JAVA GENERICS

TİP GÜVENLİ PROGRAMLAMA

Generic Classes, Methods, Wildcards, Bounded Types

GİRİŞ: GENERICS NEDİR?

Generics Nedir?

Generics = Tip parametreleri kullanarak esnek ama güvenli kod yazmak

Gerçek hayattan örnek:

- **Generic olmayan kutu:** "Bu kutuda ne var bilmiyorum, açıp bakmam lazım"
- **Generic kutu:** "Bu kutu kesinlikle kitap içeriyor, direkt alabilirsin"

Sorun (Generics Olmadan):

```
// Eski yol - Object kullanımı
ArrayList liste = new ArrayList();
liste.add("Merhaba");
liste.add(123);
liste.add(true);

// Çıkarırken tip kontrolü yok!
String metin = (String) liste.get(0);  // Manuel casting
String metin2 = (String) liste.get(1); // RUNTIME HATASI!
```

Çözüm (Generics İle):

```
// Yeni yol - Generics kullanımı
ArrayList<String> liste = new ArrayList<>();
liste.add("Merhaba");
// liste.add(123);     // DERLEME HATASI! Sadece String
// liste.add(true);    // DERLEME HATASI!

String metin = liste.get(0);  // Casting gereksiz, güvenli!
```

1. GENERICS OLMADAN SORUNLAR

1.1 Sorun 1: Tip Güvenliği Yok

```
public class GenericsOlmadan {
    public static void main(String[] args) {

        // Object tipinde ArrayList - her şeyi kabul eder
        ArrayList liste = new ArrayList();

        liste.add("Ali");
        liste.add("Ayşe");
        liste.add(123);        // Sayı da eklenebilir!
        liste.add(true);       // Boolean da!
        liste.add(new Date()); // Tarih de!

        // Problem: Çıkarırken ne olduğunu bilmiyoruz
        for (Object obj : liste) {
            String isim = (String) obj;  // İlk 2'si çalışır
            System.out.println(isim);     // 3. eleman HATA!
        }
    }
}
```

Çıktı:

```
Ali
Ayşe
Exception in thread "main" java.lang.ClassCastException:
    java.lang.Integer cannot be cast to java.lang.String
```

Sorun: Hata çalışma zamanında (runtime) ortaya çıkıyor! Geç fark ediliyor.

1.2 Sorun 2: Sürekli Casting (Tür Dönüşümü)

```
public class CastingSorunu {
    public static void main(String[] args) {

        ArrayList liste = new ArrayList();
        liste.add("Java");
        liste.add("Python");
        liste.add("JavaScript");

        // Her get işleminde casting gerekli
        String dil1 = (String) liste.get(0);  // Zahmetli
        String dil2 = (String) liste.get(1);  // Tekrarlıyor
        String dil3 = (String) liste.get(2);  // Can sıkıcı

        // Döngüde de aynı sorun
        for (Object obj : liste) {
            String dil = (String) obj;  // Her seferinde casting
            System.out.println(dil.toUpperCase());
        }
```

```
    }
}
```

2. GENERICS İLE ÇÖZÜM

2.1 Basit Generic Kullanımı

```
public class GenericsIle {
    public static void main(String[] args) {

        // <String> = Bu liste sadece String tutar
        ArrayList<String> liste = new ArrayList<>();

        liste.add("Ali");
        liste.add("Ayşe");
        // liste.add(123);     // DERLEME HATASI! ✅
        // liste.add(true);    // DERLEME HATASI! ✅

        // Casting gereksiz, direkt String döner
        String isim = liste.get(0);  // Temiz kod!
        System.out.println(isim.toUpperCase());

        // Döngüde de casting yok
        for (String ad : liste) {
            System.out.println(ad.length());
        }
    }
}
```

Avantajlar:

- Derleme zamanında hata - Erken fark edilir
- Casting gereksiz - Temiz kod
- Tip güvenli - Yanlış tip eklenemez

2.2 Farklı Tipler ile Generics

```
public class FarkliTipler {
    public static void main(String[] args) {

        // String listesi
        ArrayList<String> isimler = new ArrayList<>();
        isimler.add("Ali");
        isimler.add("Ayşe");

        // Integer listesi
        ArrayList<Integer> sayilar = new ArrayList<>();
        sayilar.add(10);
        sayilar.add(20);

        // Double listesi
        ArrayList<Double> fiyatlar = new ArrayList<>();
        fiyatlar.add(19.99);
        fiyatlar.add(29.99);
```

```
        // Her liste tip güvenli!
        String ilkIsim = isimler.get(0);      // String döner
        Integer ilkSayi = sayilar.get(0);     // Integer döner
        Double ilkFiyat = fiyatlar.get(0);    // Double döner
    }
}
```

