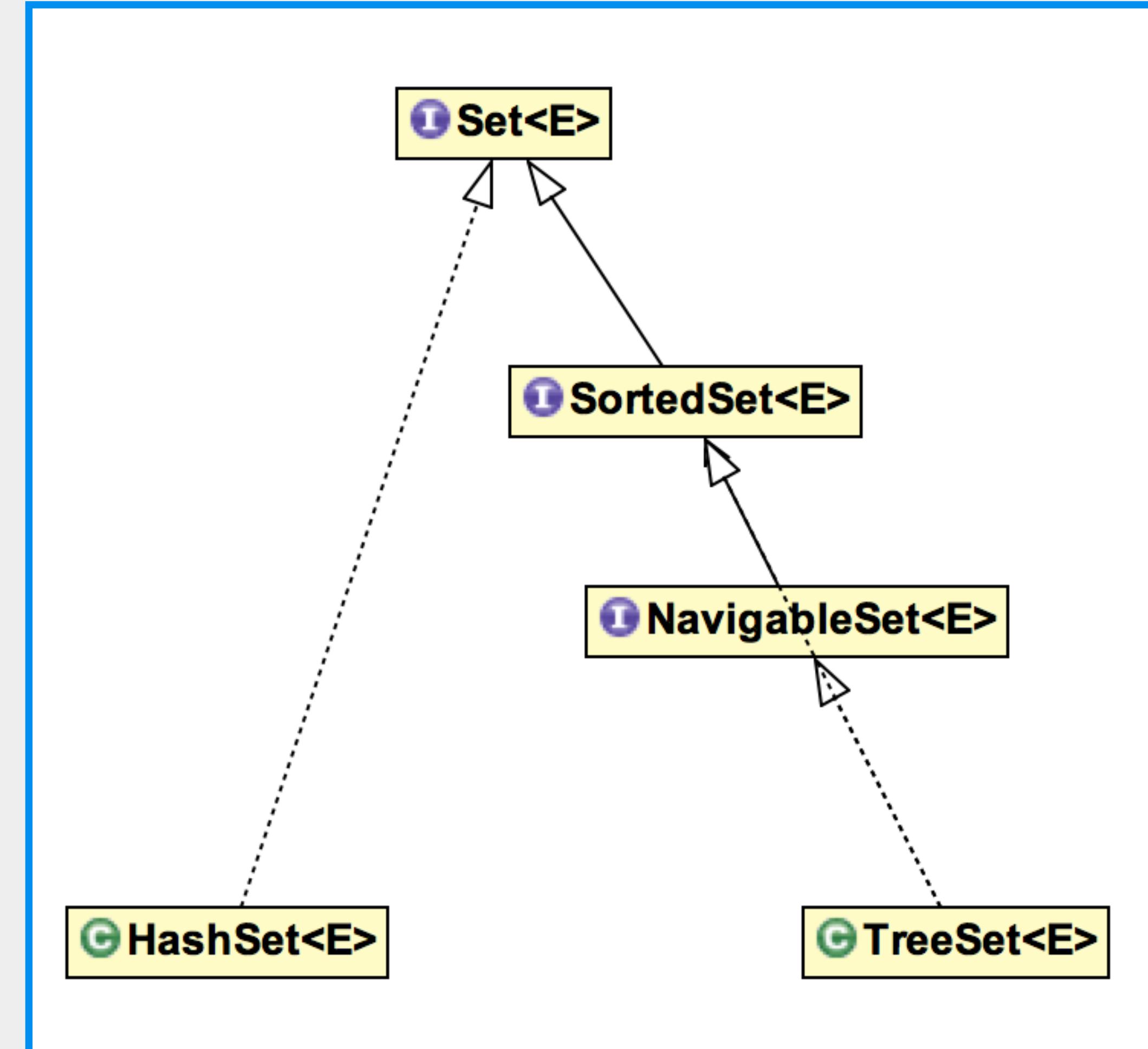


- `org.javaturk.oofp.ch10.set.FindDuplicates`
- Örneği bu sefer de `TreeSet` ile çalıştırın.
- `TreeSet`'ten içindeki elemanları iterator yoluyla alınmasına dikkat edin.
 - Elemanlar hangi sırayla geliyorlar?
 - Eleman eklenirken dizilim değişiyor mu?

TreeSet - III



- TreeSet, hem NavigableSet hem de SortedSet'tır.



SortedSet



- `java.util.SortedSet`, elemanları üzerinde sıralama yapan `Set`'tir.
- `SortedSet`'in elemanları `Comparable` arayüzüünü gerçekleştirmelidir.
- Ya da `SortedSet` gerçekleştirmesine `Comparator` geçilmelidir.

<code>Comparator<? super E></code>	<code>comparator()</code>
<code>E</code>	<code>first()</code>
<code>SortedSet<E></code>	<code>headSet(E toElement)</code>
<code>E</code>	<code>last()</code>
<code>default Spliterator<E></code>	<code>spliterator()</code>
<code>SortedSet<E></code>	<code>subSet(E fromElement, E toElement)</code>
<code>SortedSet<E></code>	<code>tailSet(E fromElement)</code>

NavigableSet



- `java.util.NavigableSet`, `SortedSet`'in yapı içinde arama amacıyla gezmeye ve aranana en yakın elemanı bulmaya izin veren arayüzdür.

```
E           ceiling(E e)
Iterator<E>      descendingIterator()
NavigableSet<E>  descendingSet()
E           floor(E e)
SortedSet<E>      headSet(E toElement)
NavigableSet<E>  headSet(E toElement, boolean inclusive)
E           higher(E e)
Iterator<E>      iterator()
E           lower(E e)
E           pollFirst()
E           pollLast()
NavigableSet<E>  subSet(E fromElement, boolean fromInclusive,
                           E toElement, boolean toInclusive)
SortedSet<E>      subSet(E fromElement, E toElement)
SortedSet<E>      tailSet(E fromElement)
NavigableSet<E>  tailSet(E fromElement, boolean inclusive)
```

NavigableAndSortedSetExample.java



- org.javaturk.oofp.ch10.set.
NavigableAndSortedSetExample



- **TreeSet**, eklenen elemanları sıralamak için şu iki yöntemi kullanır:
 - Eğer elemanların tabi dizilişi (natural ordering) varsa yani gerçekleştiriyorlarsa **Comparable** arayüzü,
 - Elemanların tabi dizilişinin olmadığı dolayısıyla **Comparable** arayüzünün gerçekleştirmediği durumlarda kendisine geçilen **Comparator** arayüzü.

TreeSet - V



- Bu yüzden TreeSet, aşağıdaki kuruculara sahiptir.
- Varsayılan kurucu, boş ve eklenen elemanları tabi dizilişine göre sıralayan bir TreeSet üretir.
- Eğer eklenecek elemanların tabi dizilişi yoksa bu durumda Comparator alan kurucu kullanılır.

`TreeSet()`

`TreeSet(Collection<? extends E> c)`

`TreeSet(Comparator<? super E> comparator)`

`TreeSet(SortedSet<E> s)`



- TreeSet'e eklenecek elemanların sınıflarında **equals()**, **hashCode()** ve Comparable arayüzünden gelen **compareTo()** (ya da Comparator arayüzündeki **compare()**) metodlarının tutarlı bir şekilde sıralamayı gerçekleştirmeleri gereklidir.
- **equals()**'un **true** döndürdüğü nesnelerin **hashCode()** da aynı **int** değeri döndürmeli,
- **equals()**'un **true** döndürdüğü nesnelerin gerçekleştirdiği Comparable arayüzün **compareTo()** (ya da Comparator arayüzündeki **compare()**) metodları da 0 döndürmelidir.



- `equals()` 'un `true` döndürdüğü nesnelerin `hashCode()` metodlarının da aynı `int` değeri döndürmesi, nesnelerin aynılık kontrolüyle ilgiliidir
- `equals()` 'un `true` döndürdüğü nesnelerin gerçekleştirdiği `Comparable` arayüzün `compareTo()` (ya da `Comparator` arayüzündeki `compare()`) metodları da 0 döndürmesi ise `TreeSet`'in eklenen nesneleri sıralamasıyla ilgiliidir.

SetWithDuplicatedObjects.java



- `org.javaturk.oofp.ch10.set.SetWithDuplicatedObjects`
 - Bu örneği **TreeSet** ile çalıştırın.
 - **Employee** nesnelerinin sıralandığını gözlemleyin.
 - Aynı örneği **HashSet** ile çalıştırıldığınızda nesneler sıralanıyor muydu?

SetWithDuplicatedObjects.java



- org.javaturk.oofp.ch10.set.SetWithDuplicatedObjects
 - Employee nesnelerinin Comparable olmadığı durumda TreeSet'in ClassCastException fırlattığını gözlemleyin.
 - Örneği Comparable arayüzüni gerçekleştiren Employee ve
 - Örneği Comparable arayüzüni gerçekleştirmeyen Employee ve Comparator ile çalıştırın.

HashSet mi TreeSet mi?



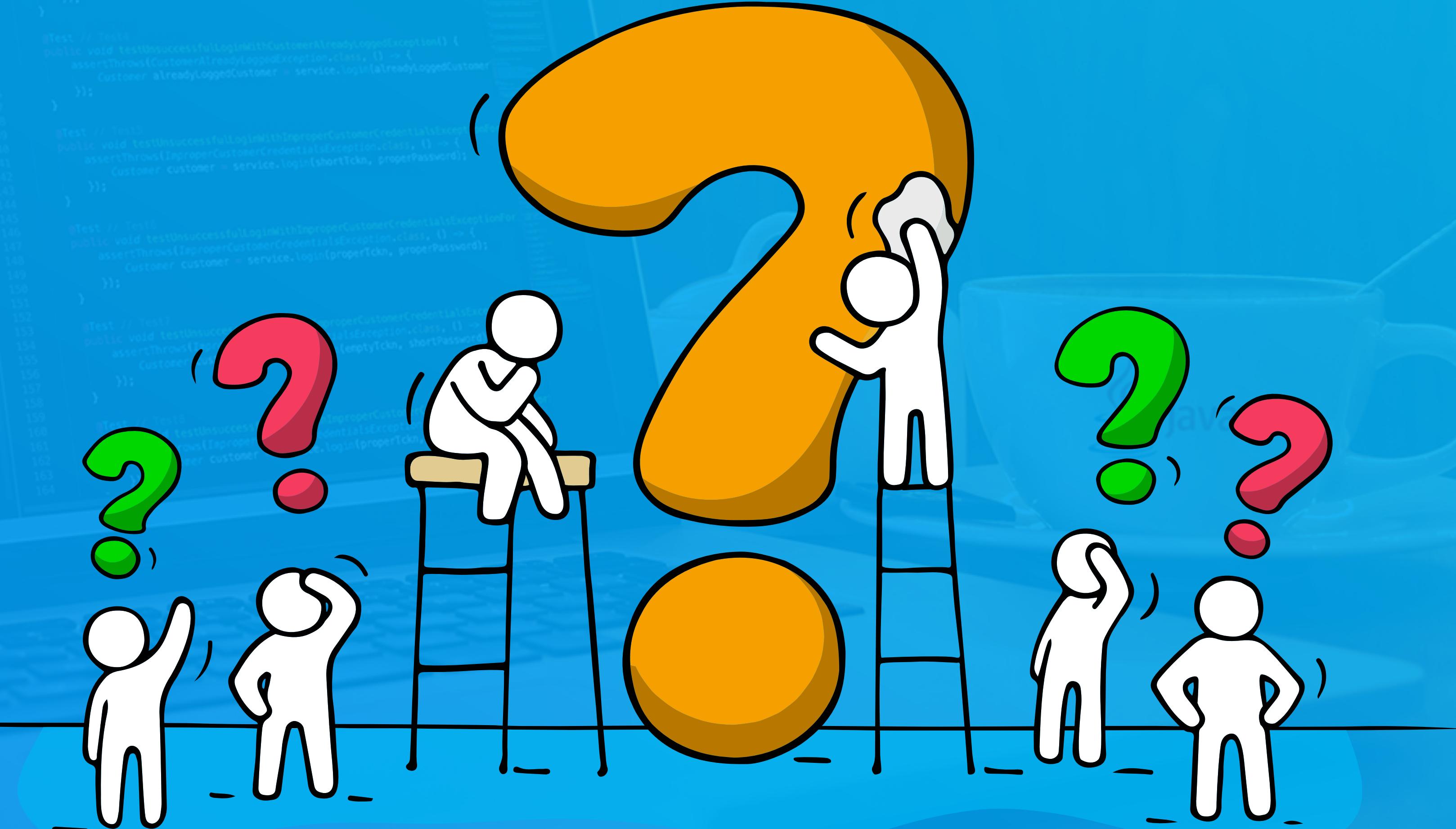
- HashSet ile TreeSet arasındaki temel fark, TreeSet'in elemanlarını sıralamasıdır.
 - Dolayısıyla sıralama isteniyorsa TreeSet kullanılmalıdır.
- Aksi takdirde HashSet kullanılmalıdır,
- Çünkü HashSet, temel fonksiyonlarda (`add()`, `remove()` ve `contains()`) sabite yakın, $O(1)$ performans sağlarken TreeSet çok daha yavaştır, $O(\lg n)$ performans sağlar.

SetPerformance.java



- org.javaturk.oofp.ch10.set.SetPerformance

Soru ve Cevap Zamani!



List ve Gerekleştirmeleri



```
115     assertEquals(successfulCustomer, loggedCustomer);
116 }
117
118 @Test // Test2
119 public void testUnsuccessfulLoginWithNoSuchCustomerException() {
120     assertThrows(NoSuchCustomerFoundException.class, () -> {
121         Customer unfoundCustomer = service.login(unfoundCustomerTckn, properPassword);
122     });
123 }
124
125 @Test // Test3
126 public void testUnsuccessfulLoginWithCustomerLockedException() {
127     assertThrows(CustomerLockedException.class, () -> {
128         Customer lockedCustomer = service.login_lockedCustomerTckn, "qwerty123");
129     });
130 }
131
132 @Test // Test4
133 public void testUnsuccessfulLoginWithCustomerAlreadyLoggedException() {
134     assertThrows(CustomerAlreadyLoggedException.class, () -> {
135         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTckn, properPassword);
136     });
137 }
138
139 @Test // Test5
140 public void testUnsuccessfulLoginWithImproperCustomerCredentialsException() {
141     assertThrows(ImproperCustomerCredentialsException.class, () -> {
142         Customer customer = service.login(shortTckn, properPassword);
143     });
144 }
145
146 @Test // Test6
147 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForEmptyTckn() {
148     assertThrows(ImproperCustomerCredentialsException.class, () -> {
149         Customer customer = service.login(emptyTckn, shortPassword);
150     });
151 }
152
153 @Test // Test7
154 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForEmptyPassword() {
155     assertThrows(ImproperCustomerCredentialsException.class, () -> {
156         Customer customer = service.login(emptyTckn, shortPassword);
157     });
158 }
```



- List arayüzü, dizinin (array) dinamik yani büyüp küçülebilen (enlarging-shrinking) halidir.
- List arayüzü, elemanları için **ardışıl (sequential)** bir dizilim sağlar, dolayısıyla öncelik-sonralık söz konusudur ve bu amaçla, eleman için **yer ya da pozisyon (position)** kavramına sahiptir.
 - Yer ya da pozisyon, **indis (index)** ile de ifade edilir.
- List, Set gibi tekilikle ilgilenmez, bu yüzden aynı elemandan birden fazla tutabilir.

