



# Java ile Nesne Merkezli ve Fonksiyonel Programlama

11. Bölüm: *Genel Programlama (Generics)*



Eğitmen:

**Akın Kaldıroğlu**

Çevik Yazılım Geliştirme ve Java Uzmanı

# Konular



- **java.lang.Object'e Genel Yapılar**
- **Genel Yapılar**
  - Tip Değişkenleri
  - Joker, Alt ve Üst Sınır Tip Kullanımı
- **Genel Tipli Sınıf Tanımlama**
- **Ödevler**

# java.lang.Object'e Genel Yapılar

# Generic Ne demektir?



- **Generic:** relating to or characteristic of a whole group or class : general <“Romantic comedy” is the generic term for such films.>
- **Generic** İngilizce’de bir gruptaki elemanlara has olan, onlarda ortak olan demektir.
- Cinse ait olan, cins ile ilgili olan, tikele özel olmayan demektir.
  - Genel, kapsamlı gibi kişiliklara sahiptir.
- Bu derste generic ile genel, genel tip terimleri, tipe has ile de tipe genel terimleri birbiri yerine geçecek şekilde kullanılmıştır.

# java.lang.Object Referansları - I



- Java'da en temel **genel programlama (generic programming)**, **java.lang.Object** sınıfıyla berhasilır.
- Java'da **Object** sınıfı, her sınıfın super sınıfı olduğundan, her nesneyi **Object** sınıfından bir referans ile gösterebiliriz:

```
Object o = new String("Java");
o = new Customer();
```

- Benzer şekilde her nesneyi, **Object** sınıfı bekleyen bir metoda geçebilir, **Object** sınıfı tipinde bir referans döndüren bir metottan geriye döndürebiliriz:

```
Object firstName = o.getFirstName();
```

# java.lang.Object Referansları - II



- Java API'sindeki sınıflar da benzer şekilde genelde `java.lang.Object` tipinden nesnelerle çalışırlar:
  - `Object` sınıfı üzerindeki `clone()` metodu `Object` döndürür,
  - Torbalar ve üzerlerindeki iteratörler yine `Object` alıp verirler:
    - `add(Object o)`, `remove(Object o)` metodları `Object` alırlar.
    - `java.util.Iterator` üzerindeki `next()` metodu `Object` döndürür.

# java.lang.Object Referansları - III



- Bir API tasarlarken de sıkılıkla **Object** tipinden nesne alıp verecek şekilde tasarlarız:

```
public void send(Object o);  
public Object receive()
```

# java.lang.Object Referansları - IV



- Bu durumun yani her yerde **Object** nesnesi kullanmanın bazı problemleri söz konusudur:
  - Alt tipten nesnelere has yapılar oluşturmak yani **Object** nesnesi yerine alt sınıflardan nesneler kullanmak gerekiğinde:
- Sürekli **Object** nesnesi kullanmak alçaltma (downcast) ihtiyacı doğurur.

```
List students = new ArrayList();
students.add(new Student());
students.add(3); // Should not allow this
students.add(new Date()); // Should not allow this
```

```
Student s = (Student) channel.receive();
Student s = (Student) students.get(i);
```

# RawTypeList.java



- org.javaturk.oofp.ch11.problem.RowTypeList

# Genel Yapılar (Generics)

# Genel Yapılar (Generics)



- Java'ya 1.5 ile birlikte generics yapısı gelmiştir.
- Muhtemelen Java'nın tarihinde, çehresini en fazla değiştiren özellik olmuştur.
- Genel yapılar (generics) ile `java.lang.Object` nesnesi yerine farklı bir tipe has/genel yapılar oluşturmak mümkündür.
- Bu bölümde genel yapılar, daha basitten karmaşığa doğru giderek, iki farklı şekliyle ele alınacaktır:
  - Önce var olan genel yapıların kullanımı,
  - Sonra ise genel yapıların oluşturulması.

# Genel Yapıların Kullanımı - I



- Genel yapılar (generics) uygun bir şekilde tanımlanmış bir yapıyı kullanmak için <> (diamond) operatörü kullanılır.
- Referans tanımlanırken <> operatörü içinde genel tip belirtilir.
  - Belirtilen genel tipe, tip parametresi (type parameter) de denir.

```
List<Student> students = new ArrayList<Student>();
```

- Java SE 1.7 ile birlikte kurucu çağrıları yapılırken sadece <> kullanmak yeterli hale getirilmiştir

```
List<Student> students = new ArrayList<>();
```

# Genel Yapıların Kullanımı - II



- Genel tip kullanılarak oluşturulan bir nesne, sadece o genel tipten olan nesnelerle çalışabilir.
- Örneğin, Java'nın torbalarının nesnelerini oluştururken genel tip kullanıldığında `<>` operatörü içinde belirtilen tip ve alt sınıfların dışında bir nesneyi o torbaya eklemeye çalışmak bir derleme hatasıdır.

```
List<Student> students = new ArrayList<>();
students.add(new Student());
students.add(new MasterStudent());
students.add(new PhDStudent());

students.add(new Person()); // Compiler error!
students.add(3);           // Compiler error!
students.add(new Date());  // Compiler error!
```