3. GENERIC CLASS OLUŞTURMA

3.1 Generic Class Nedir?

Kendi generic sınıflarınızı oluşturabilirsiniz!

```
// Generic class - T herhangi bir tip olabilir
public class Kutu<T> {

    private T icerik;

    public Kutu(T icerik) {
        this.icerik = icerik;
    }

    public T getIcerik() {
        return icerik;
    }

    public void setIcerik(T icerik) {
        this.icerik = icerik;
    }

    public void bilgiGoster() {
        System.out.println("Kutu içeriği: " + icerik);
        System.out.println("Tip: " +
            icerik.getClass().getSimpleName());
    }
}
```

Kullanım:

```
public class KutuTest {
    public static void main(String[] args) {

        // String kutusu
        Kutu<String> stringKutu = new Kutu<>("Merhaba");
        stringKutu.bilgiGoster();
        String mesaj = stringKutu.getIcerik();

        // Integer kutusu
        Kutu<Integer> intKutu = new Kutu<>(42);
        intKutu.bilgiGoster();
        Integer sayi = intKutu.getIcerik();

        // Double kutusu
        Kutu<Double> doubleKutu = new Kutu<>(3.14);
        doubleKutu.bilgiGoster();
        Double pi = doubleKutu.getIcerik();
    }
}
```

3.2 Generic Pair (Çift) Örneği

```
// İki farklı tipte değer tutan generic sınıf
public class Pair<K, V> {

    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() { return key; }
    public V getValue() { return value; }

    @Override
    public String toString() {
        return "(" + key + ", " + value + ")";
    }
}

// Kullanım
Pair<String, Integer> kisi = new Pair<>("Ali", 25);
Pair<String, Double> urun = new Pair<>("Laptop", 15000.0);
Pair<Integer, String> ogrenci = new Pair<>(1001, "Ayşe");
```

4. GENERIC METODLAR

4.1 Generic Metod Nedir?

Metodun kendisi generic olabilir (sınıf generic olmasa bile).

```
public class GenericMetodlar {

    // Generic metod - herhangi bir tipte dizi yazdırır
    public static <T> void diziYazdir(T[] dizi) {
        System.out.print("[");
        for (int i = 0; i < dizi.length; i++) {
            System.out.print(dizi[i]);
            if (i < dizi.length - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }

    // Generic metod - iki değeri karşılaştırır
    public static <T> boolean esitMi(T deger1, T deger2) {
        return deger1.equals(deger2);
    }

    public static void main(String[] args) {

        // String dizisi
        String[] isimler = {"Ali", "Ayşe", "Mehmet"};
        diziYazdir(isimler);

        // Integer dizisi
        Integer[] sayilar = {10, 20, 30, 40};
        diziYazdir(sayilar);

        // Karşılaştırma
        System.out.println(esitMi("Ali", "Ali"));  // true
        System.out.println(esitMi(10, 20));           // false
    }
}
```

4.2 Dizide Arama Örneği

```
public class DiziIslemleri {

    // Generic metod - dizide eleman arama
    public static <T> boolean iceriyorMu(T[] dizi, T aranan) {
        for (T eleman : dizi) {
            if (eleman.equals(aranan)) {
                return true;
            }
```

```
        }
        return false;
    }

    // Generic metod - dizide index bulma
    public static <T> int indexBul(T[] dizi, T aranan) {
        for (int i = 0; i < dizi.length; i++) {
            if (dizi[i].equals(aranan)) {
                return i;
            }
        }
        return -1;  // Bulunamadı
    }

    public static void main(String[] args) {
        String[] meyveler = {"Elma", "Armut", "Muz", "Kiraz"};

        System.out.println("'Elma' var mı? " +
            iceriyorMu(meyveler, "Elma"));
        System.out.println("'Üzüm' var mı? " +
            iceriyorMu(meyveler, "Üzüm"));

        System.out.println("'Armut' indexi: " +
            indexBul(meyveler, "Armut"));
    }
}
```

5. BOUNDED TYPE PARAMETERS

5.1 Bounded Types Nedir?

Generic tipi sınırlayabilirsiniz - "sadece Number veya alt sınıfları" gibi.

```
// T sadece Number veya alt sınıfları olabilir
public class SayiKutusu<T extends Number> {

    private T sayi;

    public SayiKutusu(T sayi) {
        this.sayi = sayi;
    }

    public T getSayi() {
        return sayi;
    }

    // Number metodlarını kullanabiliriz
    public double doubleValue() {
        return sayi.doubleValue();
    }

    public int intValue() {
        return sayi.intValue();
    }
}
```