List - II



- **List**'de ilk indis 0 son indis ise **size ()** -1'dir.
- **List** dizilime sahip olduğundan dolayı **add (E e)** ve **addAll (Collection<? extends E> c)** metotları sona ekler.
- **add (E e)** metoduna **Collection<? extends E>** geçilirse **Set**'te olduğu gibi bir odaya tüm torba nesnesini ekler.
- **List**'in ekleme yapan metotlar, **List** değiştirilemez (unmodifiable) olmadıkça, daima **true** döndürür çünkü **List**'i eklemeden geri tutan **Set**'teki gibi bir durum yoktur.

List - III



- List arayüzü, Collection'dan devraldığı davranışlara ardışılığın getirdiği eklemeler yapar.
- Bu ek metodlarda hep yer (positon/index) bilgisi mevcuttur.

```
void      add(int index, E element)  
  
boolean   addAll(int index, Collection<? extends E> c)  
  
E        get(int index)  
  
int      indexOf(Object o)  
  
int      lastIndexOf(Object o)  
  
E        remove(int index)  
  
E        set(int index, E element)  
  
List<E>  subList(int fromIndex, int toIndex)
```



- List, tüm Collection nesneleri gibi Iterable'dır ve iterator() metodunu devralır.
- Ayrıca elemanlarına tek tek ulaşılabilmesi için ListIterator döndüren iki yeni metot listIterator() ve listIterator(int index) ekler.
- listIterator() tüm elemanları, listIterator(int index) ise verilen indixten itibaren elemanları iterate eder.

ListIterator<E>

ListIterator()

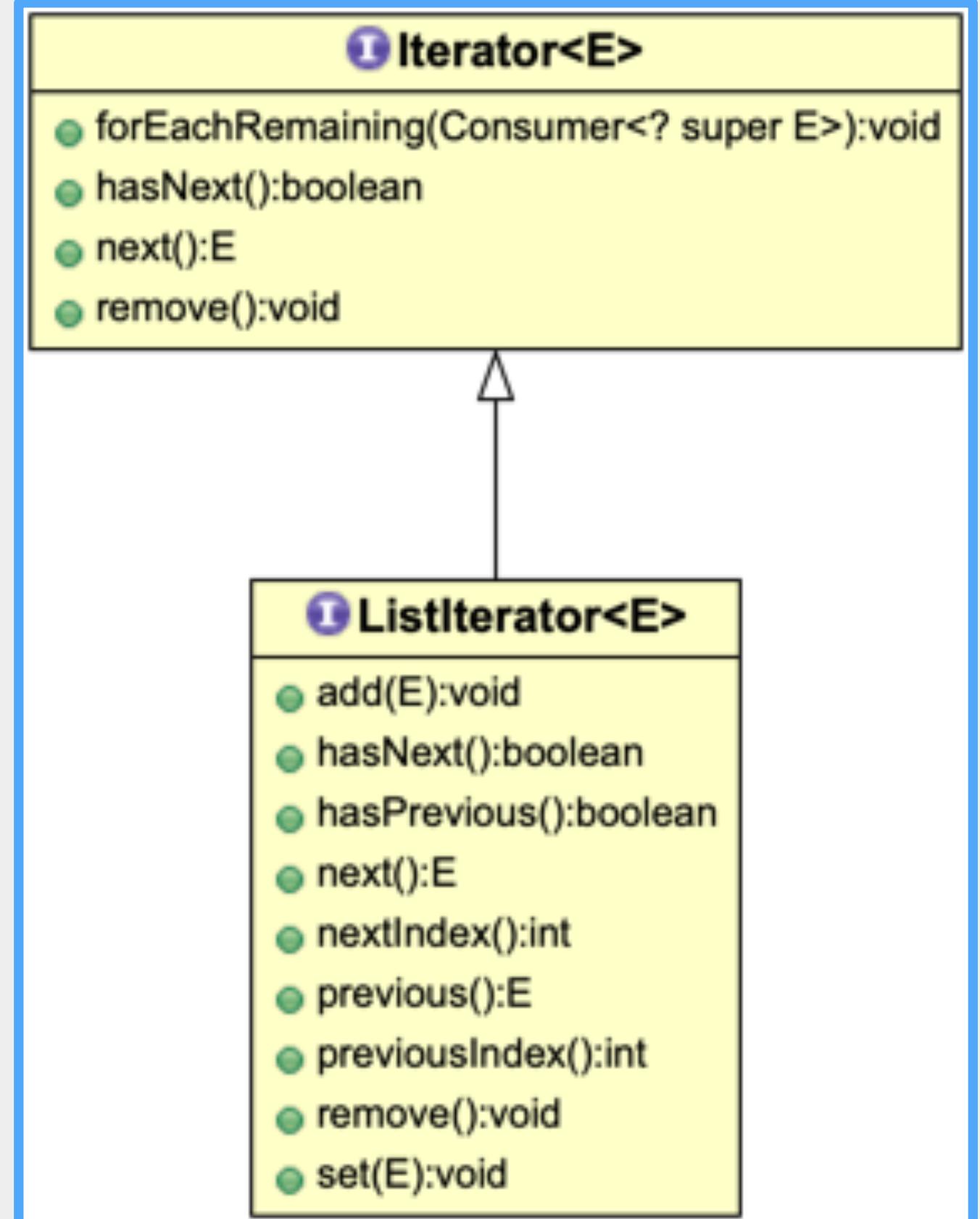
ListIterator<E>

ListIterator(int index)

ListIterator - I



- **ListIterator**, **Iterator** arayüzünden miras devralır ve dizimin sağladığı yeni davranışlar ekler.
- **ListIterator**, elemanlara, ileri ve geri, iki yönde ulaşım sağlar.
 - **hasNext()** ve **next()** ile **hasPrevious()** ve **previous()**
 - **nextIndex()** ve **previousIndex()**



ListIterator - II



- Yeni bir metot olan **add(E e)**, varsa **next()** ile dönecek elemandan önceki ya da varsa **previous()** ile dönecek elemandan sonraki pozisyon'a ekler.
- **remove()** metodu **next()** veya **previous()** ile dönen elemanı alttaki listten siler.
- Benzer şekilde yeni metot **set(E e)** de **next()** veya **previous()** ile dönen elemanın yerine geçileni koyar.



- List'de elemanları sıralamak için **default sort()** metodu da vardır.
 - Bu metot için bir **Comparator** sağlanmalıdır.
 - List'deki tüm elemanlar verilen bir **UnaryOperator<E>**'ün üreteceği bir başka eleman ile değiştirilebilir.
 - Ayrıca List'in herhangi bir parçası başka bir List olarak alınabilir.

```
default void      sort(Comparator<? super E> c)
default void      replaceAll(UnaryOperator<E> operator)
List<E>           subList(int fromIndex, int toIndex)
```

List - VI



- **List**, Java'nın 9. sürümünde eklenen 12 tane `of` isimli, overloaded `static` üretici (factory) metoda sahip olmuştur.

```
static <E> List<E> of()

static <E> List<E> of(E... elements)

static <E> List<E> of(E e1)

static <E> List<E> of(E e1, E e2)

...
```

- Ayrıca **List**'e Java'nın 10. sürümünde `static` kopyalama методу da eklenmiştir.

```
<E> List<E> copyOf(Collection<? extends E> coll)
```

List - VII



- Bu metodlar değiştirilemez (unmodifiable) **List** nesnesi oluşturur.
- Oluşturulan **List** değiştirilmeye çalışıldığında **java.lang.UnsupportedOperationException** fırlatır.
- Ayrıca bu şekilde oluşturulan **List**, **null** elemanlara izin vermez, **null** geçilerek oluşturulmaya çalışıldığında **NullPointerException** fırlatır.

List - VIII



- Ayrıca `java.util.Arrays` sınıfındaki `asList()` metodu da diziden, uzunluğu değişmeyen (fixed-size) `List` oluşturur.

`List<T> asList(T... a)`

- Dizi ile diziden oluşturulan `List` birbirlerinin değişikliklerinden haberdar olurlar.
- `List`, uzunluğunu değiştirecek işlemlerde `java.lang.UnsupportedOperationException` fırlatır.

ListExample.java



- org.javaturk.oofp.ch10.list.ListExample

List Gerçekleştirmeleri - I



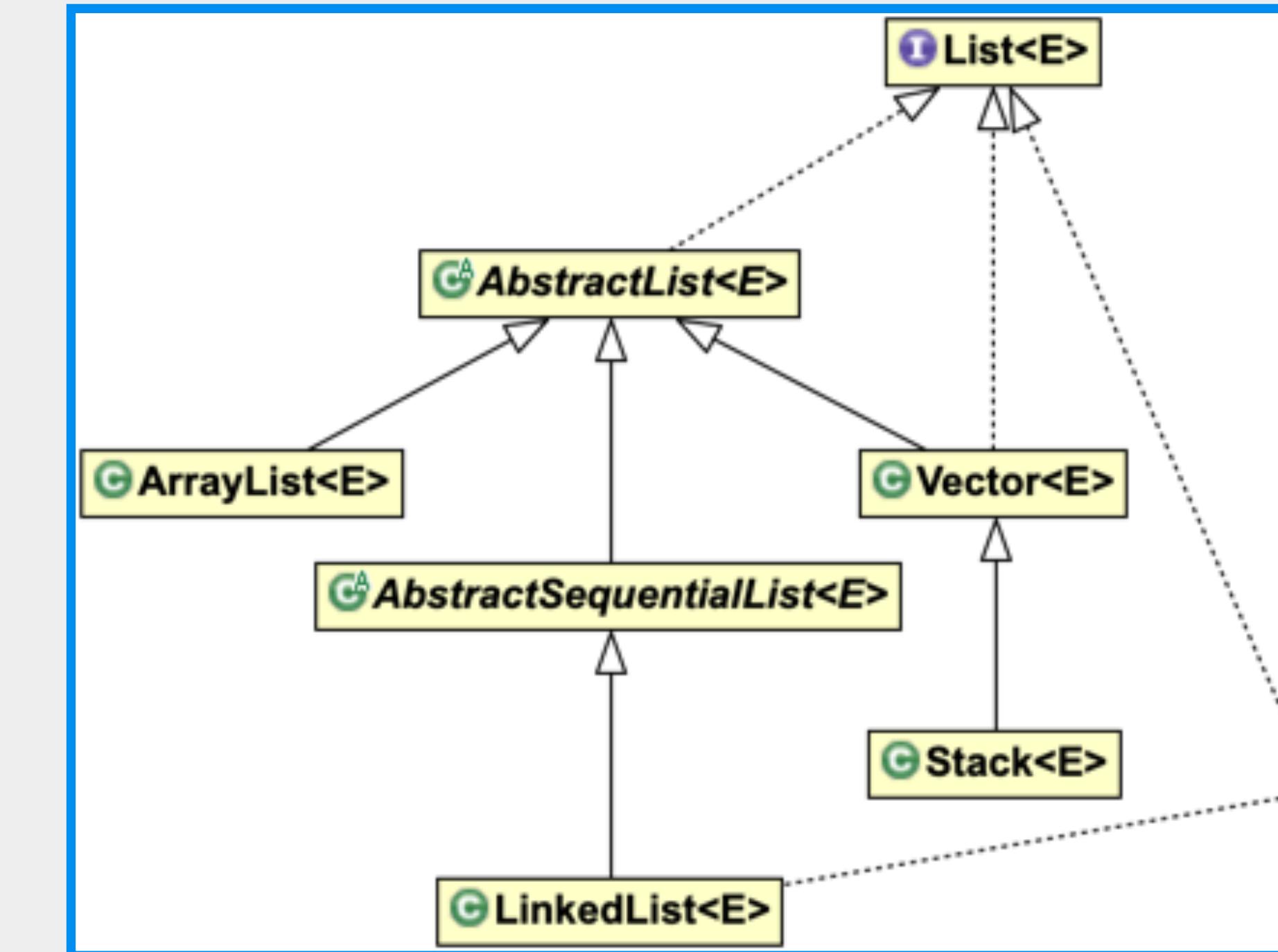
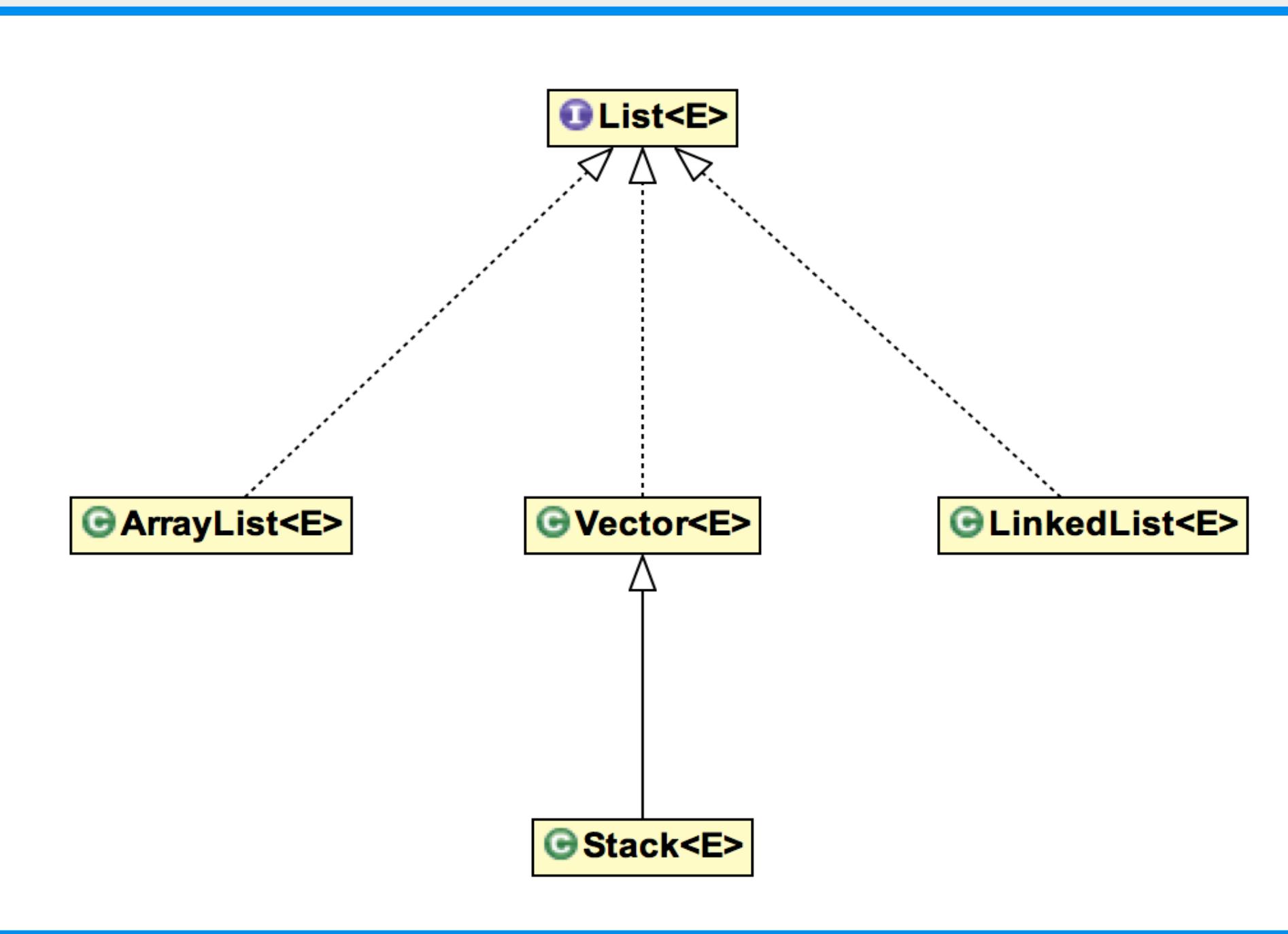
- List arayüzünün çok sık kullanılan iki gerçekleştirmesi vardır:
 - **ArrayList**: Standart ve en çok kullanılan dinamik dizi gerçekleştirmesidir.
 - **LinkedList**: Bağlı liste (linked list) gerçekleştirmesidir.