# GenericList.java



- org.javaturk.oofp.ch11.problem.GenericList

# İlkel Tipli Generic Kullanımı



- Java'da ilkel tipli generic kullanılamaz, generic tip muhakkak nesne olmalıdır.
- Bu yüzden ilkel tipler için onların nesnelerini temsil eden `java.lang` paketindeki sarmalayan (wrapper) tipleri kullanılabilir.
- İlkel tipler ile sarmalayan tipler arasındaki dönüşümler otomatiktir.

```
List<Integer> ints = new ArrayList<>();  
ints.add(3);           // Auto-boxing  
ints.add(new Integer(5));  
int i = ints.get(1);  // i is 5! Auto-unboxing  
short s = 11;  
ints.add(new Short(s)); // Compiler error!  
ints.add(new Long(3L)); // Compiler error!
```

# PrimitiveGenerics.java



- org.javaturk.oofp.ch11.generics.PrimitiveGenerics

# GenericMap.java



- org.javaturk.oofp.ch11.generics.GenericMap



```
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, properPassword);
121     });
122 }
123
124 @Test // Test 2
125 public void testSuccessfullLoginWithCustomerLockedException() {
126     assertThrows(CustomerLockedException.class, () -> {
127         Customer lockedCustomer = service.login(lockedCustomerTkn, properPassword);
128     });
129 }
130
131 @Test // Test 3
132 public void testSuccessfullLoginWithCustomerAlreadyLoggedException() {
133     assertThrows(CustomerAlreadyLoggedException.class, () -> {
134         Customer alreadyLoggedCustomer = service.login(alreadyLoggedCustomerTkn, properPassword);
135     });
136 }
137
138 @Test // Test 4
139 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
140     assertThrows(ImproperCustomerCredentialsException.class, () -> {
141         Customer customer = service.login(shortTkn, properPassword);
142     });
143 }
144
145 @Test // Test 5
146 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
147     assertThrows(ImproperCustomerCredentialsException.class, () -> {
148         Customer customer = service.login(propertiesTkn, properPassword);
149     });
150 }
151
152 @Test // Test 6
153 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
154     assertThrows(ImproperCustomerCredentialsException.class, () -> {
155         Customer customer = service.login(properTkn, shortPassword);
156     });
157 }
158
159 @Test // Test 7
160 public void testSuccessfullLoginWithImproperCustomerCredentialsException() {
161     assertThrows(ImproperCustomerCredentialsException.class, () -> {
162         Customer customer = service.login(properTkn, properPassword);
163     });
164 }
```

# Tip Değişkenleri

# Tip Değişkenleri/Parametreleri - I



- Genel yapıya uygun olarak tanımlanacak tiplerde de `<>` operatörü kullanılır.
- Bu amaçla genel tipi göstermek için **E, T, K, V vb. genel tip değişkenleri (type variable)** kullanılır.
  - Tip değişkenlerinin büyük harflerle gösterilmesi bir gelenektir.
  - Dolayısıyla eskiden `java.lang.Object` ile çalışan yapılar genericlerin Java'ya girmesiyle birlikte **E, T, K, V vb. harflerle sembolleştirilerek gösterilen tip değişkenleri** ile çalışmaya başlamıştır.

# Tip Parametreleri - I



- Bu sebeple genel tip kabul eden nesnelerin APllerinde `java.lang.Object` yerine `E`, `T`, `K`, `V` tip değişkenleri bulunmaktadır.

# Tip Değişkenleri (Parametreleri)



- Generic kullanımı API'leri de etkilemiştir.
- Örneğin Java API'sinde `java.lang.Object` yerine farklı tipleri genel tip olarak kabul eden yapıların API'leri de genel tip alabilecek şekilde belirtilmektedir.
- Dolayısıyla `E`, `T`, `K`, `V` vb. genel tip değişkenleri (type variable) Java API'sinde sıkılıkla görülür.

# java.util.List API'si



- `java.util.List`'in API'sinin Java 1.5 öncesindeki ve sonrasındaki hali aşağıdadır:

boolean	<code>add(Object o)</code> // Java SE 1.4 hali
boolean	<code>add(E e)</code> // Java SE 1.5 hali
void	<code>add(int index, Object element)</code>
void	<code>add(int index, E element)</code>
Object	<code>get(int index)</code>
E	<code>get(int index)</code>
Object	<code>remove(int index)</code>
E	<code>remove(int index)</code>
Object	<code>set(int index, Object element)</code>
E	<code>set(int index, E element)</code>
List	<code>subList(int fromIndex, int toIndex)</code>
List<E>	<code>subList(int fromIndex, int toIndex)</code>

# Generic Kullanımının API'ye Etkisi



- List arayüzünün Java SE 1.4 ve 1.8 API'leri,
- List tanımındaki farka dikkat edin: `List` ve `List<E>`

`java.util`

## Interface List

### All Superinterfaces:

[Collection](#)

### All Known Implementing Classes:

[AbstractList](#), [ArrayList](#), [LinkedList](#), [Vector](#)

---

`public interface List`

`extends Collection`

`java.util`

## Interface List<E>

### Type Parameters:

E - the type of elements in this list

### All Superinterfaces:

`Collection<E>, Iterable<E>`

### All Known Implementing Classes:

`AbstractList`, `AbstractSequentialList`, `ArrayList`, `AttributeList`, `CopyOnWriteArrayList`, `LinkedList`, `RoleList`, `RoleUnresolvedList`, `Stack`, `Vector`

