

Generics

- Methodlar

- Java'da bir methodun belirli bir veri türünde bağlı olmadan çalışmasını sağlar.
- Örneğin

Public Static <T> void printArr(T[] arr)

gibi bir method tanımladığımızda T
türü sayesinde herhangi bir türdeki dizi
(String, Integer, Double, Short vb.)
bu methoda parametre olarak verilebilir.

- Kodun yeniden kullanılabilirliğini ve esnekliğini artttırır.

Class'lar

- Bir sınıfın herhangi bir türle çalışabilmesini sağlamak için kullanılır.

Örneğin :

Public Class Box <T>{...}

şeklinde bir sınıf tanımlanabilir.

- Bu sayede Box <Integer> veya Box<String>
gibi farklı türler için aynı sınıf kullanarak
nesneler oluşturabilirsiniz.

- Verilerin tur güvenliğini sağlar ve aynı kodu tekrar tekrar yazmak zorunda kalmazsınız.

- Bounding

- Generic türleri kısıtlamanızı sağlar. Örneğin sadece "Number" sınıfından türeyen veri türleri ile çalışmak istiyorsanız;

public class Box < T extends Number >

seklinde bir tanımlama yapabilirsiniz.

- Üst sınırlama (Upper bound) ile belirli sınırlardan veya interface'lerden türeyen türleri kısıtlayarak, methods veya sınıflar belirli işlevselliliklerde kullanılması sağlanır.
- Alt sınırlama (lower bounding) ile belirli bir türün super sınıfları veya interface'leriyle uyumlu hale getirilebilir.

- Interface'ler

Neden Generic Interface Kullanınız?

- Generic Interface'ler bir interface'in birden fazla türde çalışmasına olanak tanır. Bu sayede de aynı interface'i farklı veri türleriyle kullanabiliriz. ve veri türlerine göre farklı davranışları tanımlayabiliriz.

Generic Interface Tanımı

Bir generic interface tanımlamak için interface isminin yanına $\langle T \rangle$ gibi bir tür parametresi eklenir. Bu tür parametresi interface'nin tür bağımsızlığını sağlar.

Örneğin bir repository $\langle T \rangle$ interface'si düşünelim. Bu interface herhangi bir veri türüyle çalışabilen bir veri deposu gibi davranabilir.

A screenshot of a code editor window. The title bar says "java". The code area contains the following Java code:

```
public interface Repository<T> {
    void save(T item);      // T türünde bir öğe kaydet
    T find(int id);        // T türünde bir öğe bul
}
```

The code is syntax-highlighted, with "public", "interface", "Repository", "void", "save", "T", "item", "find", "int", and "id" in blue, and the type parameters and method bodies in black.

Bu generic interface bir $\text{Repository} \langle T \rangle$ olarak tanımlandığından T türü farklı veri türleri için uygulanabilir. Örneğin;

$\text{Repository} \langle \text{String} \rangle$ ve $\text{Repository} \langle \text{Integer} \rangle$ olarak kullanılabilir.

-Generic Interface Kullanımı

Generic Interface kullanırken tür parametresini belirleyerek bu interface'i implement eden sınıfları esnek hale getirebiliriz.

java

Kodu kopyala

```
public class Product {  
    private int id;  
    private String name;  
    // getter, setter ve diğer metodlar  
}  
  
public class User {  
    private int id;  
    private String username;  
    // getter, setter ve diğer metodlar  
}
```

Şimdi Repository<T> interface'sını implement eden iki sınıfı yazalım.

ProductRepository ve UserRepository

java

Kodu kopyala

```
public class ProductRepository implements Repository<Product> {  
  
    @Override  
    public void save(Product item) {  
        System.out.println("Product saved: " + item.getName());  
    }  
  
    @Override  
    public Product find(int id) {  
        // Sadece örnek için basit bir ürün döndürüyoruz  
        return new Product(); // Gerçek veri tabanı işlemleri burada olabilir  
    }  
}  
  
public class UserRepository implements Repository<User> {  
  
    @Override  
    public void save(User item) {  
        System.out.println("User saved: " + item.getUsername());  
    }  
  
    @Override  
    public User find(int id) {  
        return new User(); // Gerçek veri tabanı işlemleri burada olabilir  
    }  
}
```

ProductRepository yalnızca Product türüyle çalışırken UserRepository yalnızca User türüyle çalışır.

Böylece Repository<T> interface'si farklı, veri türleriyle çalışır hale gelmiştir.

5. Sık Kullanılan Generic Interfaceler

Java'nın kendisinde sıkça kullanılan bazı generic interfaceler vardır:

- Comparable<T>: Nesnelerin sıralanmasını sağlamak için kullanılır. compareTo metodu, bir nesnenin diğer nesneye göre sıralama konumunu belirler.
- Comparator<T>: İki nesneyi karşılaştırmak için kullanılır. Özellikle özel sıralama gereğinde çok kullanışlıdır.
- Iterable<T>: Koleksiyonların foreach döngüsünde kullanılmasını sağlar. Collection, List, Set, Queue gibi sınıflar Iterable<T> interface'ini implement eder.

Yapının Detayları

Interface (Repository<T>)

- Generic yapıyı tanımlar ve genel davranışları (metot imzalarını) belirtir.
- Tür bağımsızdır, yani her türde çalışabilir (<T> ile gösterilir.)

Model Sınıfları (Product ve User)

- Bu sınıflar interface'in hangi türle çalışacağıını belirler ve farklı veri modellerini

tanımlar

- Product ve User burada bağımsız ve kendi işlevleri olan sınıflarıdır.

Interface'i Implement Eder Sınıflar

(ProductRepository ve UserRepository)

- Repository interface'ini implement eder ve Product ile User sınıflarına özgü davranışları sağlar.
- Her biri kendi türne özel işlem ve veri yönetimi uygulamalarını icerir.

Ana Sınıf Main

Programın çalıştırıldığı sınıfıdır. Burada ProductRepository ve UserRepository örnekleri oluşturulur ve methodlar çağırılarak veri işleme yapılır.

```
java Kodu kopyala

public class Main {
    public static void main(String[] args) {
        Repository<Product> productRepository = new ProductRepository();
        productRepository.save(new Product(1, "Laptop"));

        Repository<User> userRepository = new UserRepository();
        userRepository.save(new User(1, "john_doe"));
    }
}
```

Class Kullanımıyla Bir Örnek

 Kodu kopyala

```
public class Box<T> {  
    private T content;  
  
    // İçeriği ayarlayan constructor  
    public Box(T content) {  
        this.content = content;  
    }  
  
    // İçeriği döndüren getter metodu  
    public T getContent() {  
        return content;  
    }  
  
    // İçeriği ayarlayan setter metodu  
    public void setContent(T content) {  
        this.content = content;  
    }  
  
    // İçeriği yazdırın metot  
    public void printContent() {  
        System.out.println("Box içeriği: " + content);  
    }  
}
```



java

 Kodu kopyala

```
public class Main {  
    public static void main(String[] args) {  
        // String türünde bir Box oluştur  
        Box<String> stringBox = new Box<>("Merhaba");  
        stringBox.printContent(); // Çıktı: Box içeriği: Merhaba  
  
        // Integer türünde bir Box oluştur  
        Box<Integer> integerBox = new Box<>(123);  
        integerBox.printContent(); // Çıktı: Box içeriği: 123  
  
        // İçeriği değiştirme  
        integerBox.setContent(456);  
        integerBox.printContent(); // Çıktı: Box içeriği: 456  
    }  
}
```


