

# Java Generics

# Java Generics Öncesi Durum

- **Java 5** öncesinde tip bilgisi önceden kestirilemeyen durumlar için **java.lang.Object** tipi kullanılırdı
- Fakat bu java derleyicinin katı **tip kontrol kabiliyetinden** tam olarak yararlanmayı kısıtlıyordu
- Tip dönüşümleri ile ilgili **çalışma zamanında hataların** ortaya çıkmasına neden oluyordu

# Generics Olmadan List Kullanımı

```
List pilotListesi = new ArrayList();
```

```
pilotListesi.add(new Pilot("Jenson", "Button"));  
pilotListesi.add(new Pilot("Mark", "Webber"));  
pilotListesi.add("Hello World!"); ✓  
pilotListesi.add(new Pilot("Sebastian", "Vettel"));  
pilotListesi.add(new Pilot("Jarno", "Trulli"));
```

```
Pilot pilot = (Pilot)pilotListesi.get(0);
```

```
for(Object o:pilotListesi) {  
    Pilot p = (Pilot)o;  
    System.out.println(p.getAdi());  
}
```

Pilot tipini kullanabilmek  
için explicit cast gerekir

Jenson  
Mark

Exception in thread "main" [java.lang.ClassCastException:](#)  
**[java.lang.String cannot be cast to com.javaegitimleri.Pilot](#)**  
at [com.javaegitimleri.CollectionsTest.main\(CollectionsTest.java:43\)](#)

Hata!

# Java Generics Nedir?

- **Tipi önceden bilinemeyen,** kestirilemeyen nesneler üzerinde işlem yapacak kod yazmayı sağlar
- Yazılan kod içerisinde tip yerine bir **type placeholder** (generic type) kullanılır
- Bu placeholder tip bilgisi kesinleştiği vakit **asıl tip** (actual type) bu kod ile ilişkilendirilerek derleyicinin kodu **belirtilen tipe göre derlemesi** sağlanır

# Java'nın Kendi İçinden Generics Örneği

```
public interface List<T> {  
    void add(T x);  
    Iterator<T> iterator();  
}
```

T değişkenine **formal/generic type parameter** adı verilir

Metot ve constructor tanımlarındaki input parametrelere benzetilebilir

```
List<Integer> intList = new ArrayList<Integer>();
```

```
intList.add(new Integer(1));
```

```
int intVal = intList.get(0);
```

```
for(Integer i:intList) {  
    System.out.println(i);  
}
```

Integer burada **actual type parameter** olarak anılmaktadır

Nasıl ki bir metot çağrıldığı vakit asıl argümanlar veriliyor burada da benzer biçimde actual type Parameter, formal type parameter'in yerine konuyor

# Generics ile Beraber List Kullanımı

```
List<Pilot> pilotListesi = new ArrayList<Pilot>();
```

```
pilotListesi.add(new Pilot("Jenson", "Button"));  
pilotListesi.add(new Pilot("Mark", "Webber"));  
pilotListesi.add("Hello World!");  
pilotListesi.add(new Pilot("Sebastian", "Vettel"));  
pilotListesi.add(new Pilot("Jarno", "Trulli"));
```

Hata!

```
Pilot pilot = pilotListesi.get(0);
```

```
for(Pilot p:pilotListesi) {  
    System.out.println(p.getAdi());  
}
```

Derleme zamanında  
hata verir

Explicit cast işlemine gerek  
kalmaz

# Java Generics ve <> Syntax

- Java 7 ile birlikte generic type kullanan sınıftan nesne yaratılırken **actual type parametrelerinin verilmesi** zorunlu değildir

```
List<Integer> intList = new ArrayList<Integer>();
```



```
List<Integer> intList = new ArrayList<>();
```

# Java Generics ve Raw Type

- Generic type bekleyen sınıflar her zaman **actual type parametreleri verilmeden** de kullanılabilir

```
List intList = new ArrayList();
```

- Bu tür kullanıma **raw type** adı verilir
- Java derleyicisi bu tür kullanımda **uyarı verecektir**, ancak **derleme gerçekleşir**
- Böyle bir durumda **generic type'in derleme zamanı avantajlarından mahrum** kalınacaktır



# Java Generics ve Inheritance

```
List<String> strList = new ArrayList<String>();
```

❌ 

```
List<Object> objList = strList;
```

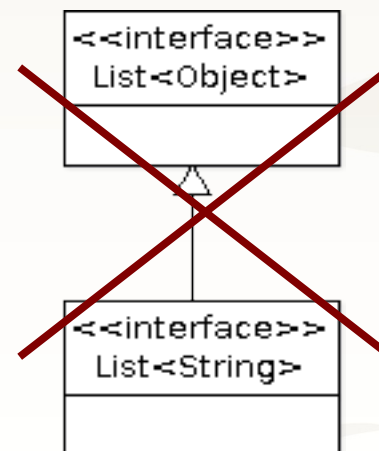
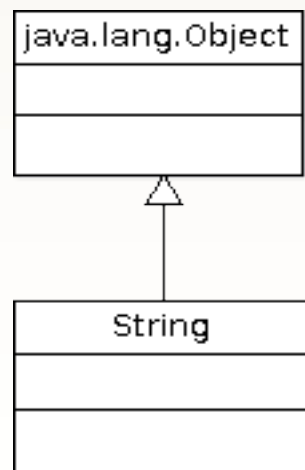
Hata!

