



# Java ile Nesne Merkezli ve Fonksiyonel Programlama

# *11. Bölüm: Genel Programlama*



# Eğitmen:

# Akın Kaldıroğlu

# Çevik Yazılım Geliştirme ve Java Uzmanı

# Konular



- **java.lang.Object'e Genel Yapılar**
- **Genel Yapılar**
  - Genel Tip Kullanımı
- **Joker, Alt ve Üst Sınır Tipler**
  - Sınırsız Joker
  - Üstten Sınırlı Joker
  - Altan Sınırlı Joker
- **Genel Tipli Sınıf Tanımlama**
- **Ödevler**

# java.lang.Object'e Genel Yapılar

# Generic Ne Demek?



- **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.

# Object Referansı - I



- Java'da en temel genel programlama (**generic programming**), `java.lang.Object` sınıfıyla yapılır.
- 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` bekleyen bir metoda geçebilir, `Object` tipinde bir referans döndüren bir metottan geriye döndürebiliriz:

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

# Heterojen Collection - I



- Java'ya generics özelliği gelmeden önce **Collection** heterojendi, en genel tip olan **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



- Benzer şekilde torbaların iteratörleri de yine **Object** alıp verirlerdi:
  - **java.util.Iterator**'ın **next()** metodu **Object** döndürür, **remove (Object o)** metodu **Object** alır.

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

```
List students = new ArrayList();
students.add(new Student(...));
students.add(new Student(...));
students.add(new Student(...));
```

```
Object o = students.get(i);
if(o instanceof Student){
    Student s = (Student) o;
    ...
}
```

```
Iterator it = students.iterator();
while(it.hasNext()){
    Student s = (Student) it.next();
}
```

# Java SE 1.4 Collection API



java.util

## Interface Collection

### Method Summary

boolean	<a href="#">add(Object o)</a>	Ensures that this collection contains the specified element (optional operation).
boolean	<a href="#">addAll(Collection c)</a>	Adds all of the elements in the specified collection to this collection (optional operation).
void	<a href="#">clear()</a>	Removes all of the elements from this collection (optional operation).
boolean	<a href="#">contains(Object o)</a>	Returns true if this collection contains the specified element.
boolean	<a href="#">containsAll(Collection c)</a>	Returns true if this collection contains all of the elements in the specified collection.
boolean	<a href="#">equals(Object o)</a>	Compares the specified object with this collection for equality.
int	<a href="#">hashCode()</a>	Returns the hash code value for this collection.
boolean	<a href="#">isEmpty()</a>	Returns true if this collection contains no elements.
<a href="#">Iterator</a>	<a href="#">iterator()</a>	Returns an iterator over the elements in this collection.
boolean	<a href="#">remove(Object o)</a>	Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	<a href="#">removeAll(Collection c)</a>	Removes all this collection's elements that are also contained in the specified collection (optional operation).
boolean	<a href="#">retainAll(Collection c)</a>	Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	<a href="#">size()</a>	Returns the number of elements in this collection.
<a href="#">Object[]</a>	<a href="#">toArray()</a>	Returns an array containing all of the elements in this collection.
<a href="#">Object[]</a>	<a href="#">toArray(Object[] a)</a>	Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

# Object Referansı- II



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

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

# Object Referansın Problemleri - I



- Bu durumun yanı tüm referansları en genel **Object** tipinden yapmanın bazı problemleri vardır:
  - Alt tipten nesnelere has yapılar oluşturmak yanı **Object** nesnesi yerine alt sınıflardan nesneler kullanmak mümkün olmaz.
  - Yani sadece **Student** tutan bir torba oluşturmak mümkün değildir.

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

# Object Referansın Problemleri - II



- Sürekli Object nesnesi kullanmak alçaltma (downcast) ihtiyacı doğurur.
  - Alçaltma (downcast) da instanceof kullanma ihtiyacına sebep olabilir.

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

```
Object o = channel.receive();
if(o instanceof Student){
    Student s = (Student) o;
    ...
}
```

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

# RawTypeList.java



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

# Genel Yapılar (Generics)

# Genel Yapılar (Generics) - I



- Java'ya 1.5 ile birlikte genel tipler (türler) (generic types ya da kısaca generics) gelmiştir.
- Muhtemelen Java'nın tarihinde, 8. sürüm ile gelen fonksiyonel programlama ile birlikte çehresini en fazla değiştiren özellik olmuştur.
- Genel yapılar (generics) ile `java.lang.Object` nesnesi yerine farklı bir tipe has/genel yapılar oluşturmak mümkündür.

