Java Generics

Generics parametreleştirilmiş tipler anlamına gelir ve sınıf, interface veya metod yazarken tek bir data tipine bağlı kalmadan üzerinde işlem yapacağımız data tipini parametre olarak alabilmemizi sağlar. Bu sayede, farklı data tipleri üzerinde çalışan tek bir sınıf, metod veya interface yazmak mümkündür.
Bu şekilde yazılan sınıflara **generic sınıf**, metotlara **generic metod** denir.

Generics hataları azaltmak ve tip güvenliğini (type-safety) sağlamak amacıyla JDK 5 ile eklenmiştir. Buna rağmen, Java'nın en temel özelliklerinden biridir ve dili temelden etkilemiştir. Bu yüzden, jenerikleri iyi anlamak Java'yı öğrenmek açısından büyük önem taşır.

Generic class, metod veya interface oluştururken data tipini parametre olarak almak için elmas(diamond) işareti <> kullanılır ve çoklu parametre(multiple parameter) kullanılabilir. Parametreler genelde T, S, U, V gibi harfleri ile gösterilir.

Genericsde primitive(ilkel) data tipleri parametre olarak kullanılamaz.

Generic Class

Generic bir sınıfın genel tanımı aşağıdaki gibidir:

```
class class_name<T>
{
sınıf değişkenleri;
```

.....
sınıf metodları;
}

Sınıf tanımlandıktan sonra, istediğiniz herhangi bir data tipinde aşağıdaki gibi nesneler oluşturabilirsiniz:

class_name <T> obj = new class_name <T> ();

Örneğin, integer data tipi için nesne şöyle olacaktır:

class_name <Integer> obj = new class_name<Integer>;

Generic Metod

Generics ile farklı data tiplerini parametre olarak alan ve döndüren metod yazabilirsiniz.

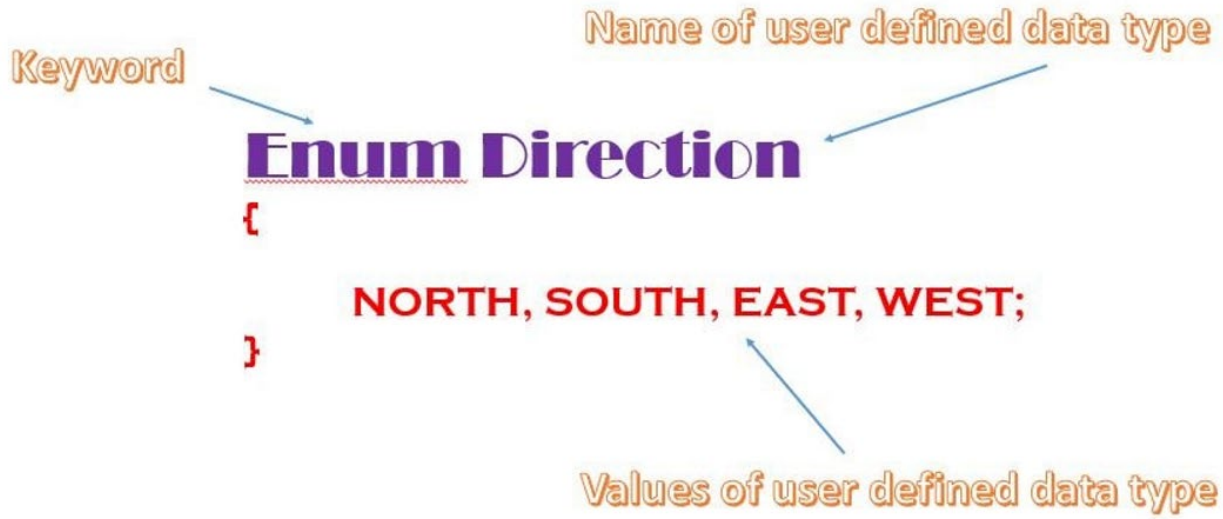
```
public static <T> void yazdir(T type){
   System.out.print(type);
}
```

Generics ile data tipleri üst veya alt tip sınırlanabilir.

Java Generics Wildcard

Javada, wildcard(?) genelde genericsde parametre olarak bilinmeyen data tiplerine atıfta bulunmak için kullanılır.