List Gerçekleştirmeleri - II

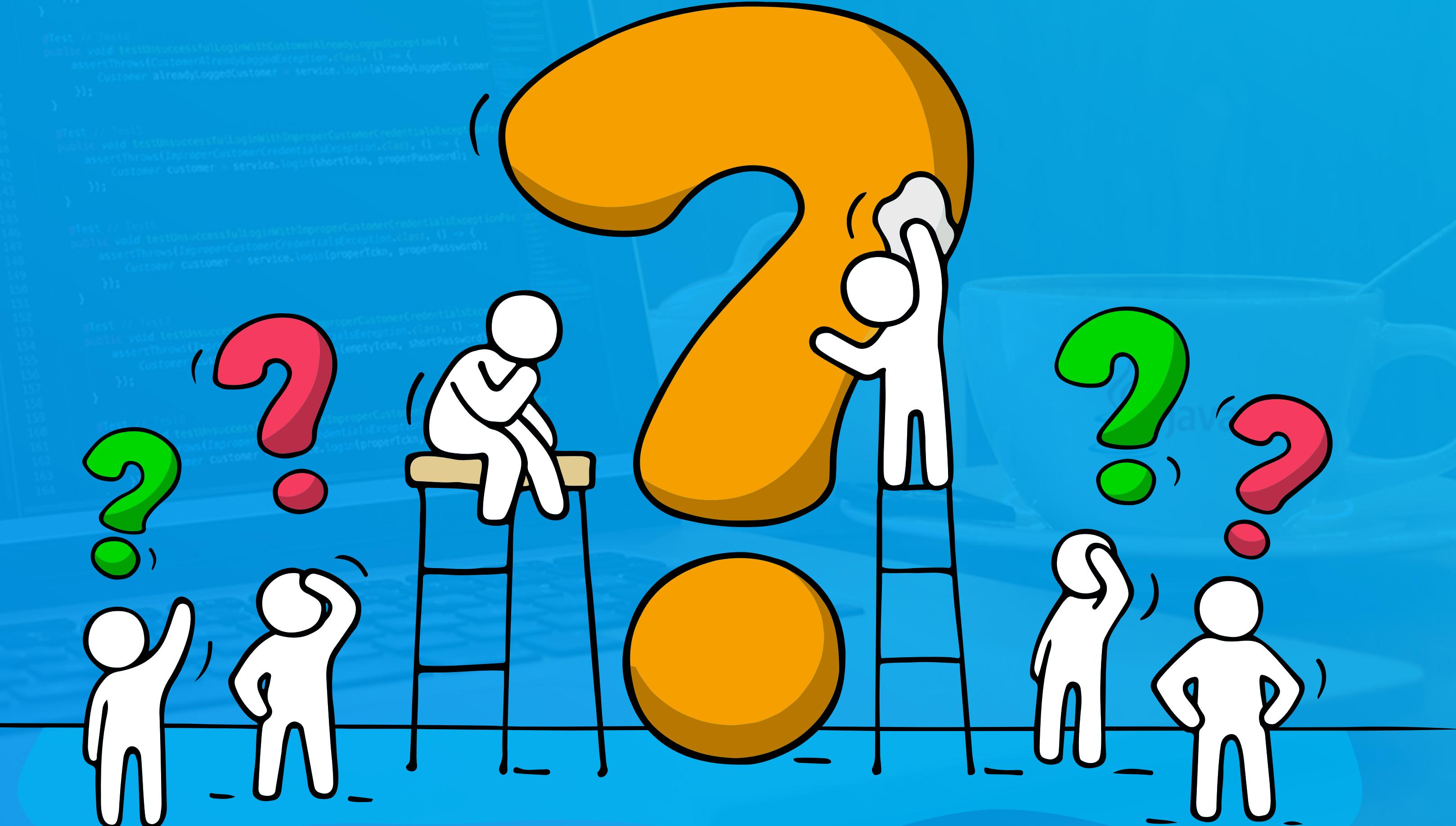


- Bunlar dışında, en baştan bu yana Java'da olan **Vector** ve onun da alt tipi **Stack** gerçekleştirmeleri vardır.
- Java SE 1.2 sürümüyle, APIs'i **List**'e uygun hale getirildi.
 - **Vector**: Dinamik dizi gerçekleştirmesidir.
 - **synchronized** olduğu için pek sık kullanılmaz **ArrayList** ya da **LinkedList** tercih edilir.
 - **Stack**: Yığın gerçekleştirmesidir; çoğunlukla sadece yığın APIs'i kullanılır.

List Gerçekleştirmeleri - III



Soru ve Cevap Zamani!





```
111 public void testSuccessfullLogin() {  
112     assertEquals(CustomerNotFoundException.class, customer.  
113         login("customer", "password").getClass());  
114     assertEquals("Customer logged in successfully.", customer.  
115         login("customer", "password").getLogInMessage());  
116 }  
117  
118 @Test // Test 1  
119 public void testSuccessfullLoginWithUnfoundCustomer() {  
120     assertEquals(UnfoundCustomerException.class, () -> {  
121         customer.unfoundCustomer = service.login("customer", "password");  
122     }).getClass();  
123 }  
124  
125 @Test // Test 2  
126 public void testSuccessfullLoginWithLockedCustomer() {  
127     assertEquals(CustomerLockedException.class, () -> {  
128         customer.lockedCustomer = service.login("lockedCustomer", "password");  
129     }).getClass();  
130 }  
131  
132 @Test // Test 3  
133 public void testSuccessfullLoginWithAlreadyLoggedCustomer() {  
134     assertEquals(CustomerAlreadyLoggedException.class, () -> {  
135         customer.alreadyLoggedCustomer = service.login("alreadyLoggedCustomer", "password");  
136     }).getClass();  
137 }  
138  
139 @Test // Test 4  
140 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
141     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
142         customer = service.login("shortTckn", "properPassword");  
143     }).getClass();  
144 }  
145  
146 @Test // Test 5  
147 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
148     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
149         customer = service.login("properTckn", "properPassword");  
150     }).getClass();  
151 }  
152  
153 @Test // Test 6  
154 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
155     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
156         customer = service.login("properTckn", "shortPassword");  
157     }).getClass();  
158 }  
159  
160 @Test // Test 7  
161 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
162     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
163         customer = service.login("properTckn", "properShortPassword");  
164     }).getClass();  
165 }
```

ArrayList ve LinkedList

ArrayList



- **ArrayList**: En standart ve sık kullanılan **List** gerçekleştirmesidir.
- **ArrayList**'in üç tane kurucusu vardır.
 - **ArrayList**'in varsayılan kurucusu boş bir list oluşturur.
 - **ArrayList**'in girilen değer kadar odaya sahip olan bir nesne oluşturan kurucusu ile geçen **Collection**'un elemanlarıyla list oluşturan kurucusu da vardır.

`ArrayList()`

`ArrayList(int initialCapacity)`

`ArrayList(Collection<? extends E> c)`

LinkedList - I



- **LinkedList**, bağlı listedir.
- **LinkedList**'deki her eleman, sağ ve sol taraftaki komşularını bilir.
- Yani **LinkedList**'in **iki taraflı bağlı listdir (double-linked list)**.
- **LinkedList**'in iki kurucusu vardır; varsayılan kurucu boş bir list, diğer ise geçilen **Collection**'un elemanlarıyla list oluşturur.

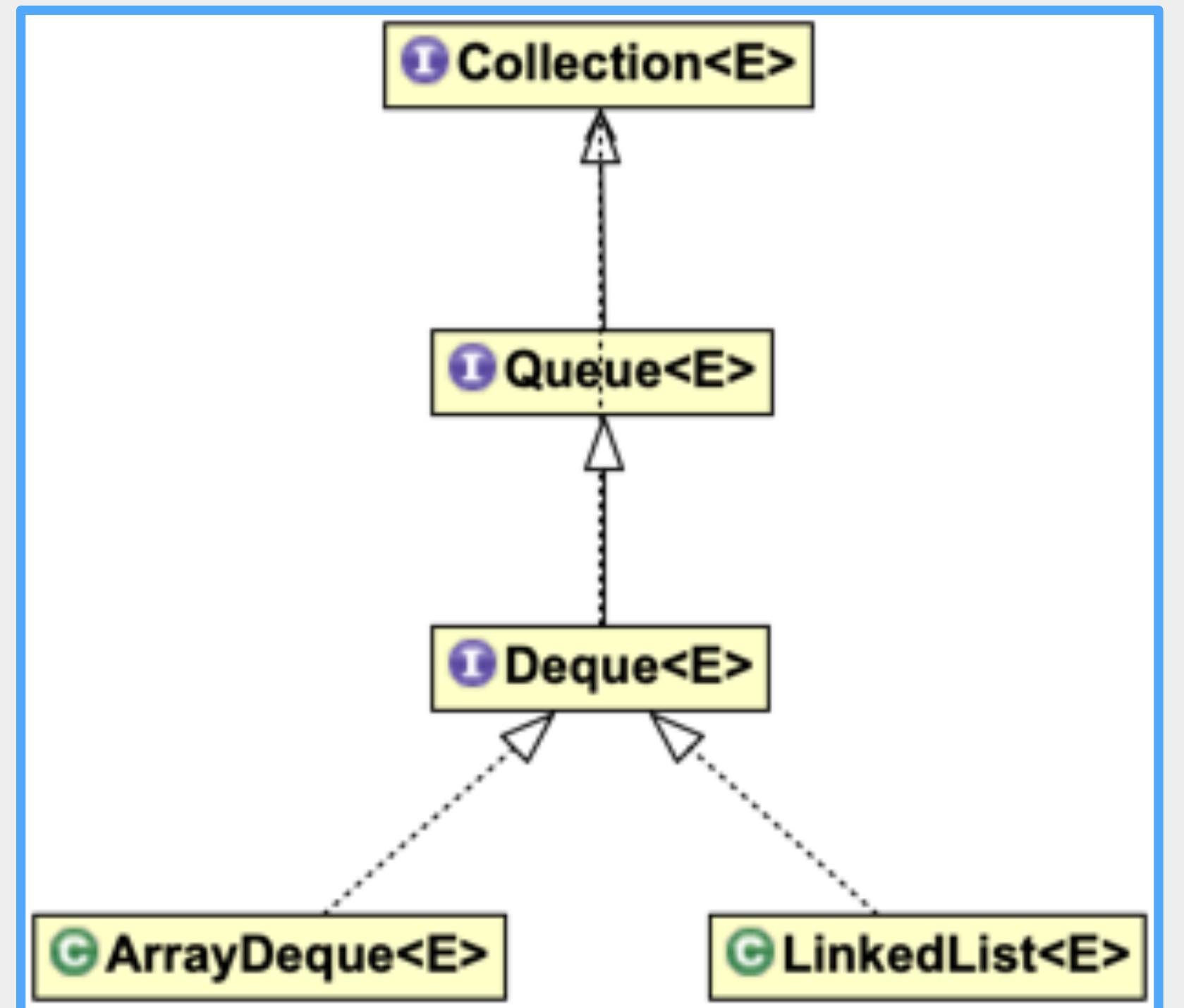
LinkedList()

LinkedList(Collection<? extends E> c)

LinkedList - II



- `LinkedList`, ayrıca `Queue` ve `Deque` arayüzlerini de gerçekleştirir, bundan dolayı `ArrayList`'ten farklı metodları vardır.



ArrayList mi LinkedList mi? - I



- **ArrayList** ile **LinkedList** arasındaki temel fark, **LinkedList**'in elemanlarının her iki tarafındaki komşu elemanı biliyor olmasıdır.
- Bu ise **LinkedList** ve **ArrayList** arasında farklı davranışlar için farklı performansa sebep olur.
 - Erişimde **ArrayList** sabit, $O(1)$ performansa sahipken **LinkedList** elemanı linkler üzerinden giderek bulduğu için doğrusal bir performansa $O(n)$ sahiptir.
 - Ekleme ve çıkarmada (**add()** & **remove()**) **ArrayList** $O(n)$, **LinkedList** ise sabit, $O(1)$ performansa sahiptir.

ArrayList mi LinkedList mi? - II



- Dolayısıyla, başa ya da araya ekleme-çıkarma yapılmadığında **ArrayList** tercih edilmelidir.
- Çünkü **LinkedList**'te eleman erişimi, linkler üzerinden gidilerek yapıldığından, lineerdir, pahalıdır.
- Ama başa ya da araya ekleme-çıkarma yapılacaksa **LinkedList** tercih edilmelidir.
- Çünkü **ArrayList**'te sona yapılanlar dışındaki ekleme-çıkarma işlemleri doğrusal karmaşıklığa sahiptir, pahalıdır.