# Genel Yapılar (Generics) - II



- Bu bölümde genel tipler, daha basitten karmaşığa doğru giderek, iki farklı şekliyle ele alınacaktır:
  - Önce var olan genel tiplerin kullanımı,
  - Sonrasında genel tiplerin oluşturulması.



- Bir genel tip (generic type) **<> baklava dilimi işlemcisi (diamond operator)** ile tanımlanır.
- Bu tanımda **<>** içinde **genel tip (generic type)** belirtilir.
- Bu amaçla genel tipi göstermek için **E, T, K, V vb. tip değişkeni** ya da **tip parametresi(type variable)** ya da **type parameter** kullanılır.
  - Tip değişkenlerinin büyük harflerle gösterilmesi bir gelenektir.
  - Örneğin **Collection<E>**, ya da **Map<K, V>** gibi.

# Genel Tipler - II



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

# Collection<E> - I

- Collection<E>'de E tip parametresidir ve nesneyi E tipine has kılar, dolayısıyla Collection artık homejendir.
- Tip parametresi Collection<E>'in metodlarında da kullanılır:
  - **boolean add(E e):** Collection<E>'a sadece E tipinden nesne ekler.

```
public interface Collection<E> extends Iterable<E> {  
  
    boolean contains(Object o);  
  
    Iterator<E> iterator();  
  
    Object[] toArray();  
  
    <T> T[] toArray(T[] a);  
  
    default <T> T[] toArray(IntFunction<T[]> generator)  
  
    boolean add(E e);  
  
    boolean remove(Object o);  
  
    boolean containsAll(Collection<?> c);  
  
    boolean removeAll(Collection<?> c);  
  
    default boolean removeIf(Predicate<? super E> filter)  
  
    boolean retainAll(Collection<?> c);  
  
    boolean equals(Object o);  
  
    default Spliterator<E> spliterator()  
  
    default Stream<E> stream()  
  
    default Stream<E> parallelStream()  
    ...  
}
```

# Collection<E> - II

- Collection<E>'de E bir semboldür ve tipi gösteren bir parametredir ve kod yazılırken E yerine gerçek bir tip kullanılır.
- Yani Collection nesnesi, oluşturulurken verilen tipe has (generic to given type) olur, mümkün olduğunca sadece o tip ile işlem yapar.

```
public interface Collection<E> extends Iterable<E> {  
  
    boolean contains(Object o);  
  
    Iterator<E> iterator();  
  
    Object[] toArray();  
  
    <T> T[] toArray(T[] a);  
  
    default <T> T[] toArray(IntFunction<T[]> generator)  
  
    boolean add(E e);  
  
    boolean remove(Object o);  
  
    boolean containsAll(Collection<?> c);  
  
    boolean removeAll(Collection<?> c);  
  
    default boolean removeIf(Predicate<? super E> filter)  
  
    boolean retainAll(Collection<?> c);  
  
    boolean equals(Object o);  
  
    default Spliterator<E> spliterator()  
  
    default Stream<E> stream()  
  
    default Stream<E> parallelStream()  
    ...  
}
```

# java.util.List API - I



- `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>

# java.util.List API - II



- List arayüzüünün 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>

extends [Collection<E>](#)

# Java SE 1.4 API



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



```
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 }
```

# Genel Tip Kullanımı

# Genel Tiplerin Kullanımı - I



- **ArrayList**, bir **Collection** nesnesidir ve genel bir tiptir.
- **ArrayList** nesnesi oluşturulurken tip parametresi olan **E** yerine gerçek tip geçilir:

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

# Genel Tiplerin Kullanımı - II



- Java SE 1.7 ile birlikte kurucu çağrısı yapılırken sadece <> kullanmak yeterli hale gelmiştir:

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

- Ama bir referansa atamadan kullanmak istenirse bu durumda kurucudan sonra genel tipi vermek gereklidir.

```
new ArrayList<Student>();
```

- Çünkü tip olmadan <> kullanımı, **ArrayList<>()** gibi, problemlidir.