---

`public interface List<E>`

# GenericList.java



- org.javaturk.oofp.ch11.problem.GenericList

# Java SE 1.4 API



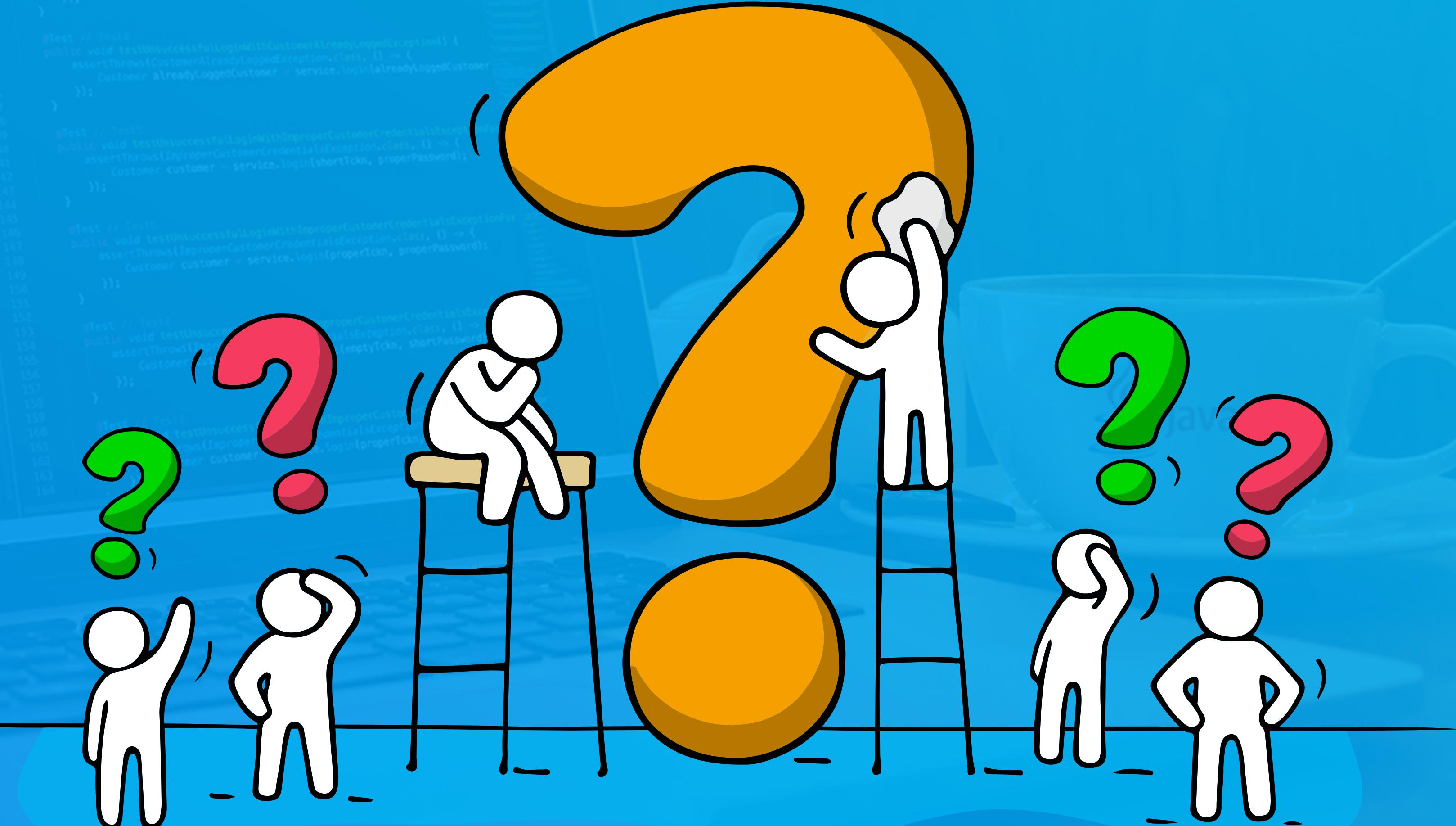
- Java SE 1.4 API için: <https://docs.oracle.com/javase/1.4.2/docs/api/>

# School Paketi



- org.javaturk.oofp.ch11.School paketi.
  - EmployeeComparator
  - TeacherComparator

# Soru ve Cevap Zamani!





```
111 public void testSuccessfullLogin() {  
112     assertEquals(CustomerNotFoundException.class, customer.  
113         login("customer1", "password1").getClass());  
114     assertEquals("Customer logged in successfully.", customer.  
115         login("customer1", "password1").getSuccessMessage());  
116 }  
117  
118 @Test // Test 2  
119 public void testSuccessfullLoginWithUnfoundCustomer() {  
120     assertEquals(UnfoundCustomerException.class, () -> {  
121         customer.unfoundCustomer = service.login("customer1", "password1");  
122     }).getClass();  
123 }  
124  
125 @Test // Test 3  
126 public void testSuccessfullLoginWithLockedCustomer() {  
127     assertEquals(CustomerLockedException.class, () -> {  
128         customer.lockedCustomer = service.login("lockedCustomer1", "password1");  
129     }).getClass();  
130 }  
131  
132 @Test // Test 4  
133 public void testSuccessfullLoginWithAlreadyLoggedCustomer() {  
134     assertEquals(AlreadyLoggedCustomerException.class, () -> {  
135         customer.alreadyLoggedCustomer = service.login("alreadyLoggedCustomer1",  
136             "password1");  
137     }).getClass();  
138 }  
139  
140 @Test // Test 5  
141 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
142     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
143         customer = service.login("shortToken", "properPassword");  
144     }).getClass();  
145 }  
146  
147 @Test // Test 6  
148 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
149     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
150         customer = service.login("properToken", "shortPassword");  
151     }).getClass();  
152 }  
153  
154 @Test // Test 7  
155 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
156     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
157         customer = service.login("properToken", "properPassword");  
158     }).getClass();  
159 }  
160  
161 }
```

