

Bölüm 13:Java Naming Convention,Paketler,Erişim Belirleyiciler

115:Java Naming Convention(Adlandırma Gelenekleri) Nedir?

- Naming Convention Javada metodlara,sınıflara,değişkenlere ve paketlere isim verirken uyduğumuz bazı geleneklerdir. Bu geleneklere uymak zorunda olmasak da bu geleneklere uymak projemizi daha sonradan inceleyecek geliştiricilere oldukça kolaylık sağlayacaktır.

Değişken İsimleri

- İlk kelime küçük harf ile , ikinci kelime büyük harf ile başlamalı
- Değişken ismi değişkenin ne iş yaptığıni iyi vurgulamalı
- _ çok nadir kullanılmalı
- For ve While döngülerinde değişkenler genelde i,j,k gibi harfler olmalı(Fortrandan kalma bir alışkanlık)
- ForEach Döngülerinde temp gibi değişken isimleri kullanılmalı
- Türkçe karakter kullanılmamalı
- Örnekler : ogrenciNumarası, hesapNo, hayvanısmı



Sabit Değişken İsimleri

- Programlarımızda değerleri değizmeyen değişkenler varsa bu değişkenlerin isimlerinin her harfi büyük harf olmalı
- _ kullanılmalı
- Değerlerinin sonradan değişimmemesi için final anahtar kelimesi ile tanımlanmalı
- Örnekler: PI_SAYISI, AVOGADRO_SAYISI, MAX_SIZE

SQL Id'si gibi değerler.

Method ve Fonksiyon İsimleri

- İlk kelime küçük harf ile , ikinci kelime büyük harf ile başlamalı
- Fiil İfadesi barındırmalı
- İsmi yapacağı iş ile alakalı olmalı
- Türkçe karakter kullanılmamalı
- setter ve getter metodlarının isimleri set ve get kelimeleri barındırmalı
- degerleriYazdir(),
getNumara(),setNumara(),puanHesapla()

Class, Abstract Class ve Interface İsimleri

- İlk kelime ve ikinci Kelime büyük harf ile başlamalı
- Yaptığı işi iyi vurgulamalı
- Fiil değil , İsim olmalı
- Interfacelerin isimleri içerdikleri metodları anlatmalı
- LinkedList,AdayOgrenci, View.OnClickListener

Paket İsimleri

- Küçük harf ile başlamalı
- Benzersiz bir isim verilmeli
- Projenizin websitesi "www.mustafamurat.com/ners" ise paket ismi "com.mustafamurat.ners" olmalı
- "-" yerine "_" kullanılmalı
- Paket ismi Javadaki anahtar kelimeleri içeriyorsa veya değişkenlere verme kurallarına aykırıysa "_" ile düzeltme yapılmalı
- while.xyz.com ---> com.xyz._while

Paket İsimleri

- Küçük harf ile başlamalı
- Benzersiz bir isim verilmeli
- Projenizin websitesi "www.mustafamurat.com/ners" ise paket ismi "com.mustafamurat.ners" olmalı
- "-" yerine "_" kullanılmalı
- Paket ismi Javadaki anahtar kelimeleri içeriyorsa veya değişkenlere verme kurallarına aykırıysa "_" ile düzeltme yapılmalı
- while.xyz.com ---> com.xyz._while

Paket Kullanmanın Önemi

- Dünya üzerinde yaklaşık 10 milyon Java geliştiricisi bulunuyor.
- Bunun için classların ve interfacelerin adlarının çakışmaması imkansız hale geliyor.
- Aynı zamanda büyük projelerde benzer class isimlerinin olması da kaçınılmaz oluyor.

Paket Nedir ?

- Paketler birbiriyle ilişkili classların, interfacelerin birarada bulunduğu bir konteynır görevi görür. Yani biz bir çok paket oluşturarak ilişkili classları bu paketlerde depolayabiliriz.
- Paketler sayesinde, biz bir projenin içinde aynı isimden bir çok classı farklı paketlerin içinde depolayabiliriz.

Paket Kullanmanın Avantajları

- Paketler sayesinde geliştiriciler ilişkili classları ve interfaceleri kolaylıkla bulabilir. Ayrıca paket isimlerinin anlamlı olması da bir paketteki classların ve interfacelerin görevlerini anlatarak geliştiricilerin işlerini kolaylaştırması açısından önemlidir.
- Paketler kullanımı sayesinde class ve interface isimlerinin çakışması en aza indirilmiş olur.
- Paket içindeki classlar başka paketteki classlara bazı sınırlandırmalar haricinden kolaylıkla erişebilir.