ArrayList mi LinkedList mi? - III



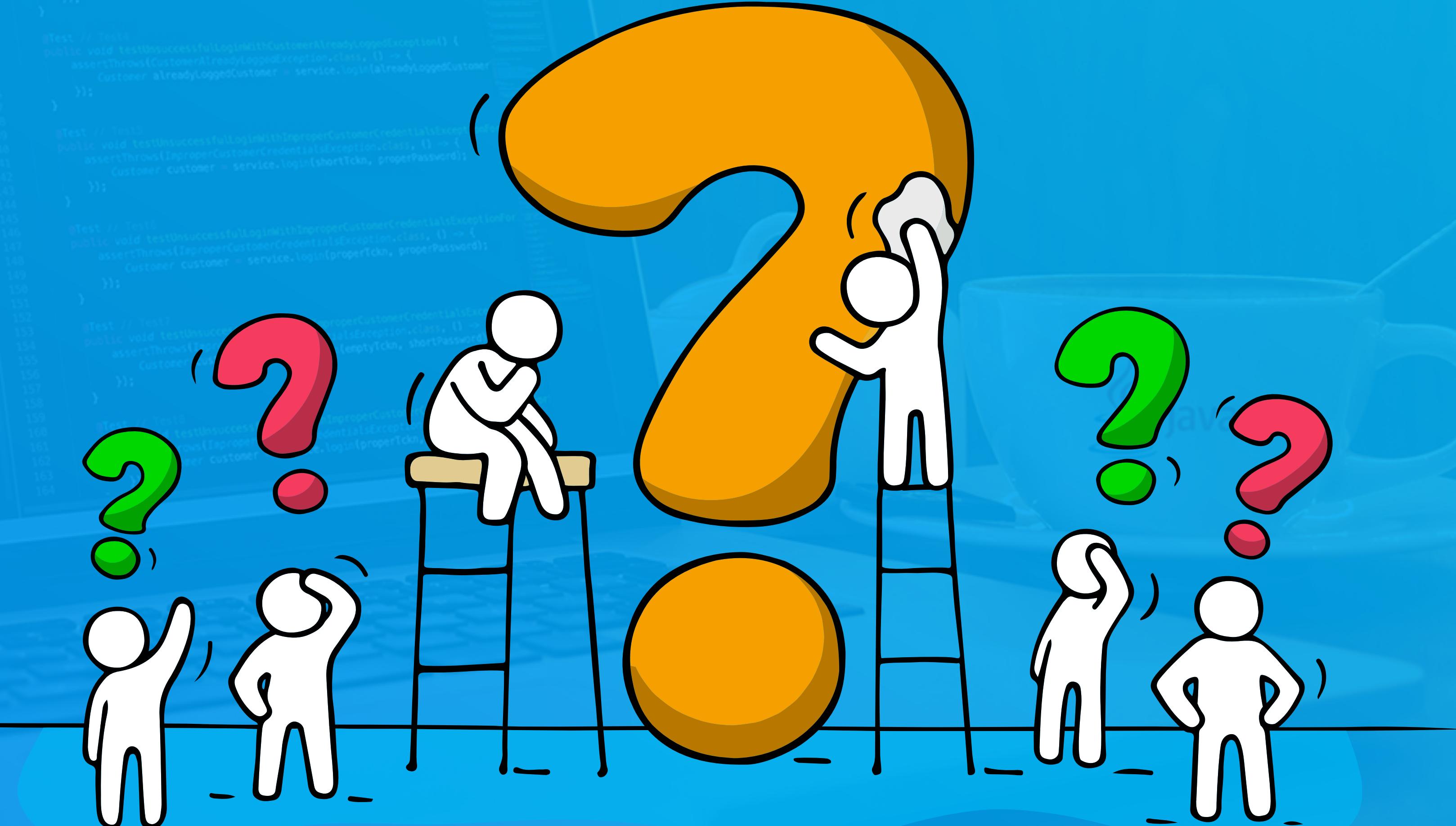
- Arama, örneğin `contains()` metot çağrıları, her iki yapı için de $O(n)$ 'dır.

ListPerformance.java



- org.javaturk.oofp.ch10.list.ListPerformance

Soru ve Cevap Zamani!



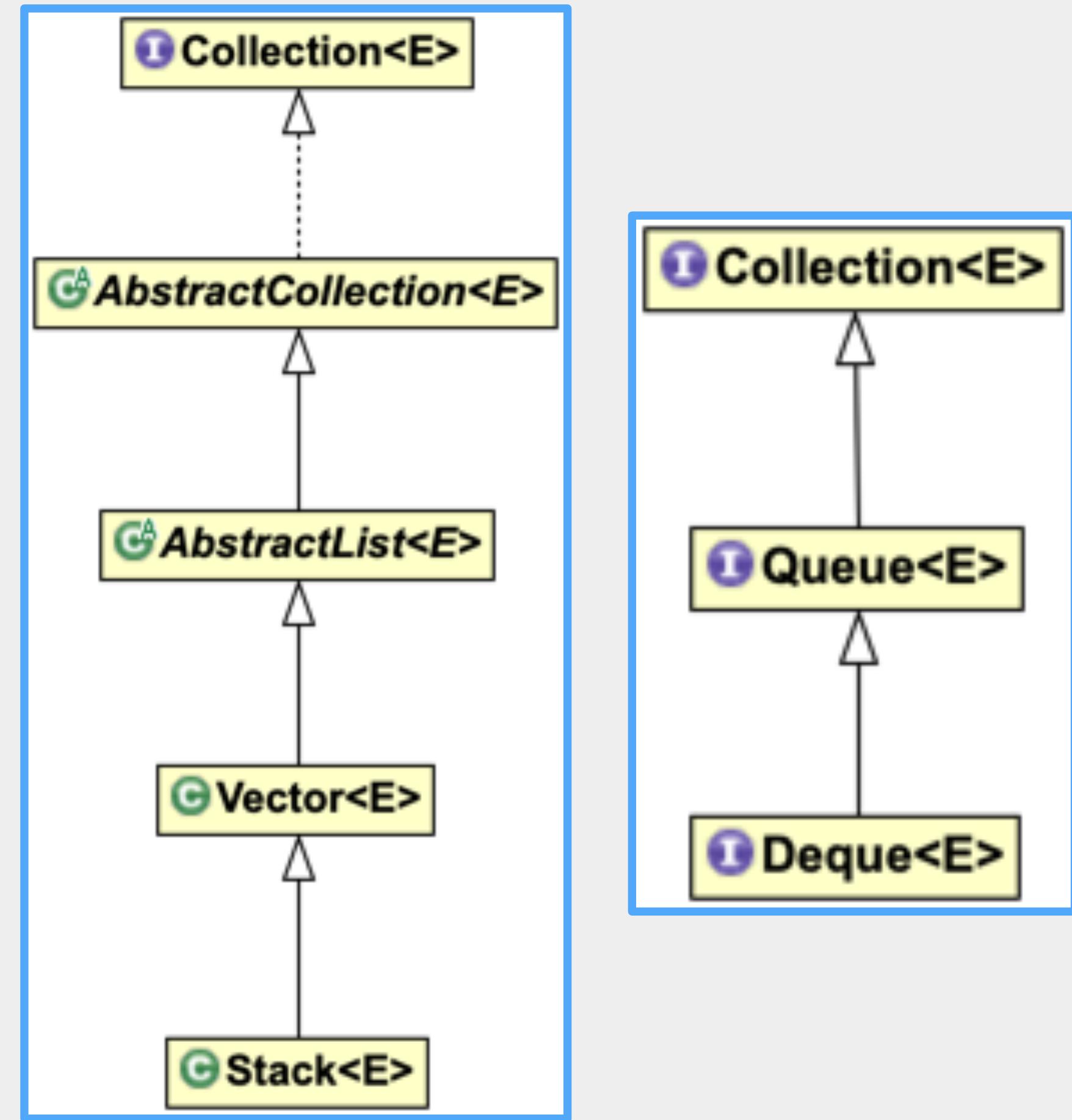
Düğü Collection Nesneleri



Diğer Collection Nesneleri



- Buraya kadar ele alınan **Set** ve **List** arayüzleri ve **HashSet** ve **TreeSet** ile **ArrayList** ve **LinkedList** gerçekleştirmeleri dışında **Collection** arayüzünün başka alt arayüzleri ve gerçekleştirmeleri de vardır.
 - **Queue** ve **Deque** arayüzleri
 - **Stack** ve **Vector** sınıfları





```
111 public void testSuccessfulLogin() {
112     // Given
113     // When
114     // Then
115     Customer loggedCustomer = service.login(properties, properPassword);
116     assertEquals(successfulCustomer, loggedCustomer);
117 }
118
119 @Test // Test 1
120 public void testSuccessfulLoginWithNoSuchCustomerException() {
121     // Given
122     // When
123     // Then
124     assertThrows(NoSuchCustomerNotFoundException.class, () -> {
125         Customer unfoundCustomer = service.login(unfoundCustomerTkn, properPassword);
126     });
127 }
128
129 @Test // Test 2
130 public void testSuccessfulLoginWithCustomerLockedException() {
131     // Given
132     // When
133     // Then
134     assertThrows(CustomerLockedException.class, () -> {
135         Customer lockedCustomer = service.login(lockedCustomerTkn, "password");
136     });
137 }
138
139 @Test // Test 3
140 public void testSuccessfulLoginWithCustomerAlreadyLoggedException() {
141     // Given
142     // When
143     // Then
144     assertThrows(CustomerAlreadyLoggedException.class, () -> {
145         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTkn, properPassword);
146     });
147 }
148
149 @Test // Test 4
150 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {
151     // Given
152     // When
153     // Then
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(shortTkn, properPassword);
156     });
157 }
158
159 @Test // Test 5
160 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {
161     // Given
162     // When
163     // Then
164     assertThrows(ImproperCustomerCredentialsException.class, () -> {
165         Customer customer = service.login(propertiesTkn, properPassword);
166     });
167 }
```

Vector



- `java.util.Vector` de dinamik bir listedir.
- Java'nın ilk günlerinden kalan bir nesnedir, API'si Java 1.2 ile tekrar düzenlenmiş ve `List`'i gerçekleştirecek hale getirilmiştir.
- Dolayısıyla şu anda hem eski metodlara hem de `List` metodlarına sahiptir.
- Örneğin, `E firstElement()`, `E lastElement()`,
`Enumeration<E> elements()` metodları vardır.

Vector - II



- **Vector** diğer **Collection** nesnelerinin tersine, **synchronized**'dır.
- Dolayısıyla çoklu kanallı (multi-threaded) ortam söz konusu değilse **Vector** kullanılmamalıdır çünkü gereksiz performans kaybı yaşatır.
- Böyle durumlarda **ArrayList** ya da **LinkedList** tercih edilmelidir.

VectorExample.java



- org.javaturk.oofp.ch10.list.VectorExample



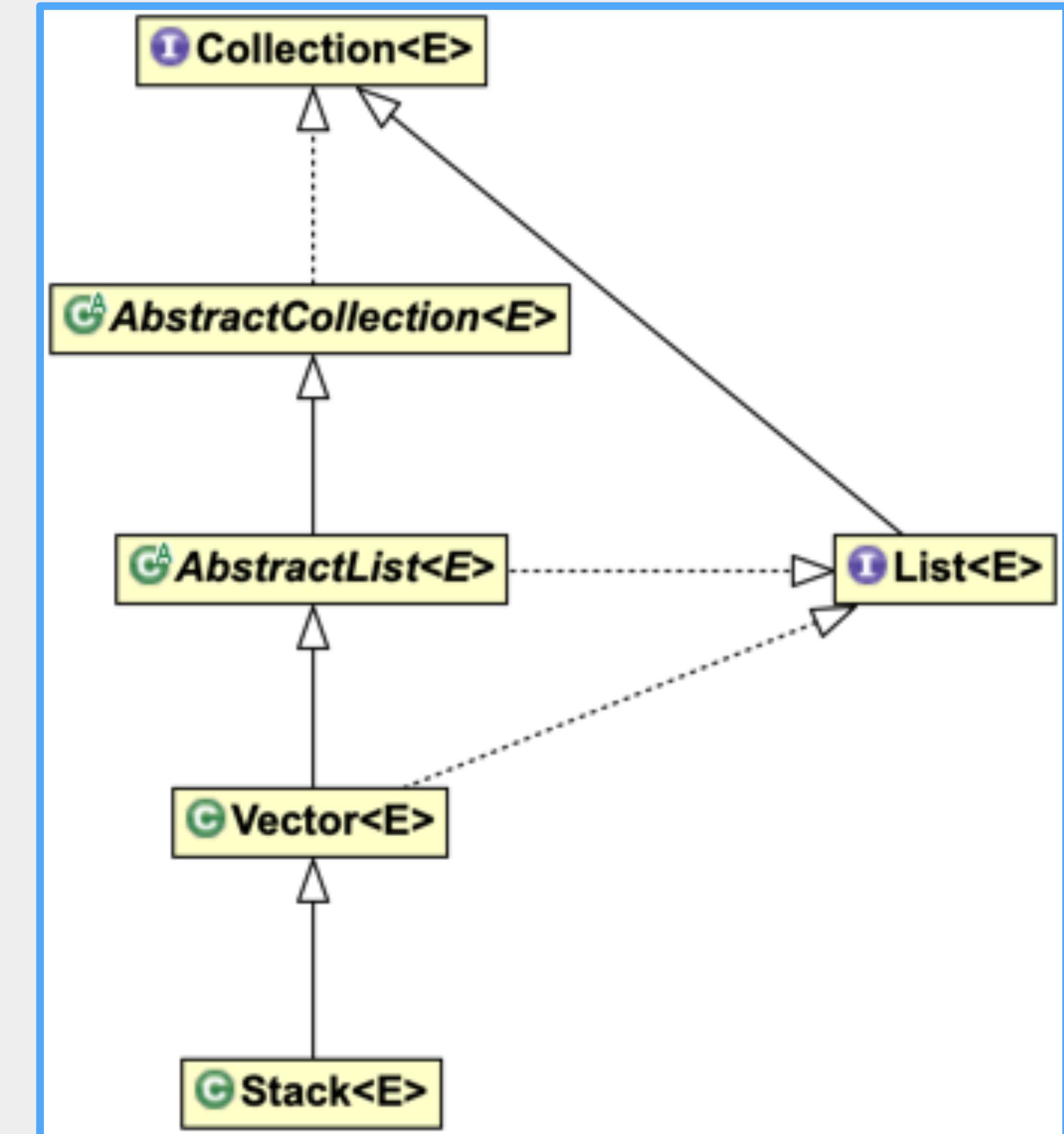
```
111 public void testSuccessfullLogin() {
112     // Given
113     // When
114     // Then
115     Customer loggedCustomer = service.login(properties, properPassword);
116     assertEquals(successfulCustomer, loggedCustomer);
117 }
118
119 @Test // Test 2
120 public void testSuccessfullLoginWithNoSuchCustomerException() {
121     // Given
122     // When
123     // Then
124     assertThrows(NoSuchCustomerNotFoundException.class, () -> {
125         Customer unfoundCustomer = service.login(unfoundCustomerToken, properPassword);
126     });
127 }
128
129 @Test // Test 3
130 public void testSuccessfullLoginWithCustomerLockedException() {
131     // Given
132     // When
133     // Then
134     assertThrows(CustomerLockedException.class, () -> {
135         Customer lockedCustomer = service.login(lockedCustomerToken, "password");
136     });
137 }
138
139 @Test // Test 4
140 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {
141     // Given
142     // When
143     // Then
144     assertThrows(CustomerAlreadyLoggedException.class, () -> {
145         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerToken, "password");
146     });
147 }
148
149 @Test // Test 5
150 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
151     // Given
152     // When
153     // Then
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(shortToken, properPassword);
156     });
157 }
158
159 @Test // Test 6
160 public void testSuccessfullLoginWithImproperCustomerCredentialsSpecificationException() {
161     // Given
162     // When
163     // Then
164     assertThrows(ImproperCustomerCredentialsSpecificationException.class, () -> {
165         Customer customer = service.login(propertiesToken, properPassword);
166     });
167 }
```

Stack

Stack - I



- `java.util.Stack`, adından da anlaşılacağı gibi, elemanları üst üste koyan bir yiğindir ve “son giren ilk çıkar” (last-in-first-out, LIFO) ilkesiyle çalışır.
- `Stack`, hem bir `List` geçeklestirmesi hem de `Vector`'ün bir alt tipidir dolayısıyla ikisinin de APIsini destekler.