# Genel Tiplerin Kullanımı - III



- Genel tip kullanılarak oluşturulan bir nesne, sadece o genel tipten olan nesnelerle çalışabilir.
- Örneğin, **ArrayList** 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

# Genel Tiplerin Kullanımı - III



- Bazı sınıflar birden çok genel tip ile çalışır.
  - Bu durumda nesnesi oluştururken birden çok gerçek tip verilir.
- Örneğin, `Map<K, V>`, anahtar ve değer için iki tane tip parametresi alır.
  - Bu durumda `V put(K key, V value)` metodu da iki tip parametresi alır.

# GenericMap.java



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

# Genel Tiplerin Kullanımı - IV



- Tip parametresi alan nesnelerin metodları da tip parametresi alır.
- Bu durumda metodlar da sadece o tipten parametre ile çalışır hale gelir.
- Örneğin **Comparator<T>** tipinin tek soyut методу da **int compare (T o1, T o2)** olur.
- Bu durumda **Comparator<T>** hangi tipe has oluşturulmuşsa, **compare ()** методу da sadece o tipten iki nesne alır.

# SchoolTest1.java



- org.javaturk.oofp.ch11.school.SchoolTest1
  - EmployeeComparator
  - TeacherComparator

# İlkel Tipli Genel Kullanımı



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

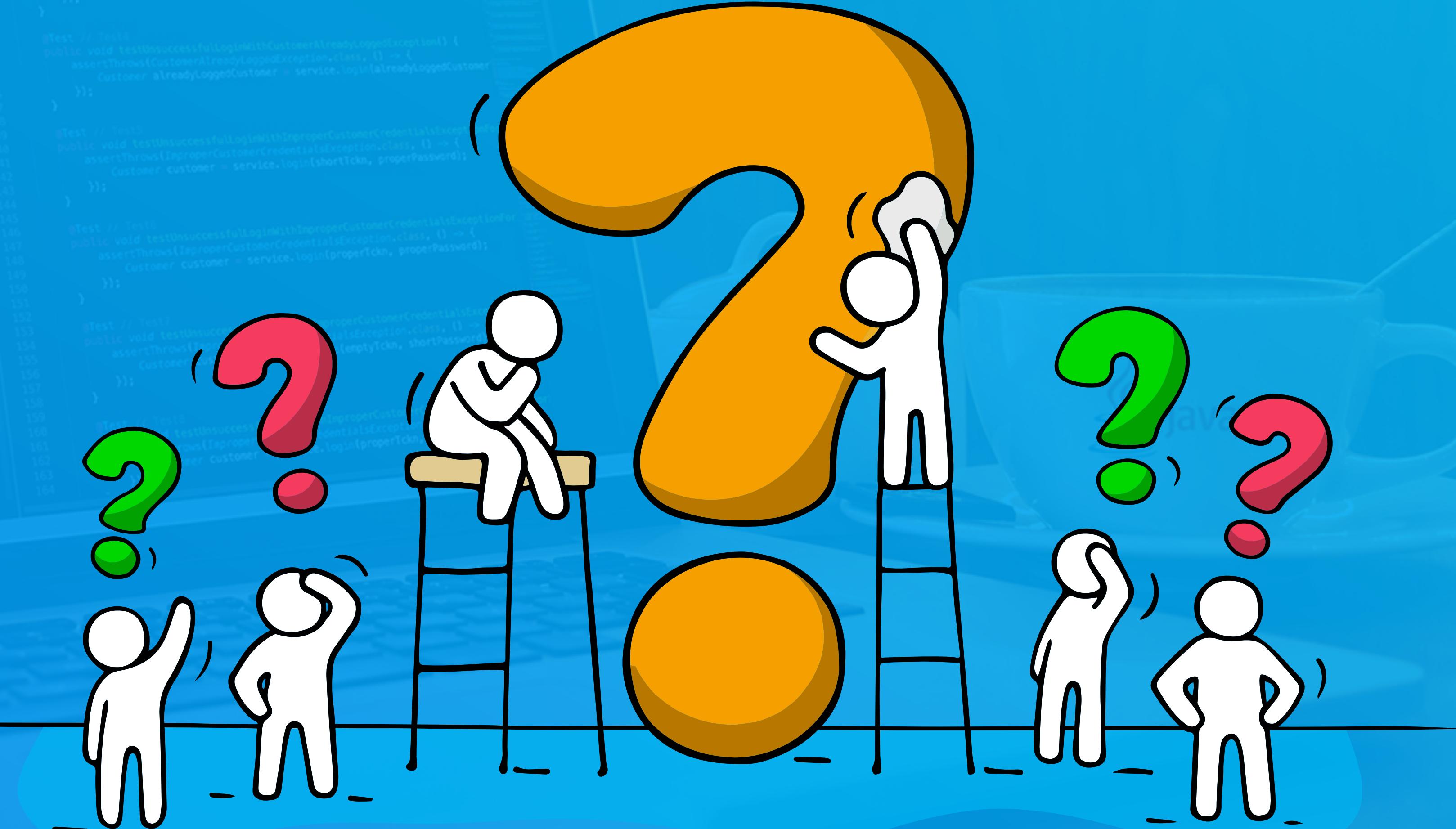
```
List<Integer> ints = new ArrayList<>();  
ints.add(3);           // Auto-boxing  
ints.add(new Integer(5));  
int i = ints.get(1);   // i is 5! Auto-unboxing  
short s = 11;  
ints.add(new Short(s)); // Compiler error!  
ints.add(new Long(3L)); // Compiler error!
```