Java Enums



Enum türü , bir değişkenin önceden tanımlanmış bir sabitler kümesi olmasını sağlayan özel bir veri türüdür. Değişken, kendisi için önceden tanımlanmış değerlerden birine eşit olmalıdır.

Yaygın örnekler arasında pusula yönleri (KUZEY, GÜNEY, DOĞU ve BATI değerleri) ve haftanın günleri yer alır.
Sabit olduklarından, bir enum türünün alanlarının adları büyük harflerle yazılır.
Java programlama dilinde, "enum" anahtar kelimesini kullanarak bir enum türü tanımlarsınız. Örneğin, haftanın günleri numaralandırma türünü şu şekilde belirtirsiniz:

```
public enum Days {
   PAZAR,
       PAZARTESI,
       SALI,
       CARSAMBA,
   PERSEMBE,
       CUMA,
       CUMARTESI
}
```

Yukarıda tanımlanan Days enumını nasıl kullanacağınızı gösteren bazı kodlar :

```
public class EnumTest {
   Days days;
```

```
public EnumTest(Days days) {
    this.days = days;
}

public void printDays() {
    switch (days) {
        case PAZARTESI:
            System.out.println("Pazartesi günleri pek sevilmez.");
            break;
        case CUMA:
            System.out.println("Cuma günleri daha iyidir.");
            break;
        case CUMATESI: case PAZAR:
            System.out.println("Hafta sonları en iyisidir.");
            break;
        default:
            System.out.println("Diğer günler şöyle böyledir.");
            break;
    }
 }

    public static void main(String[] args) {
    EnumTest firstDay = new EnumTest(Days.PAZARTESI);
    firstDay.printDays();
    EnumTest thirdDay = new EnumTest(Days.CARSAMBA);
    thirdDay.printDays();
    EnumTest fifthDay = new EnumTest(Days.CUMA);
    fifthDay.printDays();
    EnumTest sixthDay = new EnumTest(Days.CUMARTESI);
    sixthDay.printDays();
    EnumTest seventhDay = new EnumTest(Days.PAZAR);
    seventhDay.printDays();
  }
}
```

Çıktı:

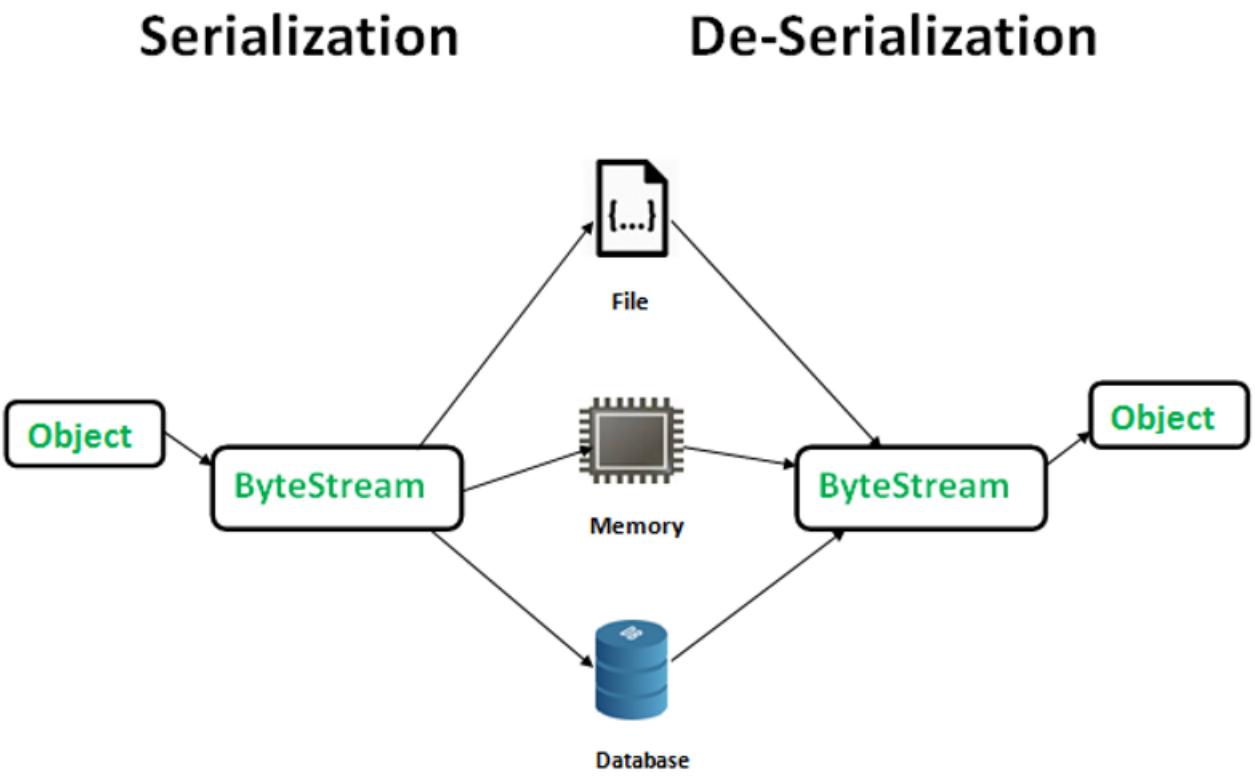
Pazartesi günleri pek sevilmez.