Stack - II



- Stack'in boş bir yiğin oluşturan bir tane kurucusu vardır:

Stack()

- Stack bir List'tir ama onu “yiğin” yapan farklı metotları da vardır:

E peek()
E pop()
E push(E item)
int search(Object o)

StackExample.java



- org.javaturk.oofp.ch10.list.StackExample



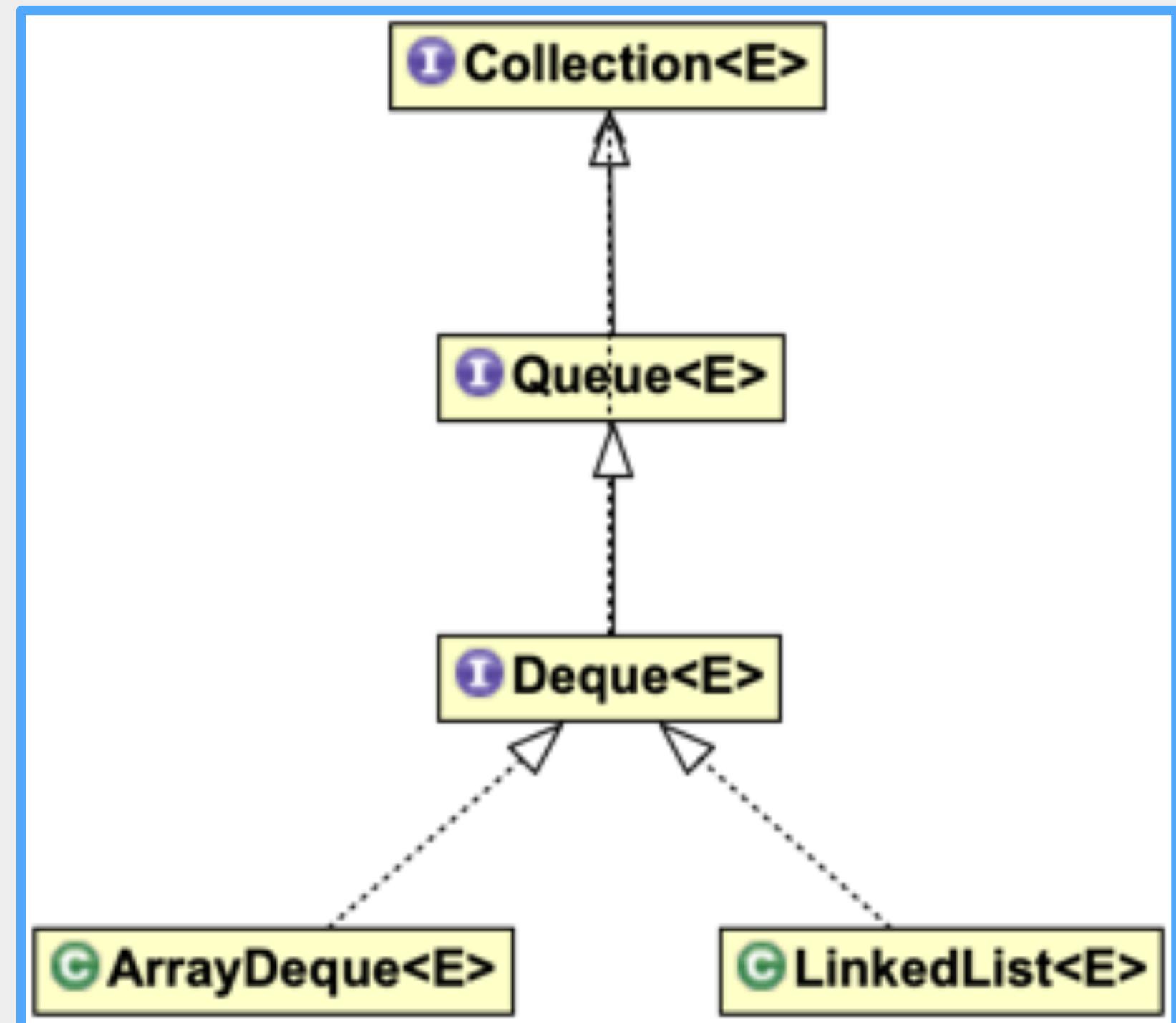
```
111 public void testSuccessfulLogin() {  
112     assertEquals(CustomerNotFoundException.class, customer.  
113         login("customer", "password").getClass());  
114     assertEquals("Customer", customer.loggedCustomer);  
115     assertEquals("Customer", customer.loggedCustomer);  
116 }  
117  
118 @Test // Test 1  
119 public void testSuccessfulLoginWithNoSuchCustomerException() {  
120     assertEquals(NoSuchCustomerException.class, () -> {  
121         customer.unfoundCustomer = service.login("customer", "password");  
122     }).  
123 }  
124  
125 @Test // Test 2  
126 public void testSuccessfulLoginWithCustomerLockedException() {  
127     assertEquals(CustomerLockedException.class, () -> {  
128         customer.lockedCustomer = service.login("lockedCustomer", "password");  
129     }).  
130 }  
131  
132 @Test // Test 3  
133 public void testSuccessfulLoginWithCustomerAlreadyLoggedException() {  
134     assertEquals(CustomerAlreadyLoggedException.class, () -> {  
135         customer.alreadyLoggedCustomer = service.login("alreadyLoggedCustomer", "password");  
136     }).  
137 }  
138  
139 @Test // Test 4  
140 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {  
141     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
142         customer = service.login("shortTckn", "properPassword");  
143     }).  
144 }  
145  
146 @Test // Test 5  
147 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {  
148     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
149         customer = service.login("properTckn", "properPassword");  
150     }).  
151 }  
152  
153 @Test // Test 6  
154 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {  
155     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
156         customer = service.login("properTckn", "shortPassword");  
157     }).  
158 }  
159  
160 @Test // Test 7  
161 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {  
162     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
163         customer = service.login("properTckn", "properTckn");  
164     }).  
165 }
```

Queue ve Dequeue

Queue ve Deque



- Queue ve Deque, kuyruk ve iki yönlü kuyruk arayüzleridir.
- ArrayDeque ve LinkedList gerçekleştirmeleri vardır.

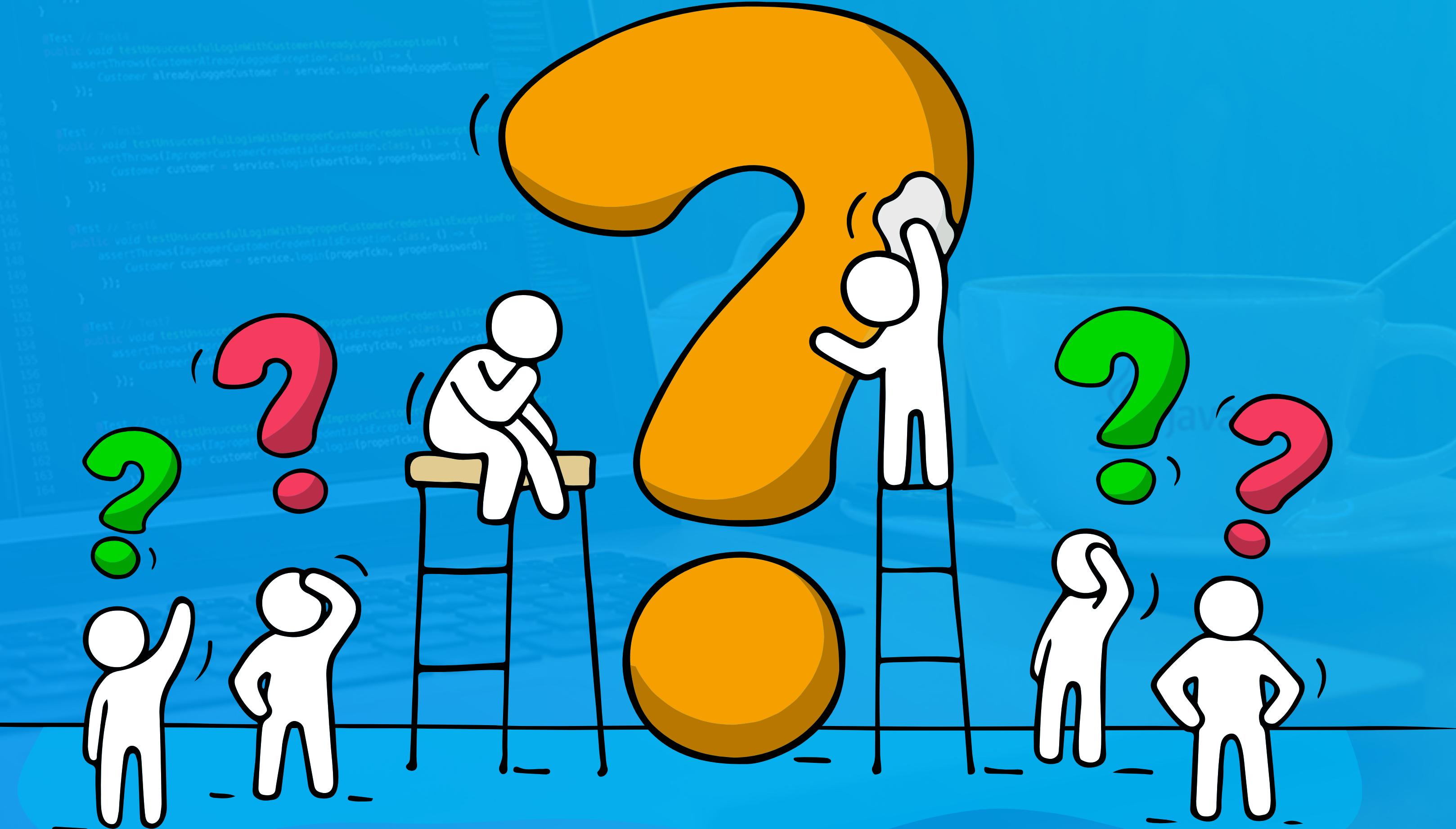


QueueExample.java



- org.javaturk.oofp.ch10.list.QueueExample

Soru ve Cevap Zamani!



Map ve Gerekleştirmeleri



Map - I



- **Map<K , V>**, elemanlarını anahtar nesnelerle eşleştirerek tutan torbaların ana arayüzüdür.
- Dolayısıyla **Map**, sözlük (dictionary) gibi, **anahtar – değer (key-value pair)** ikilisi saklar.
 - Saklanan eleman değerdir ve bir anahtar ile eşleştirilerek torbada tutulur.
 - Bundan dolayı da genelde elemanlara anahtarlarıyla ulaşılır.
- Bu yapısından dolayı **Map**'e ilişkilendiren dizi (associative array) de denir.

Map - II



- Map'de, değerleri anahtar ile ilişkili olarak tuttuğu için ekleme metoduna iki nesne geçilir: **v put (K, v)**
- **put (K, v)**, geçen anahtar ile ilişkili bir değer varsa onu geçen değer ile değiştirir ve eski değeri geri döndürür, yoksa **null** geri döndürür.
- Çünkü Map'te aynı anahtarla ilişkili sadece tek bir değer olabilir, bundan dolayı aynı anahtar için yeni bir değer **put ()** ile girilirse değer değişir.

V	put(K key, V value)
void	putAll(Map<? extends K, ? extends V> m)
default V	putIfAbsent(K key, V value)

Map - III



- Eğer Map'te anahtar yoksa değeri ile birlikte eklenmesi isteniyorsa **putIfAbsent()** kullanılmalıdır.
- **default V putIfAbsent(K, V)** metodu sadece eğer geçen anahtarla ilişkili bir değer yok ise değeri Map'e ekler, var ise var olanla değiştirmez.

```
V          put(K key, V value)  
void      putAll(Map<? extends K, ? extends V> m)  
default V  putIfAbsent(K key, V value)
```

Map - IV



- Map'ten değer almak için anahtar verilir, anahtara karşı gelen değer alınır: `v get(K)`
- Eğer Map'te anahtar karşı gelen bir değer yoksa, varsayılan bir değerin dönmesi sağlanabilir: `default v getOrDefault(K, v)`

<code>V</code>	<code>get(Object key)</code>
<code>default V</code>	<code>getOrDefault(Object key, V defaultValue)</code>

Map - V



- Map'de iki Collection nesnesi vardır:
 - Anahtarları tutan Set'tir, dolayısıyla anahtarlar tekildir.
 - Anahtar Set'i keySet() ile alınır.
 - Değerleri tutan ise Collection'dır, dolayısıyla farklı anahtarlarla eşleştirilmiş aynı nesneler tutulabilir.
 - Değer Collection'ı values() ile alınır.

Set<K>	keySet()
Collection<V>	values()

Map - VI



- Map'de üçüncü bir Collection da anahtar-değer ikilisini temsil eden EntrySet'tir:
- EntrySet, Set<Map.Entry<K, V>> entrySet() ile alınır.
- java.util.Map.Entry<K, V> bir arayüzdür ve anahtar-değer ikilisini tutar.

Set<K>	keySet()
Collection<V>	values()
Set<Map.Entry<K, V>>	entrySet()

MapEntry



- `java.util.Map.Entry`, Map'teki anahtar-değer çiftini (key-value pair) ifade eden bir arayüzdür.
- Arayüzden anahtar ve değer alınabilir.

```
static <K extends Comparable<? super K>,V> Comparator<Map.Entry<K,V>> comparingByKey()

static <K,V extends Comparable<? super V>> Comparator<Map.Entry<K,V>> comparingByValue()

static <K,V> Comparator<Map.Entry<K,V>> comparingByKey(Comparator<? super K> cmp)

static <K,V> Comparator<Map.Entry<K,V>> comparingByValue(Comparator<? super V> cmp)

boolean equals(Object o)

K getKey()

V getValue()

V setValue(V value)
```

Map'ten Anahtar-Değer Alma



- Map'de MapEntry yerine anahtarlar ve değerlerini ayrı ayrı almak için önce keySet() ile anahtar Set'i alınır ve anahtarlar üzerinden Iterator ile geçilir,
- Her anahtar get() metoduna geçilerek karşı gelen değeri alınır.

```
Set keys = map.keySet();
Iterator iterator = keys.iterator();
while (iterator.hasNext()) {
    Key key = (Key) iterator.next();
    Value value = (Value) map.get(key);
}
```

Map - VII



- **forEach ()** : Anahtar ve değer geçilen **BiConsumer** nesnesi alır.
- **merge ()** : Eğer geçilen anahtar bir değer ile ilişkili olarak **Map**'te yoksa ya da varsa ama ilişkili değer **null** ise, geçilen anahtarı **BiFunction**'ın ürettiği değer ile ilişkilendirir.

```
default void forEach(BiConsumer<? super K, ? super V> action)
```

```
default V merge(K key, V value, BiFunction<? super V, ? super V, ? extends V> remappingFunction)
```

Map - VIII



- Map'ten hem anahtar hem de anahtar-değer ikilisiyle silme yapılabilir.
- Map'e hem anahtar-değer ikilisiyle hem de anahtar-eski değer-yeni değer üçlüsüyle, bir anahtara karşı gelen değer değiştirilebilir.
- Ayrıca BiFunction ile anahtar-değer geçerek tüm anahtarlar için yeni değer üretilebilir.

```
V           remove(Object key)

default boolean remove(Object key, Object value)

default V       replace(K key, V value)

default boolean replace(K key, V oldValue, V newValue)

default void    replaceAll(BiFunction<? super K, ? super V, ? extends V> function)
```

Map - IX



- `clear()` ile tüm anahtar ve değerler silinebilir.
- Map'te sorgulama metotları da vardır.