# Joker, Alt ve Üst Sınır Tip Kullanımı

# Genel Sınıflarda Alt ve Üst Tipler - I



- Genel tiplerin kullanımında miras önemli bir yer kaplar.
- Örneğin, sadece genel bir tip değil, bir şemsiye tip (ya da üst tip) veya alt tip nasıl ifade edilir?
- Örneğin, belli bir tipe genel **Collection** nesnesine, arayüzündeki **addAll()** metoduyla başka bir **Collection**'daki nesneler nasıl eklenir?
- **Collection<Employee>** torbasına sadece **Employee** ve alt türlerine genel bir **Collection** torbasındaki nesnelerin eklenebileceği nasıl ifade edilir?

# Genel Sınıflarda Alt ve Üst Tipler - II



- Bu tür sorular genel tipler kullanımının miras ile ilişkisi bağlamına ele alınır.
- Bu amaçla alt ve üst sınır tiplerin kullanımı ele alınacak.

# Geneller ve Alt Tipler - I



- Bir tipe has genel yapı ile o tipin bir alt tipine has genel yapı arasında bir is-a ilişkisi var mıdır?
- Ya da örneğin `List<String>`, `List<Object>`'nin bir alt tipi midir?
- Yani aşağıdaki atama yapılabilir mi?

```
List<String> strings = new ArrayList<>();
List<Object> objects = strings; // Can we do this?
// If we can
strings.add(new String("java"));
objects.add(new Date());
Object o = objects.get(0); // This is ok, it is a String
String s = strings.get(1); // But it is not a String!
```

# Geneller ve Alt Tipler – II



- Dolayısıyla, aralarında is-a ilişkisi olan iki tipe has oluşturulan genel tipli nesneler birbirlerine atanamazlar!
- Dolayısıyla `List<Manager>`, `List<Employee>`'nin bir alt tipi değildir!
- Bu şu anlama gelir: `List<Employee>` beklenen yere `List<Manager>` geçilemez.

```
List<Employee> employees = new ArrayList<>();
makeUpTeam(employees);

List<Manager> managers = new ArrayList<>();
makeUpTeam(managers); // Can't do that!

static void makeUpTeam(List<Employee> employees){}
```

# GenericParameters.java



- org.javaturk.oofp.ch11.generics.GenericParameters

# GenericInvariance.java



- org.javaturk.oofp.ch11.generics.GenericInvariance

# Genellerde Joker (Wildcard) - I



- Bu problem ancak alt ve üstten sınırlı **joker (wildcard)** kullanımıyla çözülebilir.
- Bir genel yapı, **joker (wildcard)** ile ifade edilebilir.
- Java'da joker ? (soru işaretti) ile gösterilir ve bununla herhangi bir tip kastedilir.
- Sınırsız jokerli genel ifade ise <?> şeklindedir.

# Genellerde Joker (Wildcard) - II



- Bu ifadelerdeki ? işaretine sınırsız joker (unbounded wildcard) denir çünkü onunla her tip ya da herhangi bir tip kastedilir.
- Dolayısıyla sadece ? kullanıldığında, tip bilgisi belirlenmediğinden, okuma yapılabilir ama yazma yapılamaz.

```
List<?> list = new ArrayList<>();
list.add(new Object()) // Error!
Object obj = list.get(0);
list.forEach((Object o) → System.out.println(o));
```

# Genellerde Joker (Wildcard) - III



- Sınırsız jokerli bir yapıdan okuma yapıldığında `java.lang.Object` nesnesi alınır.
- Sınırsız joker kullanılarak `List` nesnesi oluşturulduğunda üzerinde ancak şu metodlar çağrılabılır:
  - `java.lang.Object` metodları,
  - `List` nesnesindeki tip belirtmeye gerek duymayan metodlar.

```
Iterator<?> i = List.iterator();
list.add(new Object()); // Error!
list.set(1, "I love Java"); // Error!
```

```
UnaryOperator<Integer> operator = j → j * j;
list.replaceAll(operator); // Error!
```

# Genellerde Joker (Wildcard) - IV



- Sınırsız joker kullanıldığında tip değişkeni kullanılmaz.
- Java API'sinde zaman zaman sınırsız joker kullanılır.
  - Örneğin `java.util.List` arayüzünde:

```
boolean containsAll(Collection<?> c)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
```

# Genellerde Joker (Wildcard) - IV



- Nitekim bu metodlara geçilen **Collection** nesnesinin genel tipinin belirtilmesine/bilinmesine gerek yoktur.
  - Bir List nesnesinden örneğin, herhangi bir tipe has oluşturulmuş **Collection**'ın varlığı **containsAll ()** metoduyla sorgulanabilir.
  - Bu yüzden **containsAll()** metoduna sınırsız jokerli **Collection** argümanı geçirilir.