Kullanım:

```
// Integer - Number'ın alt sınıfı ✅
SayiKutusu<Integer> intKutu = new SayiKutusu<>(42);

// Double - Number'ın alt sınıfı ✅
SayiKutusu<Double> doubleKutu = new SayiKutusu<>(3.14);

// Long - Number'ın alt sınıfı ✅
SayiKutusu<Long> longKutu = new SayiKutusu<>(1000000L);

// String - Number'ın alt sınıfı DEĞİL ❌
// SayiKutusu<String> stringKutu = new SayiKutusu<>("test");
// DERLEME HATASI!
```

5.2 Bounded Generic Metod

```
public class MatematikIslemleri {

    // Sadece Number ve alt sınıflarını kabul eder
    public static <T extends Number> double toplam(T[] sayilar) {
        double sum = 0;
        for (T sayi : sayilar) {
```

```
            sum += sayi.doubleValue();
        }
        return sum;
    }

    public static <T extends Number> double ortalama(T[] sayilar) {
        return toplam(sayilar) / sayilar.length;
    }

    public static void main(String[] args) {

        Integer[] intDizi = {10, 20, 30, 40, 50};
        System.out.println("Toplam: " + toplam(intDizi));
        System.out.println("Ortalama: " + ortalama(intDizi));

        Double[] doubleDizi = {1.5, 2.7, 3.9, 4.1};
        System.out.println("Toplam: " + toplam(doubleDizi));
        System.out.println("Ortalama: " + ortalama(doubleDizi));
    }
}
```

6. WILDCARDS (JOKER KARAKTERLER)

6.1 Wildcard Nedir?

Wildcard (?) = "Bilinmeyen tip" - Esneklik sağlar

3 Çeşit Wildcard:

- **Unbounded:** ? - Herhangi bir tip
- **Upper Bounded:** ? extends Type - Type veya alt sınıfları
- **Lower Bounded:** ? super Type - Type veya üst sınıfları

6.2 Unbounded Wildcard

```
import java.util.*;

public class UnboundedWildcard {

    // Herhangi bir tipte listeyi yazdırır
    public static void listeYazdir(List<?> liste) {
        for (Object eleman : liste) {
            System.out.print(eleman + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {

        // String listesi
        List<String> isimler = new ArrayList<>();
        isimler.add("Ali");
        isimler.add("Ayşe");
        listeYazdir(isimler);

        // Integer listesi
        List<Integer> sayilar = new ArrayList<>();
        sayilar.add(10);
        sayilar.add(20);
        listeYazdir(sayilar);

        // Double listesi
        List<Double> fiyatlar = new ArrayList<>();
        fiyatlar.add(19.99);
        listeYazdir(fiyatlar);
    }
}
```

6.3 Upper Bounded Wildcard

? extends Type = Type veya alt sınıfları

```
public class UpperBoundedWildcard {
```

```
    // Sadece Number ve alt sınıfları
    public static double toplam(List<? extends Number> liste) {
        double sum = 0;
        for (Number sayi : liste) {
            sum += sayi.doubleValue();
        }
        return sum;
    }

    public static void main(String[] args) {

        // Integer listesi - Number'ın alt sınıfı ✅
        List<Integer> intListe = new ArrayList<>();
        intListe.add(10);
        intListe.add(20);
        System.out.println("Toplam: " + toplam(intListe));

        // Double listesi - Number'ın alt sınıfı ✅
        List<Double> doubleListe = new ArrayList<>();
        doubleListe.add(1.5);
        doubleListe.add(2.7);
        System.out.println("Toplam: " + toplam(doubleListe));

        // String listesi - Number DEĞİL ❌
        // List<String> stringListe = new ArrayList<>();
        // toplam(stringListe);  // DERLEME HATASI!
    }
}
```

6.4 Lower Bounded Wildcard

? super Type = Type veya üst sınıfları

```
public class LowerBoundedWildcard {

    // Integer veya üst sınıflarını kabul eder
    public static void integerEkle(List<? super Integer> liste) {
        liste.add(10);
        liste.add(20);
        liste.add(30);
    }

    public static void main(String[] args) {

        // Integer listesi - Integer'ın kendisi ✅
        List<Integer> intListe = new ArrayList<>();
        integerEkle(intListe);
        System.out.println("Integer: " + intListe);

        // Number listesi - Integer'ın üst sınıfı ✅
        List<Number> numberListe = new ArrayList<>();
        integerEkle(numberListe);
```

```
        System.out.println("Number: " + numberListe);

        // Object listesi - Integer'ın üst sınıfı ✅
        List<Object> objectListe = new ArrayList<>();
        integerEkle(objectListe);
        System.out.println("Object: " + objectListe);
    }
}
```