<code>void</code>	<code>clear()</code>
<code>boolean</code>	<code>containsKey(Object key)</code>
<code>boolean</code>	<code>containsValue(Object value)</code>
<code>boolean</code>	<code>isEmpty()</code>
<code>int</code>	<code>size()</code>

Map - X



- **compute** metotları **Function** ve **BiFunction** kullanarak yeni bir değer hesaplar ve anahtarla birlikte **Map**'te tutar.
- **compute ()** : Geçilen anahtar için yeni bir değer hesaplar, anahtar **Map**'te yoksa **NullPointerException** fırlatır.
- **computeIfAbsent ()** : Geçilen anahtar yoksa yeni bir değer hesaplar ve anahtar ile ilişkilendirerek tutar, varsa hiç bir şey yapmaz.
- Anahtar yoksa **compute ()** değil **computeIfAbsent ()** kullanılmalıdır.

```
default V      compute(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)
```

```
default V      computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction)
```



- **computeIfPresent()** : Geçilen anahtar için **null** olmayan bir değer varsa anahtar ve değeri kullanarak yeni bir değer hesaplar yoksa hiç bir şey yapmaz.

```
default V    computeIfPresent(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)
```

Map - XII



- Map, Java'nın 9. sürümünde eklenen 11 tane of isminde static üretici (factory) metoda da sahip olmuştur.

```
static <K, V> Map<K, V> of()  
  
static <K, V> Map<K, V> of (K k1, V v1)  
  
static <K, V> Map<K, V> of (K k1, V v1, K k2, V v2)  
  
static <K, V> Map<K, V> of (K k1, V v1, K k2, V v2, K k3, V v3)  
..."
```

- Ayrıca Map'e Java'nın 10. sürümünde static kopyalama metodu da eklenmiştir.

```
static <K, V> Map<K,V> copyOf(Map<? extends K,? extends V> map)
```

Map - XIII



- Bu metodlar değiştirilemez (unmodifiable) **Map** nesnesi oluşturur.
- Oluşturulan **Map** değiştirilmeye çalışıldığında `java.lang.UnsupportedOperationException` fırlatır.
- Ayrıca bu şekilde oluşturulan **Map**, `null` anahtar ya da değere izin vermez, `null` anahtar ya da değer eklenmeye çalışıldığında `NullPointerException` fırlatır.

MapExample.java

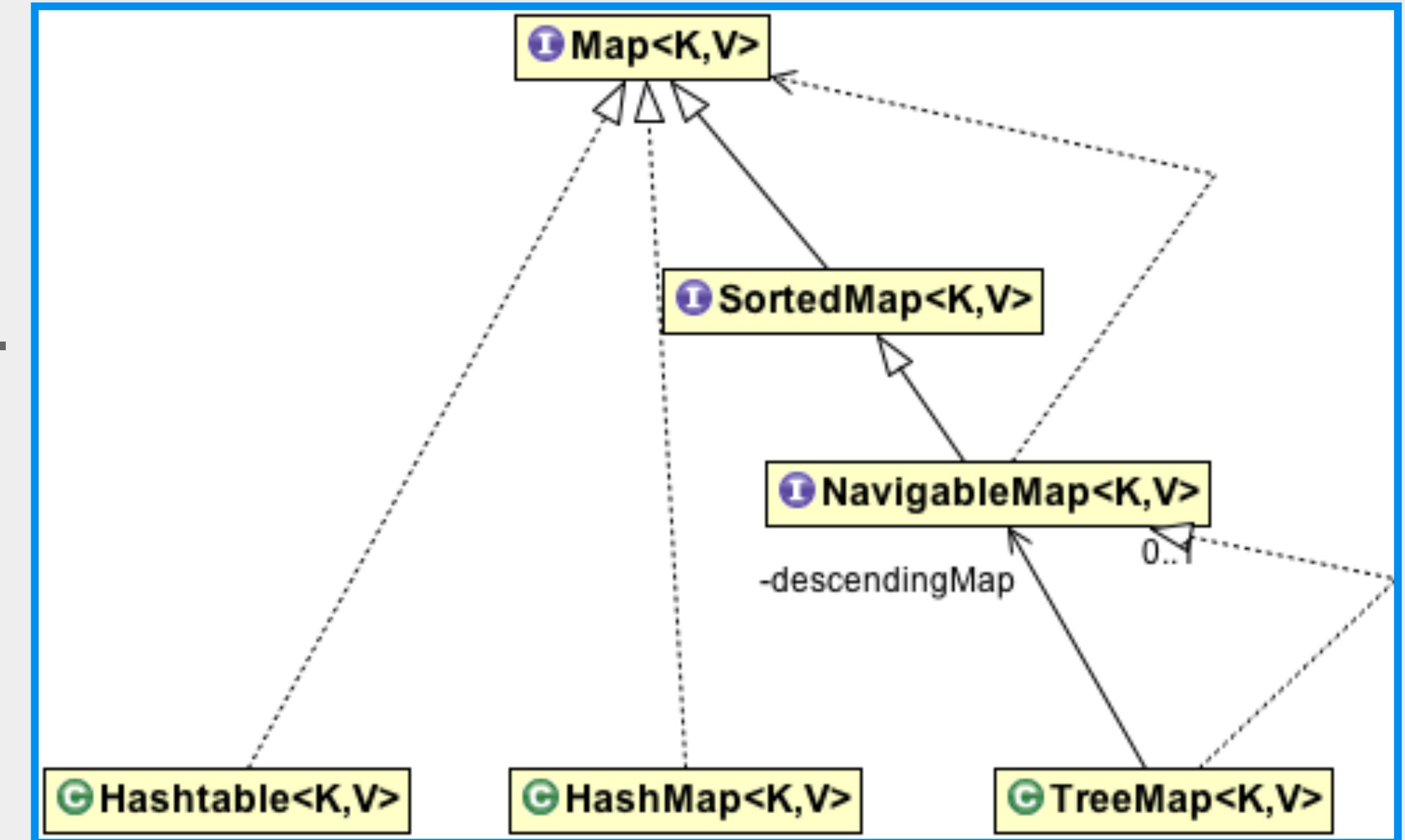


- org.javaturk.oofp.ch10.map.MapExample

Map - V



- Map arayüzünün en temel iki gerçeklestirmesi vardır:
 - **HashMap**, Map'i doğrudan gerçekleştirir ve en sık kullanılır.
 - **TreeMap** ise Map'in alt arayüzleri olan **SortedMap** ve **NavigableMap**'i de gerçekleştirir ve anahtarların sıralaması (sort) gerektiğinde kullanılır.





```
111 public void testSuccessfulLogin() {
112     // Given
113     // When
114     // Then
115     Customer loggedCustomer = service.login(properties, properPassword);
116     assertEquals(successfulCustomer, loggedCustomer);
117 }
118
119 @Test // Test 2
120 public void testSuccessfulLoginWithNoSuchCustomerException() {
121     // Given
122     // When
123     // Then
124     assertThrows(NoSuchCustomerNotFoundException.class, () -> {
125         Customer unfoundCustomer = service.login(unfoundCustomerToken, properPassword);
126     });
127 }
128
129 @Test // Test 3
130 public void testSuccessfulLoginWithCustomerLockedException() {
131     // Given
132     // When
133     // Then
134     assertThrows(CustomerLockedException.class, () -> {
135         Customer lockedCustomer = service.login(lockedCustomerToken, "password");
136     });
137 }
138
139 @Test // Test 4
140 public void testSuccessfulLoginWithCustomerAlreadyLoggedException() {
141     // Given
142     // When
143     // Then
144     assertThrows(CustomerAlreadyLoggedException.class, () -> {
145         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerToken, properPassword);
146     });
147 }
148
149 @Test // Test 5
150 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {
151     // Given
152     // When
153     // Then
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(shortToken, properPassword);
156     });
157 }
158
159 @Test // Test 6
160 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {
161     // Given
162     // When
163     // Then
164     assertThrows(ImproperCustomerCredentialsException.class, () -> {
165         Customer customer = service.login(propertiesToken, properPassword);
166     });
167 }
```

HashMap

HashMap - I



- **HashMap**, en standart ve en sık kullanılan **Map** gerçeklestirmesidir.
- **HashMap**, tamamen **Map** APIsine sahiptir.
- **HashMap**, anahtarlar ya da değerler için herhangi bir dizilim sözü vermez, var olan dizilim zaman içinde değişebilir.
- **HashMap**, anahtar ya da değer olarak **null** referanslara izin verir.
- **HashMap**, temel fonksiyonlarda (**put()**, **remove()**, **contains()**, ve **get()**) sabite yakın, **O(1)**, bir performans sağlar.

HashMap - II



- **HashMap**, 4 kurucuya sahiptir.
- Varsayılan kurucu başlangıç kapasitesi 16 ve doluluk oranı 0,75 olan boş bir **HashMap** oluşturur, arzu edilirse oluşturulurken başlangıç kapasitesi ve doluluk oranı verilebilir.
- Var olan Map kullanılarak yeni bir HashMap oluşturulabilir.

HashMap()

HashMap(int initialCapacity)

HashMap(int initialCapacity, float loadFactor)

HashMap(Map<? extends K, ? extends V> m

HashMapExample.java



- org.javaturk.oofp.ch10.map.HashMapExample

HashMap ve Hash Code - I



- **HashMap** gibi pek çok nesne, hash kodunu, nesneleri takip etmek ve hızlıca ulaşmak için kullanır.
- Örneğin **HashMap** sınıfının **put()** metodunda geçen anahtarın (key) hash değerini tutar.
- Anahtarlar tekildir, aynı anahtardan sadece bir tane vardır.
- Dolayısıyla **get()** metoduna bir anahtar geçildiğinde, önce hash kodunu bulur, sonra değere ulaşır ve $O(1)$ hızında değeri geri getirir.

HashMap ve Hash Code - II



- Eğer **HashMap**'de, **put ()** metoduna geçilen anahtardan zaten mevcutsa (bu da zaten anahtarın hash kodu ile anlaşılır), bu durumda eski değer geri döndürülüp yerine yeni değer konur.



```
111 public void testSuccessfulLogin() {
112     // Given
113     // When
114     // Then
115     Customer loggedCustomer = service.login(properties, properPassword);
116     assertEquals(successfulCustomer, loggedCustomer);
117 }
118
119 @Test // Test 2
120 public void testSuccessfulLoginWithNoSuchCustomerException() {
121     // Given
122     // When
123     // Then
124     assertThrows(NoSuchCustomerNotFoundException.class, () -> {
125         Customer unfoundCustomer = service.login(unfoundCustomerTkn, properPassword);
126     });
127 }
128
129 @Test // Test 3
130 public void testSuccessfulLoginWithCustomerLockedException() {
131     // Given
132     // When
133     // Then
134     assertThrows(CustomerLockedException.class, () -> {
135         Customer lockedCustomer = service.login(lockedCustomerTkn, "password");
136     });
137 }
138
139 @Test // Test 4
140 public void testSuccessfulLoginWithCustomerAlreadyLoggedException() {
141     // Given
142     // When
143     // Then
144     assertThrows(CustomerAlreadyLoggedException.class, () -> {
145         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTkn, properPassword);
146     });
147 }
148
149 @Test // Test 5
150 public void testSuccessfulLoginWithImproperCustomerCredentialsException() {
151     // Given
152     // When
153     // Then
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(shortTkn, properPassword);
156     });
157 }
158
159 @Test // Test 6
160 public void testSuccessfulLoginWithImproperCustomerCredentialsSpecificationException() {
161     // Given
162     // When
163     // Then
164     assertThrows(ImproperCustomerCredentialsSpecificationException.class, () -> {
165         Customer customer = service.login(propertiesTkn, properPassword);
166     });
167 }
```

TreeMap

TreeMap - I



- **TreeMap**, anahtarları sıralayan bir **Map** gerçekleştirmesidir.
- **TreeMap**, hem **SortedMap** hem de **NavigableMap**'tir.
- **TreeMap**, bir **Red-Black tree** olarak gerçekleştirilir.
- **TreeMap**, anahtar olarak **null**'a izin vermez ama değer olarak **null**'a izin verir.
- **TreeMap**, temel fonksiyonlarda (**put()**, **remove()**, **contains()**, ve **get()**), **O(lgn)** bir performans sağlar.

SortedMap Arayüzü



- **java.util.SortedMap**, anahtarları sıralayan Map'tir.
- Anahtarlar **Comparable** arayüzüünü gerçekleştirmelidir.
 - Ya da **SortedMap** gerçekleştirmesine **Comparator** geçilmelidir.

Comparator<? super K>	comparator()
Set<Map.Entry<K, V>>	entrySet()
K	firstKey()
SortedMap<K, V>	headMap(K toKey)
Set<K>	keySet()
K	lastKey()
SortedMap<K, V>	subMap(K fromKey, K toKey)
SortedMap<K, V>	tailMap(K fromKey)
Collection<V>	values()

NavigableMap



- `java.util.NavigableMap`, `SortedMap`'in in yapı içinde aramaya yapmaya ve aranana en yakın elemanları bulmayı sağlayan arayüzdür.



<code>Map.Entry<K,V></code>	<code>ceilingEntry(K key)</code>
<code>K</code>	<code>ceilingKey(K key)</code>
<code>NavigableSet<K></code>	<code>descendingKeySet()</code>
<code>NavigableMap<K,V></code>	<code>descendingMap()</code>
<code>Map.Entry<K,V></code>	<code>firstEntry()</code>
<code>Map.Entry<K,V></code>	<code>floorEntry(K key)</code>
<code>K</code>	<code>floorKey(K key)</code>
<code>SortedMap<K,V></code>	<code>headMap(K toKey)</code>
<code>NavigableMap<K,V></code>	<code>headMap(K toKey, boolean inclusive)</code>
<code>Map.Entry<K,V></code>	<code>higherEntry(K key)</code>
<code>K</code>	<code>higherKey(K key)</code>
<code>Map.Entry<K,V></code>	<code>lastEntry()</code>
<code>Map.Entry<K,V></code>	<code>lowerEntry(K key)</code>
<code>K</code>	<code>lowerKey(K key)</code>
<code>NavigableSet<K></code>	<code>navigableKeySet()</code>
<code>Map.Entry<K,V></code>	<code>pollFirstEntry()</code>
<code>Map.Entry<K,V></code>	<code>pollLastEntry()</code>
<code>NavigableMap<K,V></code>	<code>subMap(K fromKey, boolean fromInclusive,</code> <code> K toKey, boolean toInclusive)</code>
<code>SortedMap<K,V></code>	<code>subMap(K fromKey, K toKey)</code>
<code>SortedMap<K,V></code>	<code>tailMap(K fromKey)</code>
<code>NavigableMap<K,V></code>	<code>tailMap(K fromKey, boolean inclusive)</code>

NavigableAndSortedMapExample.java



- org.javaturk.oofp.ch10.map.
NavigableAndSortedMapExample

TreeMap - II



- **TreeMap**, 4 tane kurucuya sahiptir.
- Varsayılan kurucu başlangıç kapasitesi 16 ve doluluk oranı 0,75 olan, eklenecek elemanları tabi sıralamaya sahip boş bir **TreeMap** oluşturur.
- Eklenenek anahtarlar tabi dizilime sahip değilse **TreeMap**'in **Comparator** alan kurucusu kullanılmalıdır.

TreeMap()

TreeMap (Comparator<? super K> comparator)

TreeMap - III



- Ayrıca geçilen **Map** ve **SortedMap** ile de **TreeMap** oluşturulabilir.