- JDK indirmemizin sebebi onun içindeki package, classları kullanmamızdır.
- Libraries->JDK 1.8 içinde jar dosyaları bulunur. Bu jarlarda packageler vardır. Bu packagelarda scanner, arraylist gibi yapıların classları, interface ve abstract classları bulunur.
- import ederek projeye dahil ediyoruz.(import java.util.ArrayList)
- import java.util.* dersek java.util içindeki tüm classları çağırıyoruz.

```
] import org.w3c.dom.Node;  
- import javax.xml.soap.Node;
```

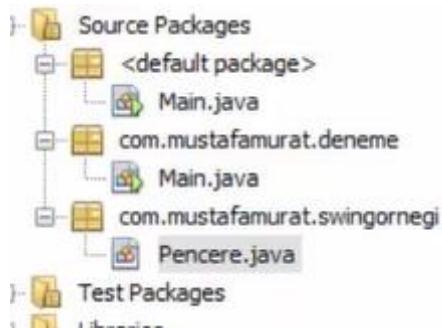
```
reference to Node is ambiguous  
both interface org.w3c.dom.Node in org.w3c.dom and interface javax.xml.soap.Node in javax.xml.soap match  
|  
public class Mai  
| public static  
|-----  
| (Alt-Enter shows hints)  
| Node node1 = null;
```

- Node için iki farklı yerden package import etmişiz. Bu durumda java hangisini kullanacağını anlamayacağı için hata verir.

```
import javax.xml.soap.Node;

public class Main {
    public static void main(String[] args) {
        Node node1 = null;
        org.w3c.dom.Node node2 = null;
```

- Böyle olduğu zaman c++ std::cout gibi olur. O yüzden hata vermez.



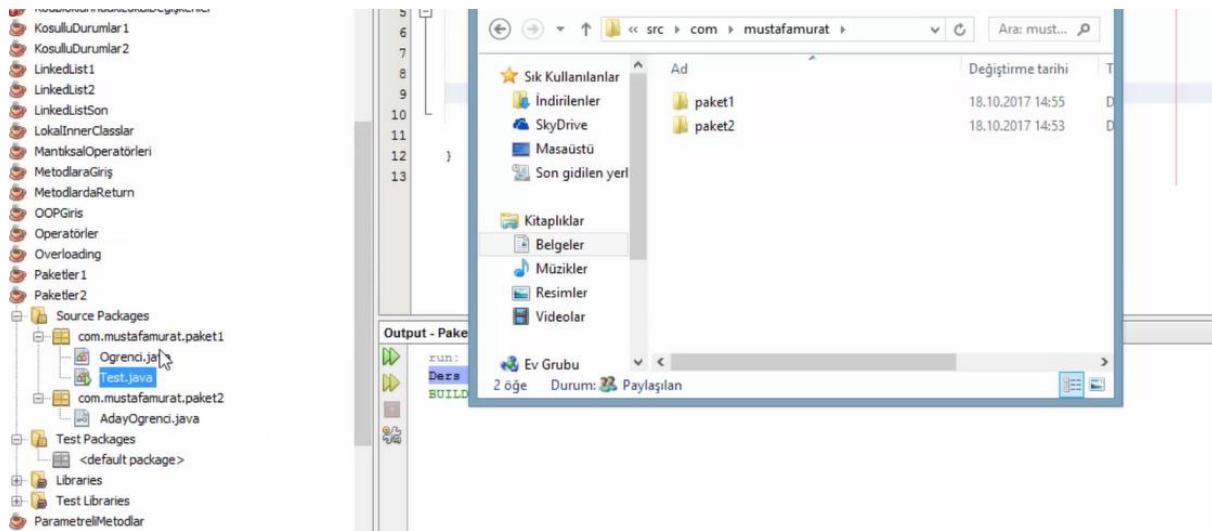
- Bu şekilde paketlerde class tanımlayabilirsin.

The screenshot shows a Java IDE interface with two tabs at the top: "Main.java X" and "Pencere.java X". The "Source" tab is selected. Below the tabs is a toolbar with various icons. The main area displays Java code. The cursor is positioned at the end of the line "JFrame frame = new JFrame();". A tooltip or code completion dropdown is visible, showing the suggestion "JFrame frame = new JFrame();".

```
1
2     package com.mustafamurat.deneme;
3
4     import com.mustafamurat.swingornegi.Pencere;
5     import javax.swing.JFrame;
6
7     public class Main {
8         public static void main(String[] args) {
9
10            Pencere pencere = new Pencere();
11            JFrame frame = new JFrame();
```

- Pencere farklı bir package ve onda Jframe package'si tanımlı. Ancak Main classı farklı bir package'de ve onda tekrardan Jframe packagesini tanımlamamız gereki.