# PrimitiveGenerics.java



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

# Soru ve Cevap Zamani!



# Joker, Alt ve Üst Sınır Tipiler

# Genel 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 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.

# Genel Alt Tipler - I



- Bir tipe has genel yapı ile o tipin bir alt tipine has genel yapı arasında bir is-a ilişkisi var mıdır?
  - 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!
```

# Genel 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!

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

# GenericInvarianceAndParameters.java



- org.javaturk.oofp.ch11.generics.  
**GenericInvarianceAndParameters**
- Bu örnekte **makeUpTeam1()** ve **makeUpTeam2()** metodlarının  
kullanımını inceleyin.

# Joker (Wildcard) - I



- Bu türden problemler genel yapıda **joker (wildcard)** kullanımıyla çözülebilir.
- Joker ? (soru işaretti) ile gösterilir ve bununla herhangi bir tip kastedilir.
- Üç tür joker kullanımı vardır:
  - Sınırsız joker,
  - Altan sınırlı joker,
  - Üstten sınırlı joker.

# Joker (Wildcard) - II



- Java generic yapılarında joker kullanımının bir kaç farklı yolu vardır.
  - Sınırsız joker `<?>`: Herhangi bir tipe genel bir yapı oluşturur.
  - Üstten sınırlı joker `<? extends T>`: `T` ve `T`'nin alt tiplerine has bir genel yapı oluşturur.
  - Altta sınırlı `<? super T>`: `T` ve `T`'nin üst tiplerine has bir genel yapı oluşturur.



```
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 }
```

# SİNIRSIZ JOKER

# Sınırsız Joker - I



- Tip belirtmeden tek başına kullanıldığında ? işaretini, **sınırsız jokerdir** (**unbounded wildcard**) çünkü onunla her tip ya da herhangi bir tip kastedilir.
- Sadece ? kullanıldığında, tip bilgisi belirlenmediğinden, okuma yapılabilir ama yazma yapılamaz.
- Dolayısıyla <?> kullanarak bir tanımlama yapılmaz, <?> daha çok metotlara geçilen parametrelerde kullanılır.

```
List<?> list = new ArrayList<>();
list.add(new Object()) // Error!
Object obj = list.get(0); // Not an error!
list.forEach((Object o) → System.out.println(o));
```

# Sınırsız Joker - II



- Sınırsız jokerli bir yapıdan okuma yapıldığında **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!
```



- 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)
```

- 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 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.

# Sınırsız Joker - V



- Benzer şekilde `java.util.Collections` sınıfındaki statik metodlarda sınırsız joker kullanılır çünkü `Collections` genel tipli değildir:

```
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 `shuffle()` herhangi tipe has bir `List` üzerinde çalışabilir.

# UnboundedWildcardExample.java



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



```
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(AlreadyLoggedCustomerException.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", "shortPassword");  
150     }).getClass();  
151 }  
152  
153 @Test // Test 6  
154 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
155     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
156         customer = service.login("properTckn", "properTckn");  
157     }).getClass();  
158 }  
159  
160 @Test // Test 7  
161 public void testSuccessfullLoginWithImproperCustomerCredentials() {  
162     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
163         customer = service.login("properTckn", "properTckn");  
164     }).getClass();  
165 }
```

# Üstten Sınırlı Joker

# Üstten Sınırlı Joker - I



- 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, her farklı tipte liste için ayrı bir metoda ihtiyaç vardır.