# Genellerde Joker (Wildcard) - IV



- Bu metodların gerçekleştirmeleri **java.util.AbstractCollection** sınıfındadır.

```
boolean containsAll(Collection<?> c)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
```

- Tüm gerçekleştirmelerde ya **Object** sınıfının metodları çağrılır ya da **AbstractCollection** sınıfının, torbanın tuttuğu nesnelerin gerçek tiplerini bilmesinin gerekmendiği, **contains()** ya da **size()** gibi metodlar çağrılır.

# Genellerde Joker (Wildcard) - V



- Benzer şekilde `java.util.Collections` sınıfındaki statik metodlarda sınırsız joker kullanılır:

```
boolean disjoint(Collection<?> c1, Collection<?> c2)
int frequency(Collection<?> c, Object o)
int indexOfSubList(List<?> source, List<?> target)
int lastIndexOfSubList(List<?> source, List<?> target)
void reverse(List<?> list)
void rotate(List<?> list, int distance)
void shuffle(List<?> list)
void shuffle(List<?> list, Random rnd)
void swap(List<?> list, int i, int j)
```

- `disjoint()`, iki tane bilinmeyen tipe has `Collection` nesnesinin ortak elemanı olup olmadığını sorgular. `reverse()`, `rotate()` ve `reverse()` herhangi tipe has bir `List` üzerinde çalışabilir.

# UnboundedWildcardExample.java



- org.javaturk.oofp.ch11.wildcard.  
UnboundedWildcardExample

# Geneller ve Alt Tipler (Tekrar)



- Aralarında is-a ilişkisi olan iki tipe has oluşturulan generic tipli nesneler birbirlerine atanamazlar!
- Dolayısıyla `List<Manager>`, `List<Employee>`'nin bir alt tipi değildir ve `List<Employee>` beklenen yere `List<Manager>` geçilemez.
- Her farklı tipte liste için ayrı bir metoda ihtiyaç vardır.

```
List<Employee> employees = new ArrayList<>();
makeUpTeam1(employees);

List<Manager> managers = new ArrayList<>();
makeUpTeam1(managers); // Can't do that!
makeUpTeam2(managers); // Need another method!

static void makeUpTeam1(List<Employee> employees){..}
static void makeUpTeam2(List<Manager> employees){..}
```

# GenericParameters.java



- org.javaturk.oofp.ch11.generics.GenericParameters

# Üstten Sınırlı Joker



- Bu problemi çözmenin yolu, yani, belirtilen tip veya onun alt tiplerini kabul eden generic bir ifade **üstten sınırlı bir joker (upper-bounded wildcard)** ile mümkündür.
- Üstten sınırlı joker ifadesi `<? extends Type>` şeklindedir.
- Bu ifadede verilen tipin ve tüm alt tiplerinin nesneleri kastedilmektedir.

```
List<Employee> employees = new ArrayList<>();
makeUpTeam(employees);

List<Manager> managers = new ArrayList<>();
makeUpTeam(managers); // Can do that!

void makeUpTeam(List<? extends Employee> employees){
    ..
}
```

# UpperboundedWildcardExample.java



- org.javaturk.oofp.ch11.wildcard.  
**UpperboundedWildcardExample**

# Geneller ve Sarmalayan Tipler - I



- Ayrıca sarmalayan (wrapper) sınıflarda da aşağıdaki duruma dikkat edin:

```
List<Integer> ints = new ArrayList<>();  
ints.add(3);  
ints.add(new Integer(5));  
short s = 7;  
ints.add(new Short(s));    // Compiler error!  
ints.add(new Long(3L));   // Compiler error!
```

- Çünkü sarmalayan (wrapper) nesneler arasında **is-a** ilişkisi yoktur.
  - Sadece **Number** sınıfı, **Byte**, **Short**, **Integer**, **Long**, **Float** ve **Double**'ın bir üst tipidir.

# Geneller ve Sarmalayan Tipler - II



- Aşağıdaki metotta da üstten sınırlı generic tip kullanılmıştır:

```
List<Number> square(List<? extends Number> numbers) {  
    ...  
}
```

- Bu durumda bu metoda sadece **Number** sınıfına has değil, **Byte**, **Short**, **Integer**, **Long**, **Float** ve **Double** tiplerine has **List** nesneleri de geçilebilecektir.
- Eğer aşağıdaki gibi bir metot tanımlasaydık sadece ve sadece **Number** sınıfına has **List** nesneleri geçilebilecekti.

```
List<Number> square(List<Number> numbers) {  
    ...  
}
```

# WrapperGenerics.java



- org.javaturk.oofp.ch11.generics.WrapperGenerics

# Alttan Sınırlı Joker - I



- Altan sınırlı joker ifadesi `<? super Type>` şeklindedir.
- Bu ifade ile verilen tipin (type) ve tüm üst tiplerinin nesneleri kastedilmektedir, dolayısıyla alt sınır ifadedeki tiptir.
- Üst tip, bir sınıf olabileceği gibi bir arayüz de olabilir.
- Altan sınırlı generic ifadeler, en az verili tipten oluşturulmuş generic yapılar isterler.