7. GENERIC INTERFACE

7.1 Generic Interface Oluşturma

```
// Generic interface
public interface Depo<T> {
    void ekle(T eleman);
    T cikar();
    T gor();
    boolean bosMu();
    int boyut();
}

// String depo implementasyonu
class StringDepo implements Depo<String> {

    private List<String> liste = new ArrayList<>();

    @Override
    public void ekle(String eleman) {
        liste.add(eleman);
    }

    @Override
    public String cikar() {
        if (!liste.isEmpty()) {
            return liste.remove(liste.size() - 1);
        }
        return null;
    }

    @Override
    public String gor() {
        if (!liste.isEmpty()) {
            return liste.get(liste.size() - 1);
        }
        return null;
    }

    @Override
    public boolean bosMu() {
        return liste.isEmpty();
    }

    @Override
    public int boyut() {
        return liste.size();
    }
}
```

8. GENERIC STACK ÖRNEĞİ

8.1 Tam Kapsamlı Stack

```
public class Stack<T> {

    private ArrayList<T> elemanlar;
    private int maxBoyut;

    public Stack() {
        this.elemanlar = new ArrayList<>();
        this.maxBoyut = Integer.MAX_VALUE;
    }

    // PUSH - Eleman ekle
    public void push(T eleman) {
        if (isFull()) {
            throw new IllegalStateException("Stack dolu!");
        }
        elemanlar.add(eleman);
    }

    // POP - Eleman çıkar
    public T pop() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack boş!");
        }
        return elemanlar.remove(elemanlar.size() - 1);
    }

    // PEEK - En üsttekini gör
    public T peek() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack boş!");
        }
        return elemanlar.get(elemanlar.size() - 1);
    }

    public boolean isEmpty() {
        return elemanlar.isEmpty();
    }

    public boolean isFull() {
        return elemanlar.size() >= maxBoyut;
    }

    public int size() {
        return elemanlar.size();
    }
}
```

Kullanım:

```
// String Stack
Stack<String> kitaplar = new Stack<>();
kitaplar.push("Java Programlama");
kitaplar.push("Python Temelleri");
kitaplar.push("JavaScript Guide");

System.out.println("Boyut: " + kitaplar.size());
System.out.println("En üstteki: " + kitaplar.peek());
kitaplar.pop();

// Integer Stack
Stack<Integer> sayilar = new Stack<>();
sayilar.push(10);
sayilar.push(20);
sayilar.push(30);
```

9. BEST PRACTICES

9.1 İsimlendirme Kuralları

Yaygın kullanılan tip parametreleri:

```
E - Element (koleksiyonlarda)
class ArrayList<E> { }

K - Key, V - Value (map'lerde)
class HashMap<K, V> { }

T - Type (genel amaçlı)
class Box<T> { }

N - Number (sayılar için)
class Calculator<N extends Number> { }

S, U, V - İkinci, üçüncü, dördüncü tipler
class Triple<T, S, U> { }
```

9.2 Ne Zaman Generics Kullanılır?

Kullan:

- Koleksiyonlarda
- Tip güvenliği gerektiğinde
- Yeniden kullanılabilir kodlarda

Kullanma:

- Basit, tek kullanımlık kodlarda
- Gereksiz karmaşıklık yaratacaksa

9.3 Yaygın Hatalar

```
❌ HATA 1: Primitive tiplerle generic
 ArrayList<int> sayilar;  // DERLEME HATASI!
 ✅ ÇÖZÜM: Wrapper sınıfları kullan
ArrayList<Integer> sayilar = new ArrayList<>();

❌ HATA 2: Generic dizi oluşturma
 T[] dizi = new T[10];  // DERLEME HATASI!
 ✅ ÇÖZÜM:
@SuppressWarnings("unchecked")
T[] dizi = (T[]) new Object[10];

❌ HATA 3: Static generic alan
 static T eleman;  // DERLEME HATASI!
 ✅ ÇÖZÜM: Static metod kullan
public static <T> void metod(T eleman) { }
```

SONUÇ

Bu dokümanda Java Generics'in tüm temel konularını detaylı olarak inceledik:

- **Generic Classes:** Tip güvenli sınıflar oluşturma
- **Generic Methods:** Esnek ve yeniden kullanılabilir metodlar
- **Bounded Types:** Tip parametrelerini sınırlama
- **Wildcards:** Esneklik sağlayan joker karakterler
- **Best Practices:** En iyi uygulamalar ve yaygın hatalar

Generics, Java'da tip güvenli ve esnek kod yazmanın temelidir. Doğru kullanıldığında, kodunuz daha güvenilir, okunabilir ve bakımı kolay olur.

ÖNEMLİ HATIRLATMALAR

- Koleksiyonlarda her zaman generic kullanın
- Primitive tipler yerine wrapper sınıfları kullanın
- Bounded types ile tip güvenliğini artırın
- PECS kuralını hatırlayın: Producer Extends, Consumer Super
- Gereksiz karmaşıklık yaratmayın