Derleyici bu işleme izin vermez

```
objList.add(new Object());
```

```
String str = strList.get(0);
```

Çünkü böyle bir atamaya izin verilse, daha sonra objList'e eklenen bir Object'in, strList'den String olarak alınabilmesi beklenir



- Dolayısı ile generic type parametrelerinde **subtyping çalışmaz**
- Generic type parametreleri **tam olarak birbirleri ile eşleşmelidir**
- Bu durum **generic type'lı metot input parametrelerinde sorun** olmaktadır
- Bu kısıtı aşmak için **wildcard parametreler** kullanılabilir
- Wildcard **“herhangi bir tip ile eşleşebilir”** demektir

# Java Generics ve Inheritance

```
void printCollection(Collection c) {  
    for (Object e : c) {  
        System.out.println(e);  
    }  
}
```

Her tür collection'ın elemanlarını  
print edebilir

```
void printCollection(Collection<Object> c) {  
    for (Object e : c) {  
        System.out.println(e);  
    }  
}
```

Sadece Collection<Object>  
tipinde parametreler kabul  
eder. **Oldukça kısıtlayıcı!**

Wildcard parametre

```
void printCollection(Collection<?> c) {  
    for (Object e : c) {  
        System.out.println(e);  
    }  
}
```

Collection<?> : element tipi  
herhangi bir tipten olabilir  
aslında **“collection of unknown”**  
demektir

# Wildcards

```
List<?> strList = new ArrayList<String>();
```



strList değişkeni, String elemanlardan oluşan bir ArrayList ile initialize edilmiş olsa bile eleman tipi bilinmeyen bir List demektir

```
strList.add(new String("hello world"));
```



```
String str = strList.get(0);
```



Hata!

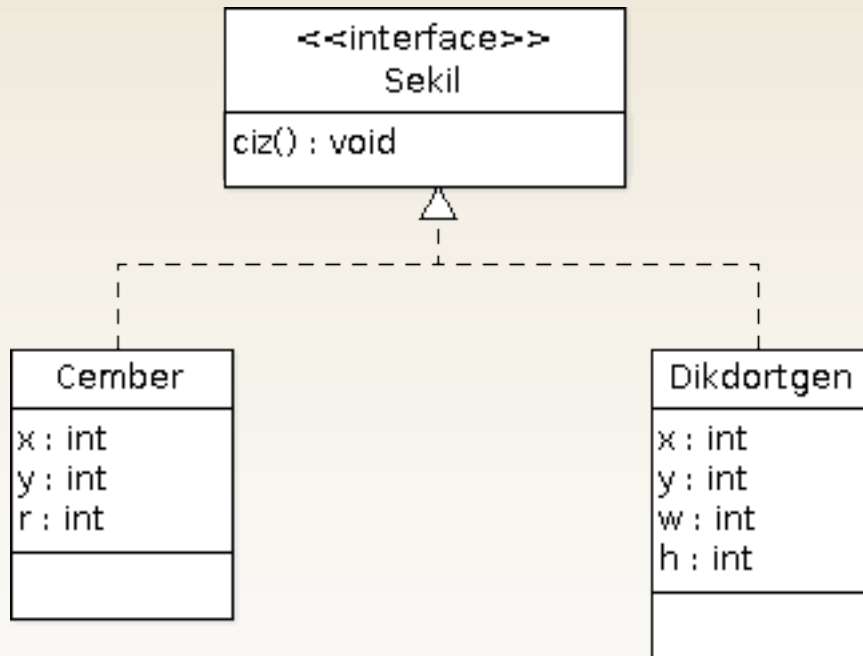
Bu durumda yukarıdaki iki işlem de derleme zamanında **hata** ile sonuçlanacaktır. Çünkü strList'i, içinde tutulabilecek eleman tipini bilmiyoruz diye tanımladık.

```
Object obj = strList.get(0);
```



Öte yandan strList'in herhangi bir elemanını Object tipinde bir değişkene atamakta bir sorun yoktur, çünkü tipini bilmesek bile bütün elemanlar nihayetinde Object'dir

# Bounded Wildcards



```

public void
hepsiniCiz(List<Sekil> sekiller)
{
    for (Sekil s: sekiller) {
        s.ciz();
    }
}
    
```

```

List<Cember> cemberler;
List<Dikdortgen> dikdortgenler;
List<Sekil> sekiller;
    
```



```
hepsiniCiz(cemberler);
```



```
hepsiniCiz(dikdortgenler);
```



```
hepsiniCiz(sekiller);
```



**Derleme hatası!**

# Bounded Wildcards (extends)

```
public void hepsiniCiz(List<? extends Sekil>  
sekilller) {  
    ...  
}
```

**extends** ile “unknown element type”,  
aslında Sekil tipinin bir subclass'ıdır  
 demiş oluyoruz

Bu durumda Sekil ve Sekil'den türeyen  
bütün alt sınıfları içeren generic list'ler  
input argüman olabilir

“upper bound”  
konumundadır



**extends** ile tanımlanan bir generic collection'dan  
sadece okuma yapılabilir. Object dahil hiçbir nesne  
Collection'a eklenemez. Sadece null değer eklenebilir.