Diğer günler şöyle böyledir.
Cuma günleri daha iyidir.
Hafta sonları en iyisidir.
Hafta sonları en iyisidir.

Java programlama dili enum türleri, diğer dillerdeki karşılıklarından çok daha güçlüdür. "enum" deklarasyonu bir sınıf tanımlar. Enum sınıfı gövdesi, metodlar ve değişkenler içerebilir. Derleyici, bir enum oluşturulduğunda bazı özel metodları otomatik olarak ekler.

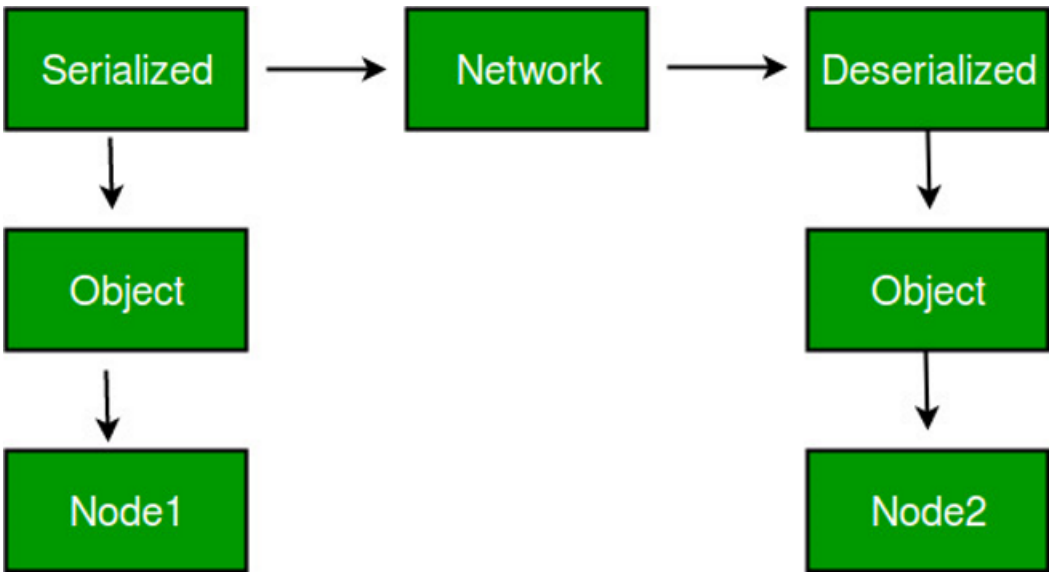
Serialization ve De-Serialization

Serialization(Serileştirme), bir nesnenin durumunu bir byte akışına dönüştürme mekanizmasıdır. De-Serialization(Seri kaldırma), byte akışının gerçek Java nesnesini bellekte yeniden oluşturmak için kullanıldığı ters işlemdir. Bu mekanizma, nesneyi kalıcı kılmak için kullanılır.