`TreeMap (Map<? extends K, ? extends V> m)`

`TreeMap (SortedMap<K, ? extends V> m)`

TreeMapExample.java



- org.javaturk.oofp.ch10.map.TreeMapExample

HashMap mi mi TreeMap mi? - I



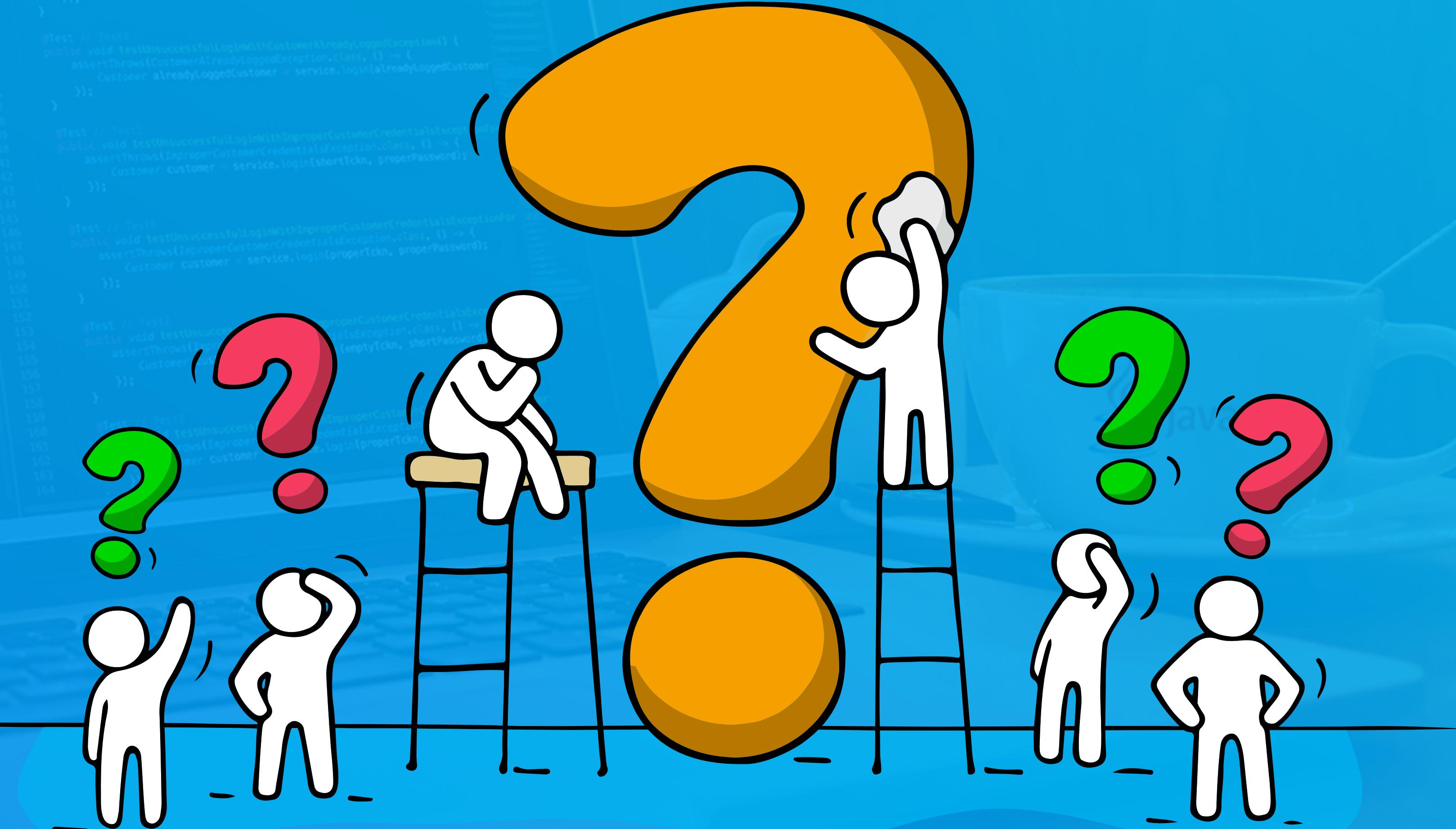
- **HashMap** ile **TreeMap** arasındaki temel fark, **TreeMap**'in anahtarları sıralamasıdır.
- Bu ise **HashMap** ve **TreeMap** arasında farklı davranışlar için farklı performanslara sebep olur.
 - **HashMap**, temel fonksiyonlarda (**put()**, **remove()**, **contains()**, ve **get()**) sabite yakın, **O(1)**, bir performans sağlarken **TreeMap**, **O(log n)** bir performans sağlar.
 - Bu fark **TreeMap**'ın sıralama yapan yapısından kaynaklanmaktadır.

HashMap mi mi TreeMap mi? - II



- Dolayısıyla sıralamaya ihtiyaç yoksa **HashMap** tercih edilmelidir.

Soru ve Cevap Zamani!





```
111 public void testSuccessfullLogin() {
112     // Given
113     // When
114     // Then
115     Customer loggedCustomer = service.login(properties, properPassword);
116     assertEquals(successfulCustomer, loggedCustomer);
117 }
118
119 @Test // Test 1
120 public void testSuccessfullLoginWithNoSuchCustomerException() {
121     // Given
122     // When
123     // Then
124     assertThrows(NoSuchCustomerNotFoundException.class, () -> {
125         Customer unfoundCustomer = service.login(unfoundCustomerToken, properPassword);
126     });
127 }
128
129 @Test // Test 2
130 public void testSuccessfullLoginWithCustomerLockedException() {
131     // Given
132     // When
133     // Then
134     assertThrows(CustomerLockedException.class, () -> {
135         Customer lockedCustomer = service.login(lockedCustomerToken, "password");
136     });
137 }
138
139 @Test // Test 3
140 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {
141     // Given
142     // When
143     // Then
144     assertThrows(CustomerAlreadyLoggedException.class, () -> {
145         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerToken, properPassword);
146     });
147 }
148
149 @Test // Test 4
150 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
151     // Given
152     // When
153     // Then
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(shortToken, properPassword);
156     });
157 }
158
159 @Test // Test 5
160 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
161     // Given
162     // When
163     // Then
164     assertThrows(ImproperCustomerCredentialsException.class, () -> {
165         Customer customer = service.login(propertiesToken, properPassword);
166     });
167 }
```

Hashtable

Hashtable - I



- `java.util.Hashtable` de ilişkilendiren (associative) bir torbadır.
- Java'nın ilk günlerinden kalan bir sınıfı, API'si Java 1.2 ile tekrar düzenlenmiş ve `Map`'i gerçekleştirecek hale getirilmiştir.
- Dolayısıyla şu anda hem eski metodlara hem de `Map` arayüzüne sahiptir.
- Örneğin, `Enumeration<K> keys()` ve `Enumeration<V> elements()` metodları vardır.

Hashtable - II



- **Hashtable** diğer **Map** nesnelerinin tersine, **synchronized**'dır.
- Dolayısıyla çoklu kanallı (multi-threaded) ortam söz konusu değilse **Hashtable** kullanılmamalıdır çünkü gereksiz performans kaybı yaşatır.
- Böyle durumlarda **HashMap** ya da **TreeMap** tercih edilmelidir.

HashtableExample.java



- org.javaturk.oofp.ch10.map.HashtableExample

Dönüşümler



```
115     assertEquals(successfulCustomer, loggedCustomer);
116 }
117
118 @Test // Test2
119 public void testUnsuccessfulLoginWithNoSuchCustomerException() {
120     assertThrows(NoSuchCustomerFoundException.class, () -> {
121         Customer unfoundCustomer = service.login(unfoundCustomerTckn, proper
122     });
123 }
124
125 @Test // Test3
126 public void testUnsuccessfulLoginWithCustomerLockedException() {
127     assertThrows(CustomerLockedException.class, () -> {
128         Customer lockedCustomer = service.login_lockedCustomerTckn, "qwerty1
129     });
130 }
131
132 @Test // Test4
133 public void testUnsuccessfulLoginWithCustomerAlreadyLoggedException() {
134     assertThrows(CustomerAlreadyLoggedException.class, () -> {
135         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomer
136     });
137 }
138
139 @Test // Test5
140 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionFor
141     assertThrows(ImproperCustomerCredentialsException.class, () -> {
142         Customer customer = service.login(shortTckn, properPassword);
143     });
144 }
145
146 @Test // Test6
147 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionFor
148     assertThrows(ImproperCustomerCredentialsException.class, () -> {
149         Customer customer = service.login(properTckn, properPassword);
150     });
151 }
152
153 @Test // Test7
154 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionFor
155     assertThrows(ImproperCustomerCredentialsException.class, () -> {
156         Customer customer = service.login(properTckn, shortPassword);
157     });
158 }
159
160 @Test // Test8
161 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionFor
162     assertThrows(ImproperCustomerCredentialsException.class, () -> {
163         Customer customer = service.login(shortTckn, properTckn, shortPassword);
164     });
165 }
```

Torbalar Arası Dönüşümler - I



- Torbalar arasında dönüşümler yapmayı sağlayan metodlar, torbaların API'lerinde mevcuttur.
- Bunların bazıları kuruculardır:

`ArrayList(Collection<? extends E> c)`

`LinkedList(Collection<? extends E> c)`

`HashSet(Collection<? extends E> c)`

`TreeSet(Collection<? extends E> c)`

`HashMap(Map<? extends K, ? extends V> m)`

`TreeMap(Map<? extends K, ? extends V> m)`

`TreeMap(SortedMap<K, ? extends V> m)`

Torbalar Arası Dönüşümler - II



- Map'ten üç tane farklı Collection nesnesi alınabilir:

<code>Set<K></code>	<code>keySet()</code>
<code>Collection<V></code>	<code>values()</code>
<code>Set<Map.Entry<K,V>></code>	<code>entrySet()</code>

Torbalar Arası Dönüşümler - III



- Bütün **Collection** nesneleri diziye dönüştürülebilir:

`Object[]`

`toArray()`

`<T> T[]`

`toArray(T[] a)`

`default <T> T[] toArray(IntFunction<T[]> generator)`

Torbalar Arası Dönüşümler - IV

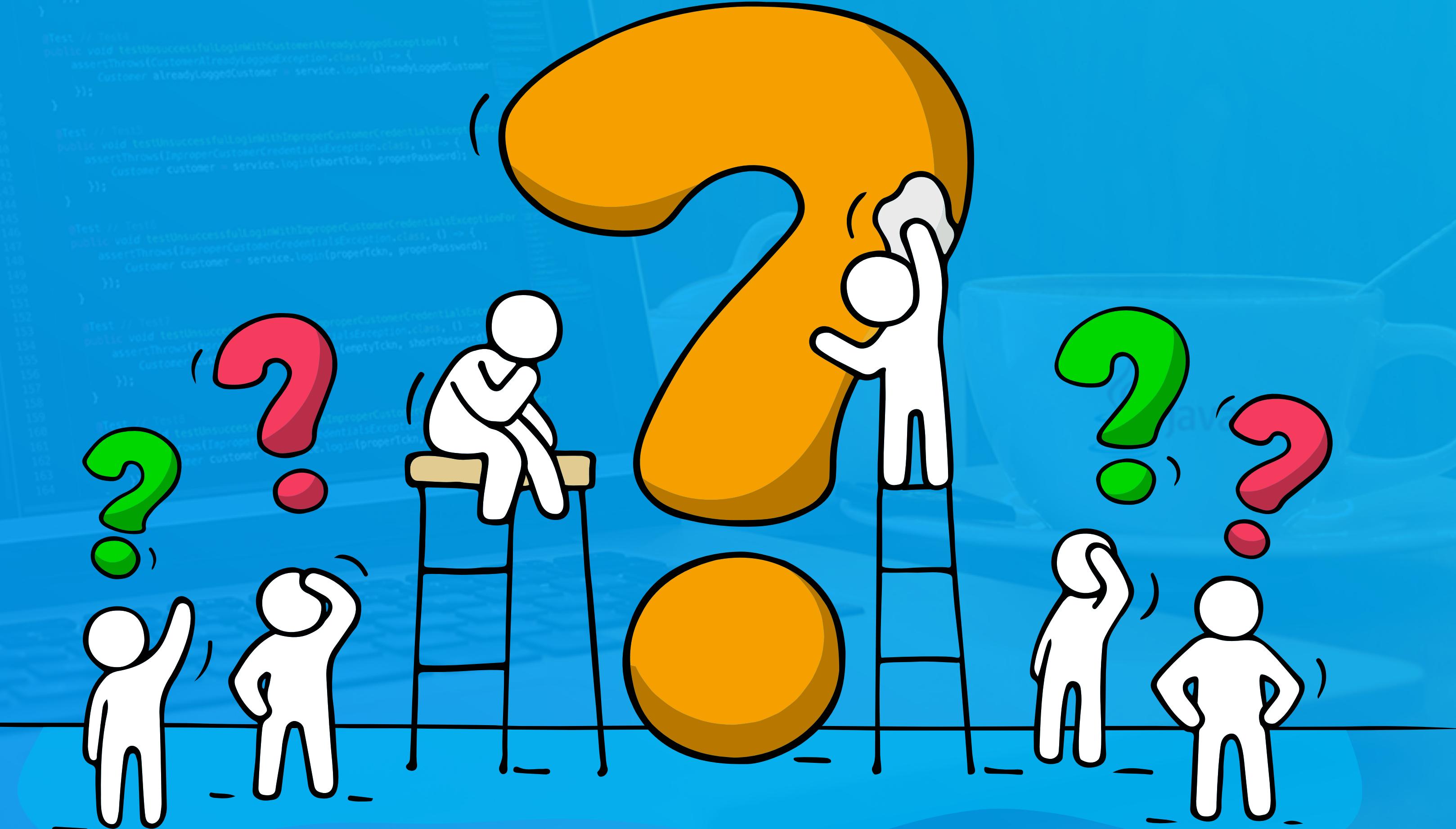


- Ayrıca `java.util.Arrays` sınıfındaki `asList()` metodu da diziden, uzunluğu değişmeyen (fixed-size) `List` oluşturur.

```
List<T> asList(T... a))
```

- Dizi ile diziden oluşturulan `List` birbirlerinin değişikliklerinden haberdar olurlar.
- `List`, uzunluğunu değiştirecek işlemlerde `java.lang.UnsupportedOperationException` fırlatır.

Soru ve Cevap Zamani!