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

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

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

```
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){...}
```

# Üstten Sınırlı Joker - II



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

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

List<Manager> managers = new ArrayList<>();
makeUpTeam(managers);

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

```
List<? extends Employee> employees = new ArrayList<>();
List<Manager> managers = new ArrayList<>();
employees = managers; // Can do that!
```

# UpperboundedWildcardExample.java



- org.javaturk.oofp.ch11.wildcard.  
**UpperboundedWildcardExample**
- Bu örnekte `makeUpTeam1()` 'e karşı `makeUpTeam2()` metodunun kullanımını inceleyin.

# 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



- Bu durum da üstten sınırlı generic tip kullanılabilir:

```
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 geçilebilir.

# WrapperGenerics.java



- `org.javaturk.oofp.ch11.generics.WrapperGenerics`
- Bu örnekte `square1()` ve `square2()` metodunun kullanımını inceleyin.



```
111 public void testSuccessfulLogin() {  
112     assertEquals(CustomerNotFoundException.class, customer.  
113         login("customer1", "password1").getClass());  
114     assertEquals("Customer locked", customer.  
115         login("customer1", "password1").getReason());  
116 }  
117  
118 @Test // Test 1  
119 public void testSuccessfulLoginWithUnfoundCustomer() {  
120     assertEquals(UnfoundCustomerException.class, () -> {  
121         customer.unfoundCustomer = service.login(properties, properPassword);  
122     }).  
123 }  
124  
125 @Test // Test 2  
126 public void testSuccessfulLoginWithCustomerLocked() {  
127     assertEquals(CustomerLockedException.class, () -> {  
128         customer.lockedCustomer = service.loginLockedCustomerToken("token");  
129     }).  
130 }  
131  
132 @Test // Test 3  
133 public void testSuccessfulLoginWithCustomerAlreadyLogged() {  
134     assertEquals(CustomerAlreadyLoggedException.class, () -> {  
135         customer.alreadyLoggedCustomer = service.loginAlreadyLoggedCustomer();  
136     }).  
137 }  
138  
139 @Test // Test 4  
140 public void testSuccessfulLoginWithImproperCustomerCredentials() {  
141     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
142         customer = service.login(shortToken, properPassword);  
143     }).  
144 }  
145  
146 @Test // Test 5  
147 public void testSuccessfulLoginWithImproperCustomerCredentials() {  
148     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
149         customer = service.login(propertiesToken, properPassword);  
150     }).  
151 }  
152  
153 @Test // Test 6  
154 public void testSuccessfulLoginWithImproperCustomerCredentials() {  
155     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
156         customer = service.login(propertiesToken, shortPassword);  
157     }).  
158 }  
159  
160 @Test // Test 7  
161 public void testSuccessfulLoginWithImproperCustomerCredentials() {  