# Altın Sınırlı Joker - II



- Örneğin `java.util.Iterable` arayüzüünü ve bu arayüze Java SE 1.8 ile gelen aşağıdaki default metodu düşünelim.

```
interface Iterable<T>
```

- Iterable arayüzünün T tipi, üzerinden aldığı torba nesnesinin generic tipidir.

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

- Bu metot Iterable nesnesinin kendisine has olduğu tipe ya da daha üst tiplerine has bir `Consumer` nesnesini argüman olarak almaktadır.
- `Consumer<? super T>` ifadesindeki `<? super T>`, T ve üst tiplerini göstermektedir.

# Altan Sınırlı Joker - III



- Ayrıca bu metod **Iterable** nesnesinin her elemanı üzerinde verilen Consumer nesnesinin **accept()** metodunu uygular.

```
interface Consumer<T>
```

```
void accept(T t)
```

- **Iterable** nesnesinin generic tipi olan **T**, **accept()** metodunun aldığı argümanın tipidir.

# Alttan Sınırlı Joker - III



- Dolayısıyla aşağıdaki metot şu anlama gelmektedir:
- Öyle bir **Consumer** nesnesi sağla ki bu nesne, kendisine uygulanacak olan Iterable nesnesinin generic tipi ya da üst tipine has olsun, sadece onlarla çalışsin.

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

- Neden? Çünkü, **Consumer** nesnesi en çok **T** üzerindeki metotları çağırmasını bilsin.

# LowerboundedWildcardExample.java



- org.javaturk.oofp.ch11.wildcard.  
**LowerboundedWildcardExample**

# Bounded Wildcard Kullanımı - I



- Java API'sinden örnekler:
  - **binarySearch (List<? extends Comparable<? super T>> list, T key)**
  - **binarySearch (List<? extends T> list, T key, Comparator<? super T> c)**

# Bounded Wildcard Kullanımı - II



- **Person <= Employee <= Manager <= Director** ise
- **List<Employee>** varsa **Comparable** nesnesi en az **Employee** generic olmalı, **Person'a** generic de olabilir.
- Çünkü **Person'i** compare edebilen **Employee'yi** de compare edbilir.

# Üstten ve Altan Sınırlı Jokerler



- Üstten sınırlı joker ifadesinde `<? extends Type>` bu ifadeye uygun olan nesnenin arayüzü verili tip (type) tarafından belirlenir.
- Dolayısıyla nesne üzerinde çağrılacak metodlar, tip (type) üzerindeki metodlardır.
- Yani `List<? extends Employee>` ifadesine uyan her element, Employee'nin arayüzüne sağılar.
- Ama alttan sınırlı joker ifadesi `<? super Type>` durum farklıdır.

# Team.java



- org.javaturk.oofp.ch11.generics.Team

# Genellerde Joker (Wildcard)



- Bir generic yapının bekleniği yere, o generic yapıdaki tipin alt tiplerinden oluşan başka bir generic yapının geçilebileceğini ifade etmenin yolu, joker (wildcard) kullanmaktadır.
- Java generic yapılarında joker kullanımının bir kaç farklı yolu vardır.
  - **<? extends T>**: T ve T'nin alt tiplerine has bir generic yapı oluşturur.
  - **<? super T>**: v ve T'nin üst tiplerine has bir generic yapı oluşturur.