Algoritmalar



```
115     assertEquals(successfulCustomer, loggedCustomer);
116 }
117
118 @Test // Test2
119 public void testUnsuccessfulLoginWithNoSuchCustomerException() {
120     assertThrows(NoSuchCustomerFoundException.class, () -> {
121         Customer unfoundCustomer = service.login(unfoundCustomerTckn, properPassword);
122     });
123 }
124
125 @Test // Test3
126 public void testUnsuccessfulLoginWithCustomerLockedException() {
127     assertThrows(CustomerLockedException.class, () -> {
128         Customer lockedCustomer = service.login_lockedCustomerTckn, "qwerty123");
129     });
130 }
131
132 @Test // Test4
133 public void testUnsuccessfulLoginWithCustomerAlreadyLoggedException() {
134     assertThrows(CustomerAlreadyLoggedException.class, () -> {
135         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTckn,
136         "qwerty123");
137     });
138 }
139
140 @Test // Test5
141 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForUsername() {
142     assertThrows(ImproperCustomerCredentialsException.class, () -> {
143         Customer customer = service.login(shortTckn, properPassword);
144     });
145 }
146
147 @Test // Test6
148 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForPassword() {
149     assertThrows(ImproperCustomerCredentialsException.class, () -> {
150         Customer customer = service.login(properTckn, shortPassword);
151     });
152 }
153
154 @Test // Test7
155 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForBoth() {
156     assertThrows(ImproperCustomerCredentialsException.class, () -> {
157         Customer customer = service.login(propertckn, shortPassword);
158     });
159 }
160
161 @Test // Test8
162 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForBoth() {
163     assertThrows(ImproperCustomerCredentialsException.class, () -> {
164         Customer customer = service.login(propertckn, shortPassword);
165     });
166 }
```



- Java'da dizilerle (array) ilgili algoritmalar genel olarak `java.util.Arrays` isimli araç (utility) sınıfında ve statik metodlar olarak bulunur.
- Java'da torbalarla ilgili algoritmalar ise genel olarak `java.util.Collections` isimli araç (utility) sınıfında ve statik metodlar olarak bulunur.
- `Collections` sınıfındaki algortimalar çok şekillidir (polymorphic),
 - Metotlar algoritmanın tabiatına göre bazen `Collection` bazen de `List` alır.

Collections Sınıfı Algoritmaları



- Metotlar, geçilen `Collections` nesnesi `null` olduğunda `NullPointerException` fırlatır.
- Algoritmalar arasında, `unmodifiableSet()`, `unmodifiableList()` gibi var olan torbanın değiştiremeyen (`unmodifiable`) şeklini üreten toplam 8 metot vardır.

Vector-Hashtable Yerine List-Map - I



- Java'nın ilk sürümünden gelen **Vector** ve **Hashtable** sınıfları **synchronized** metotlara sahip olduğundan, çok kanallı (multi threaded) olmayan yani tek bir kanalın (single threaded) olduğu ortamlarda kullanılmaması gereği, bunlar yerine **List** ve **Map** gerçekleştirmelerinin tercih edilmesi gereği söylendi.
- Peki çok kanallı ortamlarda **Vector** ve **Hashtable** kullanılmalı mı?

Vector-Hashtable Yerine List-Map - II



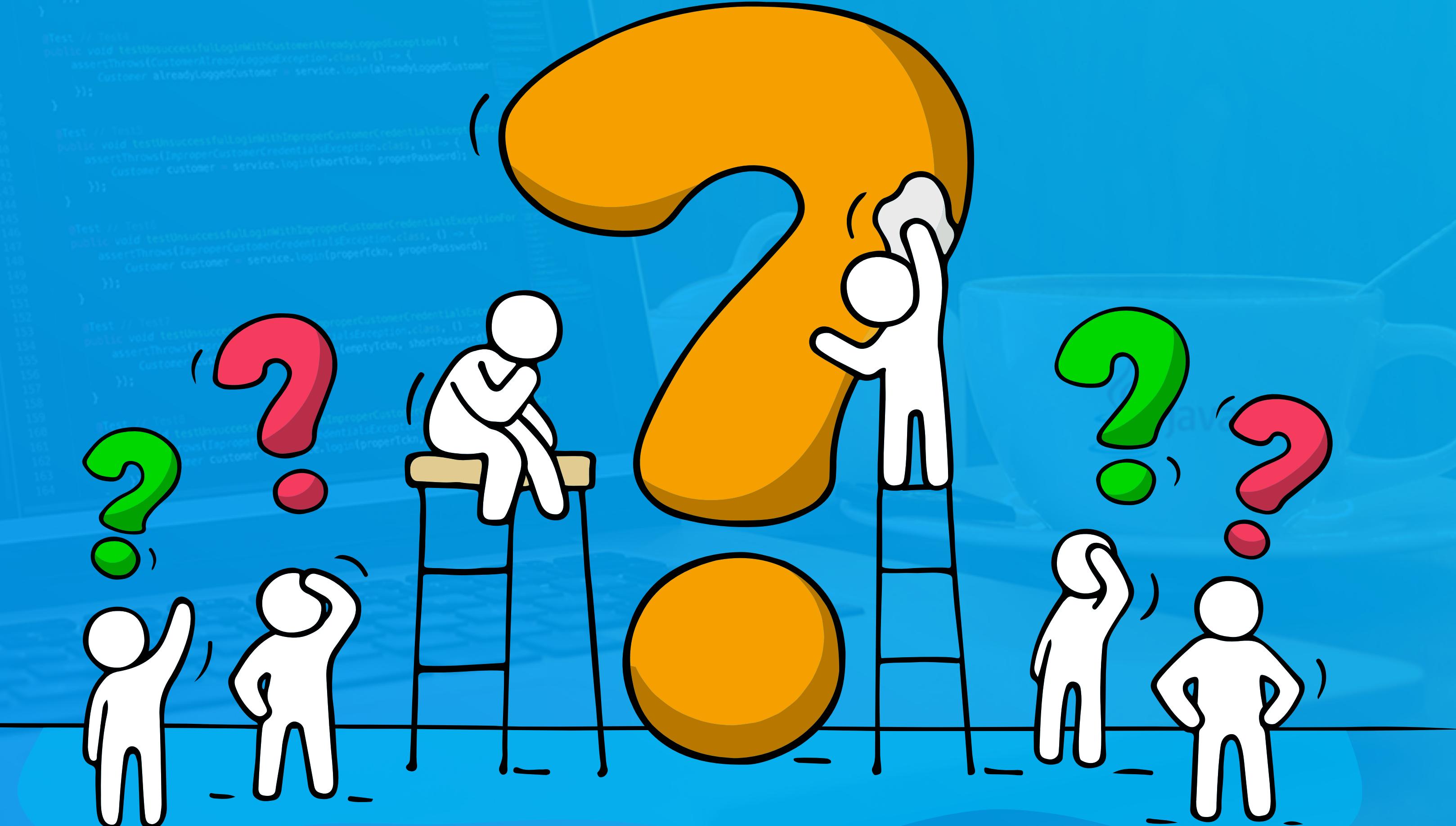
- Çok kanallı ortamlarda da **Set**, **List** ve **Map**'in **thread-safe** halleri **Collections** nesnesi üzerindeki metodlarla alınabilir:
 - **synchronizedSet()** , **synchronizedList()** ,
synchronizedMap() vb. 6 metod, geçilen torbaları **synchronized** yani **thread-safe** yapar.

Algoritma Örnekleri



- org.javaturk.oofp.ch10.algorithms paketi.

Soru ve Cevap Zamani!



Diğer Torba Kütüphaneleri



```
115     assertEquals(successfulCustomer, loggedCustomer);
116 }
117
118 @Test // Test2
119 public void testUnsuccessfulLoginWithNoSuchCustomerException() {
120     assertThrows(NoSuchCustomerFoundException.class, () -> {
121         Customer unfoundCustomer = service.login(unfoundCustomerTckn, properties.getUsername(), properties.getPassword());
122     });
123 }
124
125 @Test // Test3
126 public void testUnsuccessfulLoginWithCustomerLockedException() {
127     assertThrows(CustomerLockedException.class, () -> {
128         Customer lockedCustomer = service.login_lockedCustomerTckn, "qwerty1");
129     });
130 }
131
132 @Test // Test4
133 public void testUnsuccessfulLoginWithCustomerAlreadyLoggedException() {
134     assertThrows(CustomerAlreadyLoggedException.class, () -> {
135         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTckn, "qwerty1");
136     });
137 }
138
139 @Test // Test5
140 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForEmptyTckn() {
141     assertThrows(ImproperCustomerCredentialsException.class, () -> {
142         Customer customer = service.login(emptyTckn, shortPassword);
143     });
144 }
145
146 @Test // Test6
147 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForShortPassword() {
148     assertThrows(ImproperCustomerCredentialsException.class, () -> {
149         Customer customer = service.login(emptyTckn, shortPassword);
150     });
151 }
152
153 @Test // Test7
154 public void testUnsuccessfulLoginWithImproperCustomerCredentialsExceptionForEmptyUsername() {
155     assertThrows(ImproperCustomerCredentialsException.class, () -> {
156         Customer customer = service.login(emptyTckn, properties.getUsername());
157     });
158 }
```

Diğer Torba Kütüphaneleri



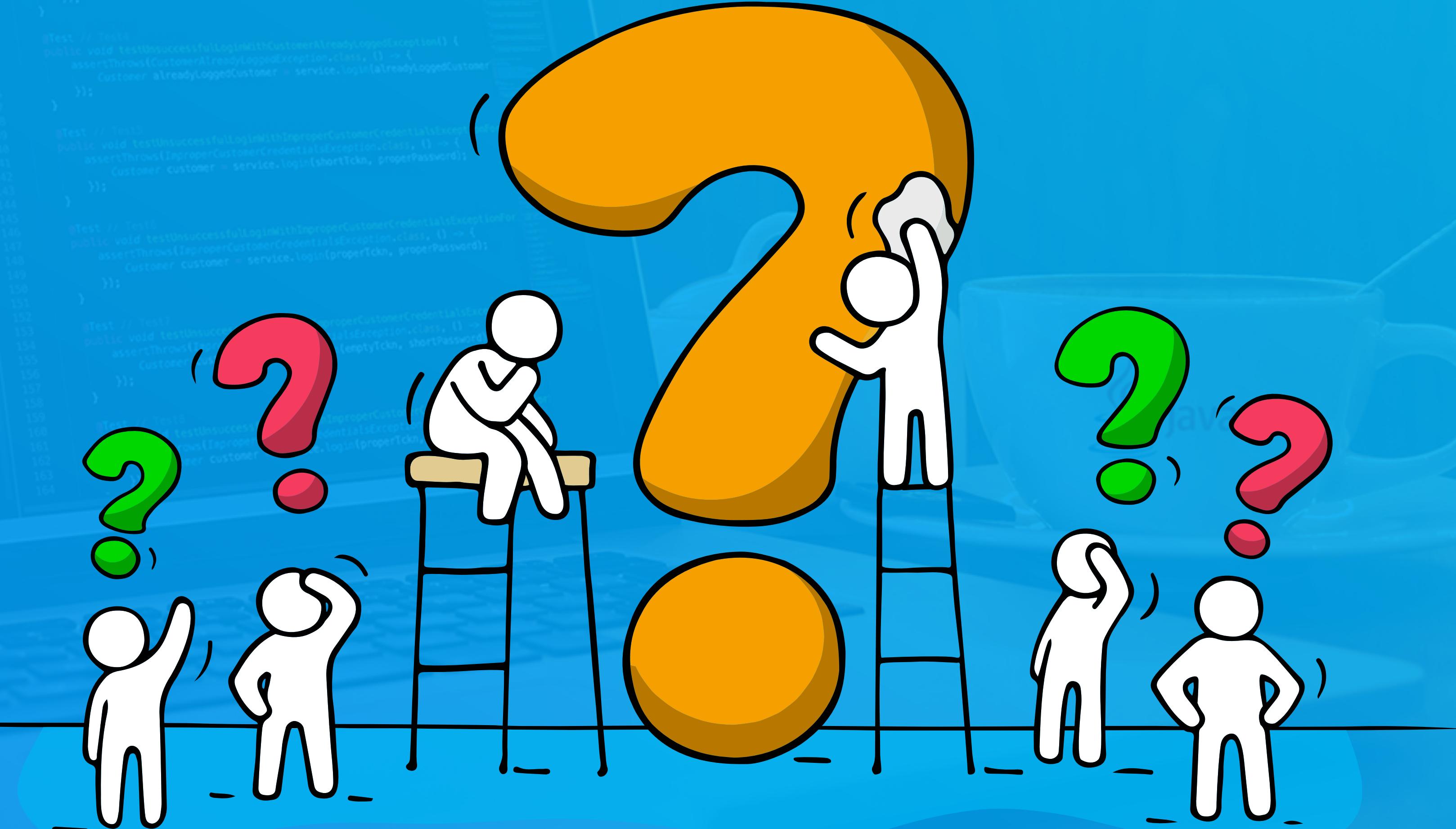
- Java'nın torbaları sorununu çözmez, farklı yapı ve yetkinlikte torbalara ihtiyacınız olursa pek çoğu ücretsiz hatta açık kaynak kodlu kütüphaneleri kullanabilirsiniz.
- **Apache Commons Collections**
 - <https://commons.apache.org/proper/commons-collections/>
- **Google Guava** da bir torba kütüphanesine sahiptir.
 - <https://github.com/google/guava>

Apache Commons Örnekleri



- org.javaturk.oofp.ch10.commons paketi.

Soru ve Cevap Zamani!





Ödevler

```
112     public void testSuccessfulLogin() {
113         throws NoSuchCustomerFoundException, CustomerLockedException, Customer
114             ImproperCustomerCredentialsException, MaxNumberOffailedLoginsException
115         Customer loggedCustomer = service.login(properticks, properPassword);
116         assertEquals(successfulCustomer, loggedCustomer);
117     }
118
119     @Test // Test1
120     public void testUnsuccessfulLoginWithNoSuchCustomerException() {
121         assertThrows(NoSuchCustomerFoundException.class, () -> {
122             Customer unfoundCustomer = service.login(unfoundCustomerTckn, proper
123         });
124     }
125
126     @Test // Test2
127     public void testUnsuccessfulLoginWithCustomerLockedException() {
128         assertThrows(CustomerLockedException.class, () -> {
129             Customer lockedCustomer = service.login(lockedCustomerTckn, "password");
130         });
131     }
132
133     @Test // Test3
134     public void testUnsuccessfulLoginWithCustomerAlreadyLoggedInException() {
135         assertThrows(CustomerAlreadyLoggedInException.class, () -> {
136             Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomer
137         });
138     }
139
140     @Test // Test4
141     public void testUnsuccessfulLoginWithInproperCustomerCredentialsExceptionFor
142         assertThrows(InproperCustomerCredentialsException.class, () -> {
143             Customer customer = service.login(shortTckn, properPassword);
144         });
145
146     @Test // Test5
147     public void testUnsuccessfulLoginWithInproperCustomerCredentialsExceptionFor
148         assertThrows(InproperCustomerCredentialsException.class, () -> {
149             Customer customer = service.login(properticks, properPassword);
150         });
151
152     @Test // Test6
153     public void testUnsuccessfulLoginWithInproperCustomerCredentialsExceptionFor
154         assertThrows(InproperCustomerCredentialsException.class, () -> {
155             Customer customer = service.login(emptyTckn, shortPassword);
156         });
157     }
158 }
```



1. Daha önce oluşturduğunuz üniversite örneğindeki **Student** hiyerarşisini ele alın ve öğrencileri önce **Comparable** sonra da **Comparator** arayüzü ile sıralanabilir yapın.
2. Daha önce oluşturduğunuz karmaşık sayılar sınıfının nesnelerini önce **Comparable** sonra da **Comparator** arayüzü ile sıralanabilir yapın.



3. İsim, soy isim ile birden fazla adres ve telefona sahip olan kişileri düşünün. Bu kişileri soy isimlerinin baş harflerine göre sınıflandırılmış şekilde tutacak bir yapı kurgulayın öyle ki

- Kişiler tekildir,
- İsim, soy isim, adres ve telefon üzerinden etkin arama işlemleri yapılabilmelidir,
- Listeleme ve sıralama yetkinlikleri olmalıdır.

Bölüm Sonu

Soru ve Cevap Zamani!