Oluşturulan byte akışı platformdan bağımsızdır. Böylece, bir platformda serileştirilen nesne, farklı bir platformda seri hale getirilebilir.
Bir Java nesnesini seri hale getirilebilir yapmak için java.io.Serializable arayüzünü uyguluyoruz.

Serileştirmenin Avantajları
1. Bir nesnenin durumunu kaydetmek/devam ettirmek.
2. Bir ağda bir nesneyi dolaşmak.



Yalnızca java.io.Serializable interfaceni uygulayan bu sınıfların nesneleri serileştirilebilir .

Java Multithreading

Thread nedir?

Her bir işlemin altında çalışan alt işlemlere "thread" adı verilir. Aynı anda birden fazla işlem yapmayı sağlayan yapıya thread denir. Bu yapı sayesinde işlemler birbirlerini beklemeden kendi işlemini yapar. Kullanıcı bir form üzerinden web isteği başlattığında web isteği cevap verene kadar kullanıcı form üzerinde işlem yapamayacaktır.

Benzer şekilde ağ programlama işlemleri sırasında karşı taraftan bir cevap beklenmeye alındığında program üzerinde işlem yapılmayacaktır. Bu ve bunun gibi durumlarda thread yapısının kullanımı iyi bir kullanıcı deneyimi için faydalı olacaktır. Aşağıdaki örnek thread kullanılmadan yapılmıştır.

```
public class ThreadOrnegi {
public static void main(String[] args) {
        uzunBirIslem();
        System.out.println("Merhaba Thread");
    }
private static void uzunBirIslem() {
        try {
            // Burada uzun bir işlem yapılıyor.
            Thread.sleep(5 * 1000);
            System.out.println("Uzun işlem sonucu");
        } catch (InterruptedException ex) {
            System.err.println(ex);
        }
    }
}
```

Örnekte basit bir Merhaba Thread yazısının yazılması için uzunBirIslem metodunun bitmesi beklenmektedir.

Thread kullanımı

Thread kullanımı için Thread sınıfını kalıtım almak(extends) veya Runnable arayüzünü uygulamak(implements) olmak üzere iki yöntem kullanılır.
Thread sınıfının kullanımı için Thread sınıfı kalıtım alındıktan sonra run metodu ezilir(override) ve gerekli komutlar yazılır.

```
public class ThreadOrnegi extends Thread {
public static void main(String[] args) {
        ThreadOrnegi threadOrnegi = new ThreadOrnegi();
        threadOrnegi.start();
        System.out.println("Merhaba Thread");
    }
@Override
    public void run() {
        try {
            // Burada uzun bir işlem yapılıyor.
            Thread.sleep(5 * 1000);
            System.out.println("Uzun işlem sonucu");
        } catch (InterruptedException ex) {
            System.err.println(ex);
        }
    }
}
```

NOT: main metodu da bir thread oluşturur.
Runnable arayüzünün kullanımı için arayüz metotları sınıf tarafından uygulanır(implements).
Arayüz uygulandıktan sonra Thread sınıfının kurucusuna sınıf parametre olarak geçilerek thread çalıştırılır.

```
public class ThreadOrnegi implements Runnable {
public static void main(String[] args) {
        Thread t1 = new Thread(new ThreadOrnegi());
        t1.start();
        System.out.println("Merhaba Thread");
    }
@Override
```

```
    public void run() {
        try {
            // Burada uzun bir işlem yapılıyor.
            Thread.sleep(5 * 1000);
            System.out.println("Uzun işlem sonucu");
        } catch (InterruptedException ex) {
            System.err.println(ex);
        }
    }
}
```

Her ikisi arasında herhangi bir fark yoktur.
NOT: Java çoklu kalıtımı desteklemediğinden Runnable arayüzünün kullanımı faydalı olacaktır.

Multithreading Nedir?

Java'da çoklu thread, maksimum CPU kullanımıyla aynı anda iki veya daha fazla iş parçacığı yürütme işlemidir.
Çok iş parçacıklı uygulamalar, iki veya daha fazla iş parçacığının aynı anda çalıştığı yerlerdir; Bu nedenle, Java'da eşzamanlılık olarak da bilinir. Bu çoklu görev, birden fazla işlem CPU, bellek vb. ortak kaynakları paylaştığında yapılır.