# Bounded Wildcards (extends)

```
public void hepsiniCiz(List<? extends Sekil> sekilller) {  
    for (Sekil s: sekilller) {  
        s.ciz();  
    }  
}
```

```
List<Cember> cemberler;  
List<Dikdortgen> dikdortgenler;  
List<Sekil> sekilller;
```

...

✓ hepsiniCiz(cemberler);

✓ hepsiniCiz(dikdortgenler);

✓ hepsiniCiz(sekilller);

# Bounded Wildcards (super)

```
List<? super Sekil> sekiller = new ArrayList<>();
```

“lower bound” konumundadır

**super** ile “unknown element type”  
Sekil sınıfı veya bunun üst  
tiplerinden birisidir demiş oluyoruz



**super** ile tanımlanan generic collection sadece ekleme yapmak için uygundur. Eğer okuma yapılmak istenirse dönen nesnelerin tipi Object olacaktır



```
sekiller.add(cember);
```



```
sekiller.add(dikdortgen);
```



```
Sekil sekil = sekiller.get(0);
```



# Bounded Wildcards

- Collection'larda **hem okuma hem de yazma** yapılacak ise **wildcard kullanılmamalıdır!**
- Çünkü **? extends T** ile yapılan Collection'lardan sadece okuma yapılabilir
- **NULL dışında** herhangi bir eleman bu şekilde tanımlanmış Collection'a **eklenemez**
- **? super T** ile tanımlanan Collection'larda ise sadece T ve T'nin super tiplerinde eleman eklemek pratiktir
- Bu şekilde tanımlanmış Collection'dan da elemanlar **sadece Object** olarak **dönülecektir**

# Collectiondaki Elemanlara Erişim ve Eleman Ekleme

```
public class Converter {  
    public void fromArrayToCollection(Object[] a, Collection<?> c) {  
        for (Object o : a) {  
            c.add(o);  
        }  
    }  
}
```

❌

Hata!

Derleme hatası verir, çünkü Object'i "unknown tip" tutan bir collection'a eklemeye çalışıyoruz. Hem okuma hem de ekleme yapılacak ise wildcard kullanılmamalıdır.

```
public class Converter<T> {  
    public void fromArrayToCollection(T[] a, Collection<T> c) {  
        for (T o : a) {  
            c.add(o);  
        }  
    }  
}
```

✓

Generic type parameter sınıf içerisinde metot parametrelerinin ve değişkenlerin tipi olarak kullanılabilir

# Birden Fazla Generic Tip Tanımı

```
public class Converter<T,S extends T> {  
    public void copy(List<S> src, List<T> dest){  
        for(S s:src) {  
            dest.add(s);  
        }  
    }  
}
```

Birden fazla formal parameter type tanımlamak da mümkündür

```
List<Cember> srcList = new ArrayList<>();  
List<Sekil> targetList = new ArrayList<>();
```

```
srcList.add(new Cember());  
srcList.add(new Cember());
```

```
Converter<Sekil, Cember> converter = new Converter<>();  
converter.copy(srcList,targetList);
```

# Generic Metotlar

```
public class Converter {
    public <T> void fromArrayToCollection(T[] a, Collection<T> c) {
        for (T o : a) {
            c.add(o);
        }
    }
}
```

İstenirse sınıf düzeyinde değil, metot düzeyinde de Generic tip tanımlanabilir

```
List<String> strList = new ArrayList<String>();
```

```
Converter c = new Converter();
```

```
c.fromArrayToCollection(new String[]{"a", "b", "c"}, strList);
```

# Generic Tiplerin Kısıtları

- Java derleyicisi generic tiplerde **“type erasure”** uygulamaktadır
- Yani derleme sırasında **generic tipler concrete tipler ile yer değiştirmektedir**
- Sonuç olarak **generic tip bilgisi çalışma zamanında kaybolmaktadır**
- Type erasure nedeni ile **ortaya çıkan bazı kısıtlar söz konusudur**

# Generic Tiplerin Kısıtları

`T var = new T();`



`T[] varArr = new T[10];`



`public class Foo<T> {`

`T instanceVar;`



`static T staticVar;`



`}`

`class MyException<T>  
    extends Throwable {  
}`



`List<int> intList;`



`myVar instanceof(T)`



# İletişim



[www.harezmi.com.tr](http://www.harezmi.com.tr)

[www.java-egitimleri.com](http://www.java-egitimleri.com)



[info@harezmi.com.tr](mailto:info@harezmi.com.tr)

[info@java-egitimleri.com](mailto:info@java-egitimleri.com)



[@HarezmiBilisim](https://twitter.com/HarezmiBilisim)

[@JavaEgitimleri](https://twitter.com/JavaEgitimleri)