162     assertEquals(ImproperCustomerCredentialsException.class, () -> {  
163         customer = service.login(propertiesToken, propertiesToken);  
164     }).  
165 }
```

# Alttan Sınırlı Joker

# 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.
- Belirtilen tip bir sınıf olabileceği gibi bir arayüz de olabilir.
- Altan sınırlı genel ifadeler, en az verili tipten oluşturulmuş genel yapılar ister.

# Alttan 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 alındığı torba nesnesinin generic tipidir.

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

# Alttan Sınırlı Joker - III



- Bu metot **Iterable** nesnesinin kendisine has olduğu **T** tipine ya da daha üst tiplerine has bir **Consumer** nesnesini argüman olarak almaktadır.
- **Consumer<? super T>** ifadesindeki **<? super T>**, **T** ve üst tiplerini göstermektedir.
- Bu da anlaşılır bir durumdur çünkü **Consumer** üzerindeki **accept()** metodu **T**'ye geneldir.

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

# Alttan Sınırlı Joker - IV



- Dolayısıyla **forEach(Consumer<? super T>)** arayüzü şu anlamda 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.
  - Bu durumda **? super T** ile **T**'nin üst tipleri üzerinde metot çağrıabilen bir yapı, **T** üzerinde de metot çağrıabilir çünkü **T**'nin arayüzü en azından **? super T**'nin arayüzüdür.

# LowerboundedWildcardExample.java



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

# Alttan Sınırlı Joker - V

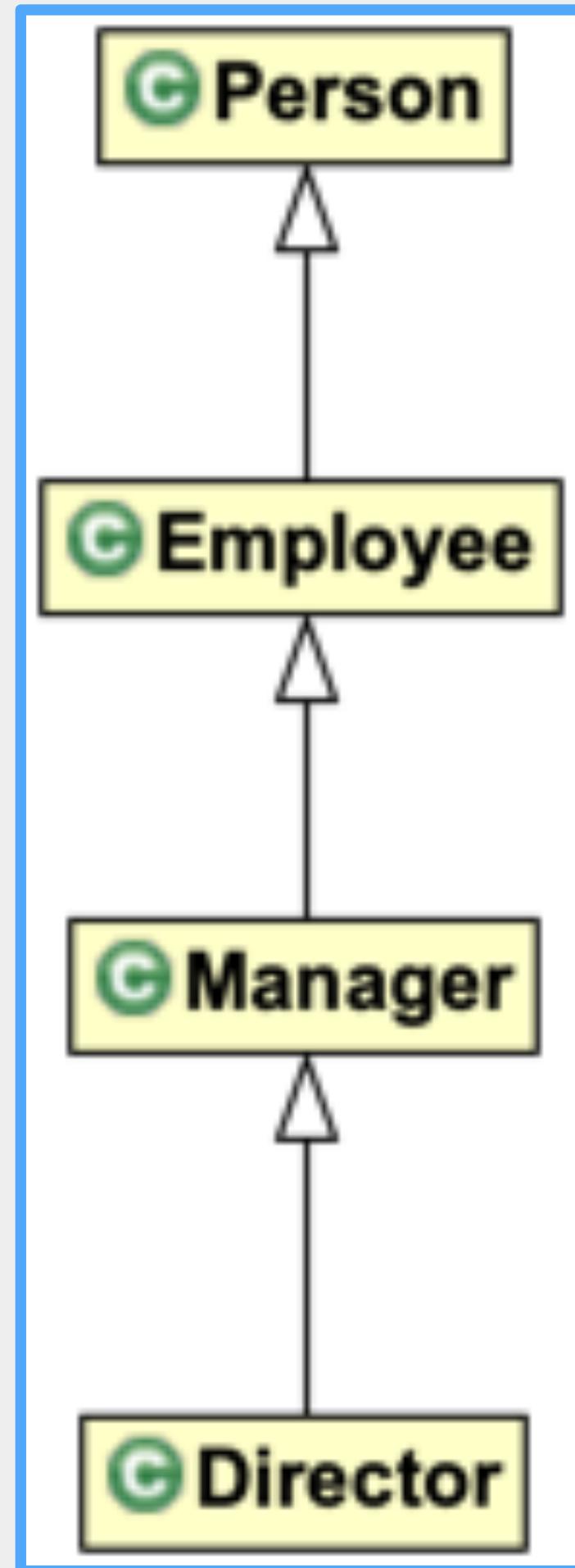


- Java API'sinden örnekler:
  - `binarySearch(List<? extends Comparable<? super T>> list, T key)`
  - `binarySearch(List<? extends T> list, T key, Comparator<? super T> c)`
    - T ve alt tiplerini tutan bir `List` içinde T tipinde bir nesneyi bulmak için, T ve üst tipleri için geçerli bir `Comparator` sağlamak gereklidir.
    - Eğer T'nin üst tipinde `compareTo()` metod gerçeklestirmesi varsa bu metot tüm hiyerarşi için geçerlidir.
  - ? `super T` yerine ? `extends T` olsaydı bu garanti edilemezdi.

# Alttan Sınırlı Joker - VI



- `List<Employee>` varsa `Comparable` nesnesi en az `Employee` genel olmalı yani en azından `Employee` nesnelerini kıyaslayabilir.
- `? super T` ifadesinden dolayı `Person`'a genel de olabilir çünkü `Person`'ı nesnelerini kıyaslayabilen `Employee`'yi de kıyaslayabilir.

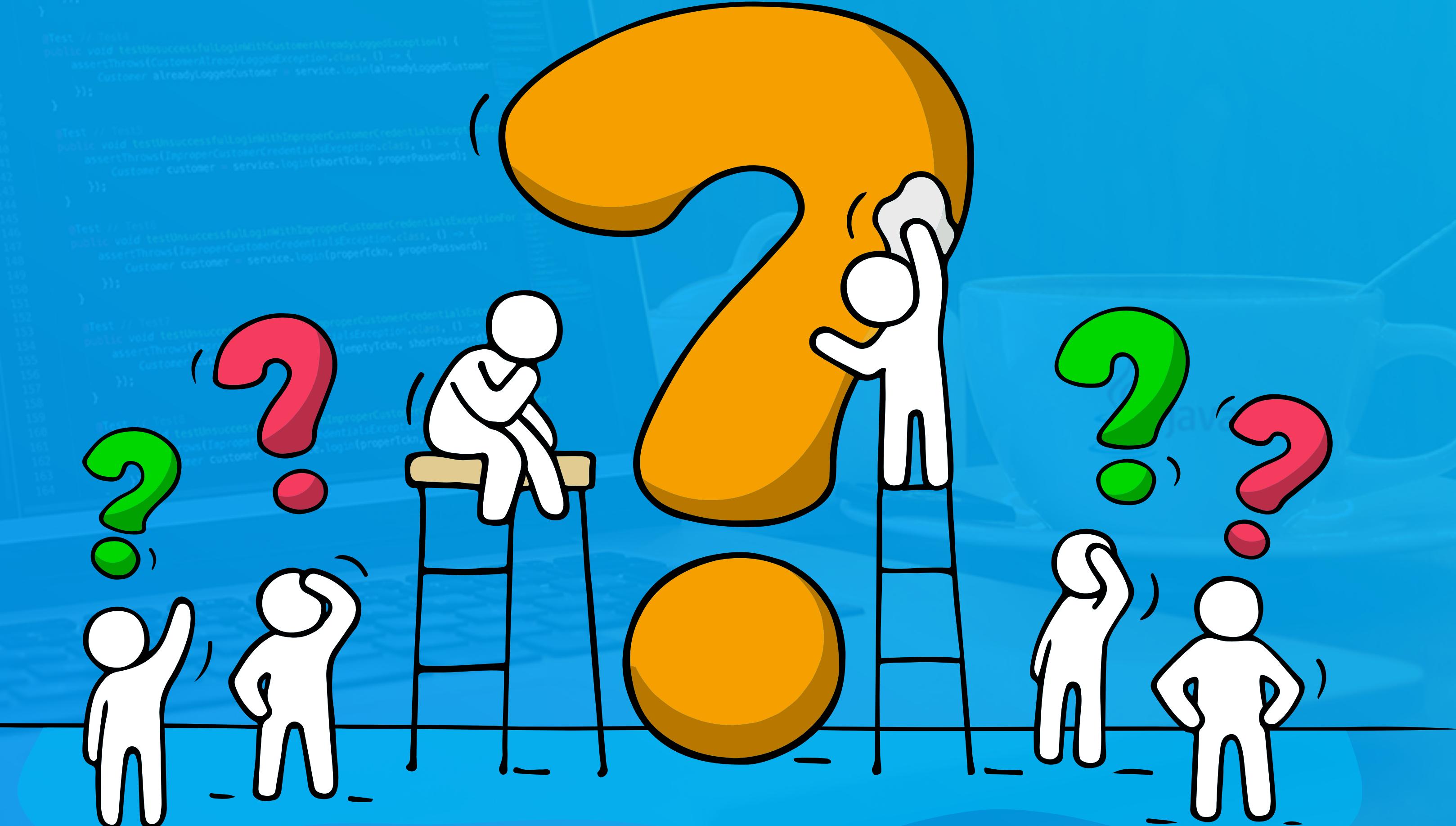


# Üstten ve Altan Sınırlı Jokerler



- Üstten sınırlı joker ifadesinde `<? extends Type>` bu ifadeye uygun olan nesnenin arayüzü verili tip (type) tarafından belirlenir.
- Dolayısıyla nesne üzerinde çağrılacak metodlar **Type** metodlarıdır.
- Yani `List<? extends Employee>` ifadesine uyan her element, `Employee`'nin arayüzüne sağılar.
- Ama alttan sınırlı joker ifadesi `<? super Type>` için durum farklıdır.
- Bu ifade ile sağlanan şey **Type**'ın herhangi bir üst tipinde sağlanan metodun için de **Type** geçerli olduğu olacaktır.

# 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



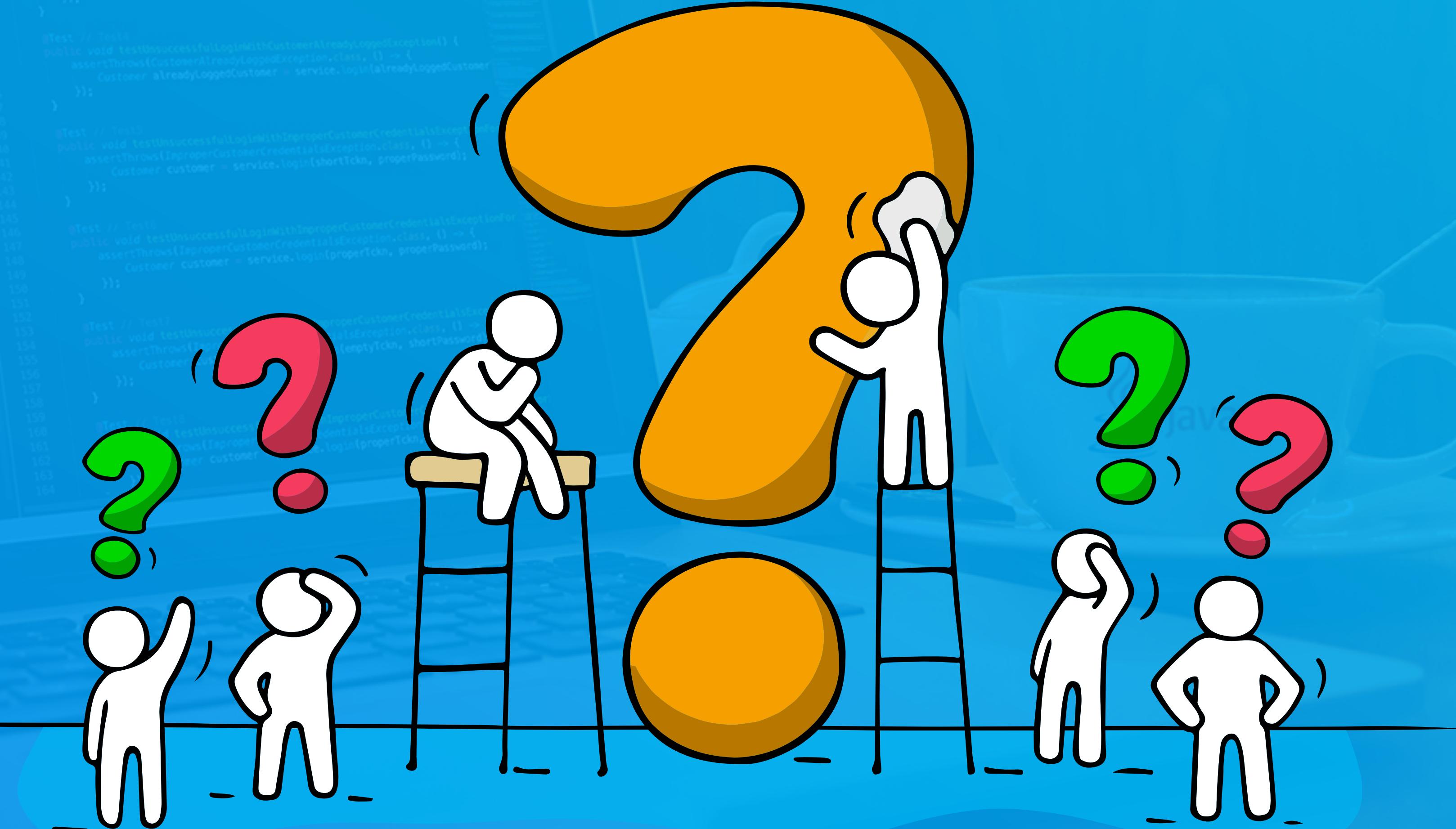
- org.javaturk.oofp.ch11.generics.Marriage

# 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!