# Generic Yapıarda Joker (Wildcard)



- Bir generic yapının bekleniği yere, o generic yapıdaki tipin alt tiplerinden oluşan başka bir generic yapının geçilebileceğini ifade etmenin yolu, joker (wildcard) kullanmaktadır.
- Java generic yapılarında joker kullanımının bir kaç farklı yolu vardır.

```
List<Employee> employees = new ArrayList<>();
makeUpTeam(employees);

List<Manager> managers = new ArrayList<>();
makeUpTeam(managers); // Can do that now!

employees = managers; // Can do that now!

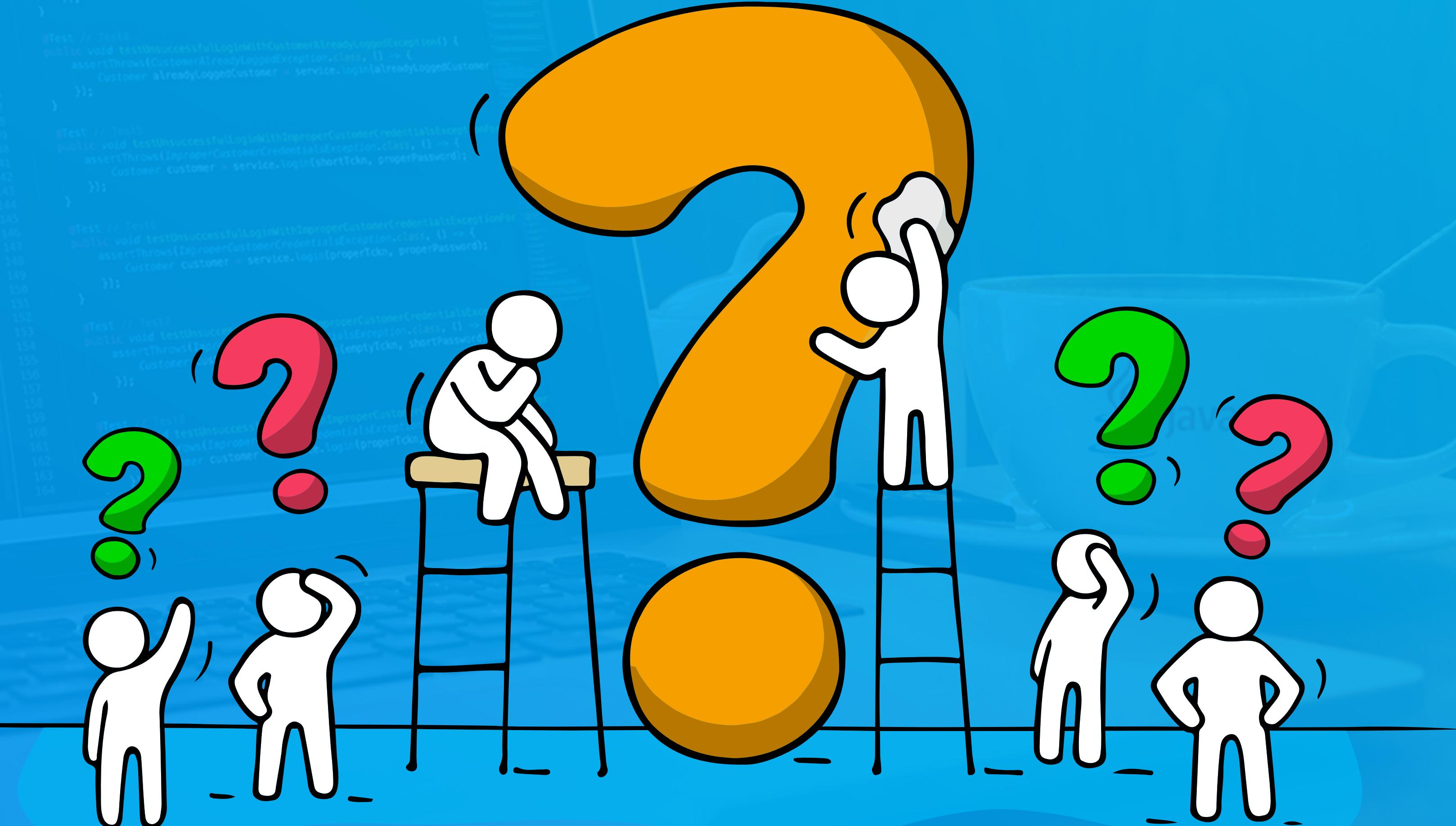
static void makeUpTeam(List<? extends Employee> employees) {
}
```

# GenericsAndSubtyping.java



- org.javaturk.oofp.ch11.generics.GenericsAndSubtyping

# Soru ve Cevap Zamani!



# Genel Tipli Sınıf Tanımlama



```
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 testUnsuccessfulLoginWithImproperCustomerCredentialsException()
141     assertThrows(ImproperCustomerCredentialsException.class, () -> {
142         Customer customer = service.login(shortTckn, properPassword);
143     });
144 }
145
146 @Test // Test6
147 public void testUnsuccessfulLoginWithImproperCustomerCredentialsException()
148     assertThrows(ImproperCustomerCredentialsException.class, () -> {
149         Customer customer = service.login(emptyTckn, shortPassword);
150     });
151 }
152
153 @Test // Test7
154 public void testUnsuccessfulLoginWithImproperCustomerCredentialsException()
155     assertThrows(ImproperCustomerCredentialsException.class, () -> {
156         Customer customer = service.login(emptyTckn, emptyPassword);
157     });
158 }
159
160 @Test // Test8
161 public void testSuccessfulLoginWithImproperCustomerCredentialsException()
162     assertThrows(ImproperCustomerCredentialsException.class, () -> {
163         Customer customer = service.login(emptyTckn, emptyPassword);
164     });
165 }
```



- Daha önce “genel tip kullanılarak oluşturulan bir nesne, sadece o generic tipten olan nesnelerle çalışabilir.” dendi.
- Bu amaçla Java torbalarında tip parametreleri kullanılır.

```
List<Student> students = new ArrayList<>();
```

- Ama bu kullanım, genel tip kullanımının ufkak bir parçasıdır.
- Bu konu ile ilgili daha pek çok detay vardır.

# Genel Sınıf Oluşturma



- Genel tipli bir sınıf (aslen bir tip) nasıl oluşturulur?
- Bu amaçla sınıfı oluştururken `<>` operatörü içinde tip değişkeni sağlanmalıdır.
- Tip değişkeni sayısında bir sınırlama yoktur ama genelde bir (en çok iki) tane olma eğilimindedir.
- Tip değişkenlerinin hangi yönde sınırlı olacaklarına karar verilmelidir.

# Genel Sınıf Oluşturma



- Tip değişkenlerini sınıf içerisinde, sınıfın sorumluluklarını yerine getirecek şekilde kullanılır.
- Daha sonra da genel sınıfın nesnesini oluştururken uygun tip(ler)de nesne(ler) geçilir.

# MyGenericClass.java



- org.javaturk.oofp.ch11.generics.MyGenericClass

# GenericReturnType.java



- org.javaturk.oofp.ch11.generics.GenericReturnType

# Team.java



- org.javaturk.oofp.ch11.generics.Team

# Marriage.java



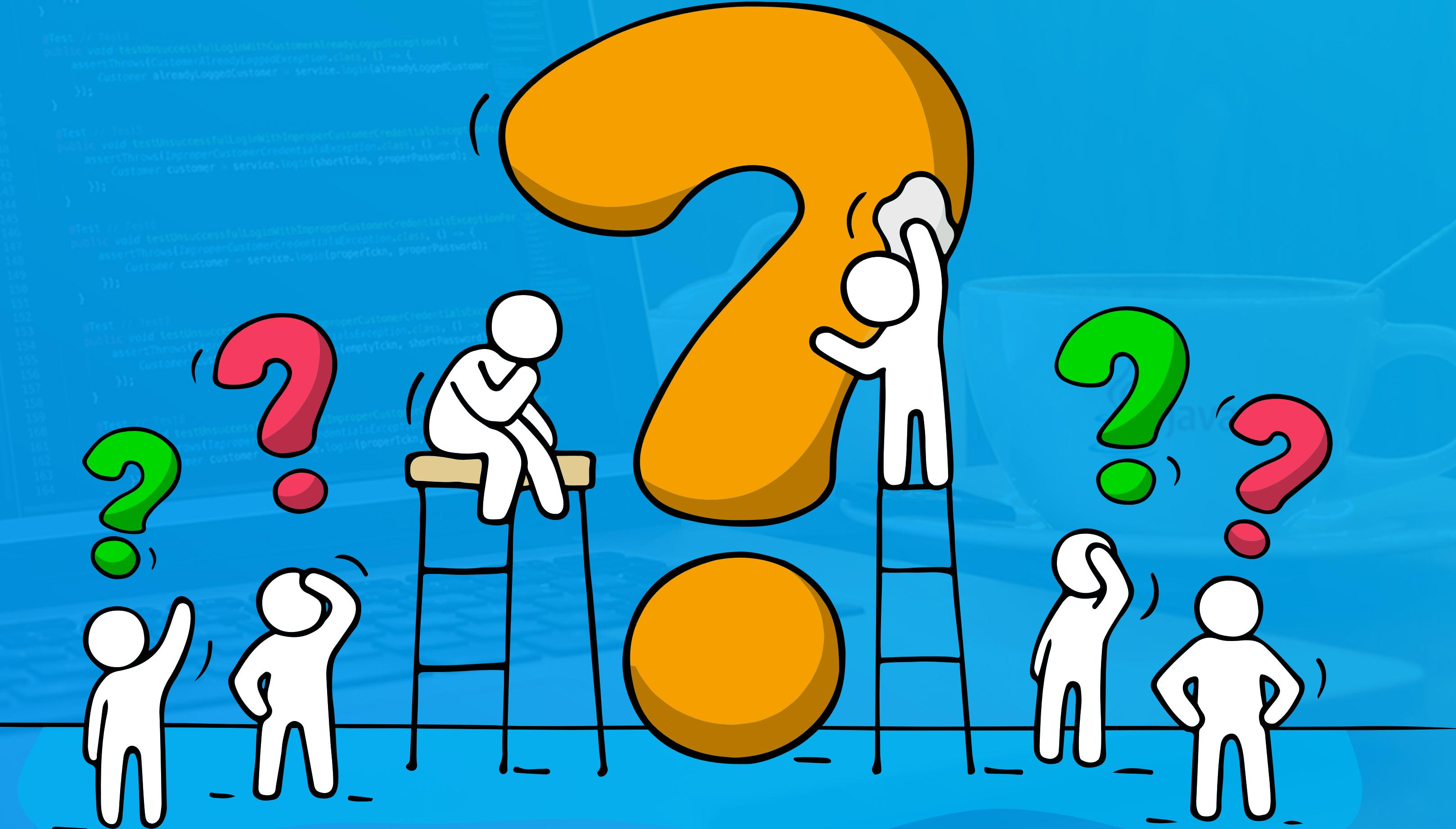
- org.javaturk.oofp.ch11.generics.Marriage

# Generics Ne Sağlar?



- Generic yapıları ne sağlar?
  - Daha okunabilen ve kısa kod,
  - Daha sağlam (robust) kod dolayısıyla daha güvenilir (reliable)
  - yazılım sağlar.

# 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(ImproperCustomerCredentialsException.class, () -> {
143             Customer customer = service.login(shortTckn, properPassword);
144         });
145
146     @Test // Test5
147     public void testUnsuccessfulLoginWithInproperCustomerCredentialsExceptionFor
148         assertThrows(ImproperCustomerCredentialsException.class, () -> {
149             Customer customer = service.login(properticks, properPassword);
150         });
151
152     @Test // Test6
153     public void testUnsuccessfulLoginWithInproperCustomerCredentialsExceptionFor
154         assertThrows(ImproperCustomerCredentialsException.class, () -> {
155             Customer customer = service.login(emptyTckn, shortPassword);
156         });
157     }
158 }
```

# Bölüm Sonu

## Soru ve Cevap Zamani!