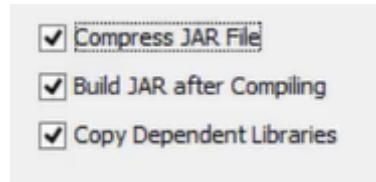
117. Paketler- Part 2 – Paketlerin Bilgisayarda Tutulmaları



- com.harunaydemir.paket dediğimiz zaman com diye klasör oluşturuluyor. Onun içinde harunaydemir diye klasör oluşuyor. Onun içinde de oluşturduğumuz paketler oluyor. Her . ayrı bir klasör açıyor.

118. Paketler- Part 3 – Jar Dosyası Oluşturmak ve Kütüphane Olarak Kullanmak

- Oluşturulan projeye sağtık->Properties->Packaging->



Seçeneklerini seçerek->OK tıklıyoruz. Sonra projeye tekrar sağ tıklayarak->Clean and Build diyerek jar dosyasını oluşturuyoruz.

```
java -jar "C:\Users\user\Documents\NetBeansProjects\Paketler3\dist\Paketler3.jar"
```

Dosya oluşuyor.

"Clean and Build" seçeneği .java uzantılı dosyaları .class uzantılı dosyalara dönüştürür. Ancak aynı zamanda, bu .class uzantılı dosyalara tıklayarak classlarımızı rahatlıkla görebilirsiniz.

- Oluşturulan Jar kullanmak için Proje->Libraries->Sağtık->Add Jar diyerek jar dosyasını kullanabilirsin.(Dışarıdan kütüphane eklemiş oluruz.)

119. Özelliklerin ve Değişkenlerin Kapsamı-Scope

```
public class KapsamSinifi {  
    private int privateDegisken=30;  
  
    public KapsamSinifi() {  
        System.out.println("privateDegisken= "+privateDegisken);  
    }  
    public void onIleCarp(){  
        int privateDegisken=10;  
  
        for(int i=1;i<6;i++){  
            System.out.println(i+"*" +privateDegisken+" = " +(i*privateDegisken));  
        }  
    }  
}
```

- Böyle bir kodda java en local olanı alır. İki tane privateDegisken var ancak metodu çağrıdığımızda method scopeleri içindeki değişkeni alır. Çünkü o method içinde ve methoda göre en local olanı o.

```
for (int i = 1; i < 6; i++) {  
    System.out.println(i + "*" + this.privateDegisken + " = " + (i * this.privateDegisken));  
}
```

- Böyle yaparsak ise this methodu sayesinde privateDegisken=30 değişkenini kullanır. this görevi budur. Eğer privateDegisken=30 silersek this. Anahtar sözüği kullanımı hataverir. Çünkü class'ın öyle bir private değişkeni olmayacak.
- Java'da çoklu kalıtım yoktur.

```
public class KapsamSinifi2 {  
    private int privateDegisken=30;  
  
    public KapsamSinifi2() {  
    }  
  
    public class DahiliSinif{  
        private int privateDegisken=20;  
        public void onileCarp(){  
            for (int i = 1; i < 6; i++) {  
                System.out.println(i+"*" +privateDegisken+" = " +(i*privateDegisken));  
            }  
        }  
    }  
}
```

- Inner class yapısında da aynı mantık geçerlidir. En local olanı kullanır.

```

public class KapsamSinifi2 {
    private int privateDegisken = 30;

    public KapsamSinifi2() {
    }

    public class DahiliSinif {
        private int privateDegisken = 20;

        public void onileCarp() {
            //int privateDegisken = 10;

            for (int i = 1 ; i< 6 ; i++) {

```

- `privateDegisken=10` yorum satırı içinde olmadığından ve programı bu şekilde çalıştırırsak ve `onileCarp` methodu kullanırsak program `privateDegisken=10` u kullanır.

```

public KapsamSinifi2() {
}

public class DahiliSinif {
    private int privateDegisken = 20;

    public void onileCarp() {
        //int privateDegisken = 10;

        for (int i = 1 ; i< 6 ; i++) {

            System.out.println(i + "*" + this.privateDegisken + " = " + (i * this.privateDegisken));
        }
    }
}

```

This anahtar sözcüğü bir üst sınıfın değişkeini kullanır. Inner class larda içinde bulunduğu sınıfın değerini alır. Burada mesela 20 olan değeri alır.

```

public class DahiliSinif {
    private int privateDegisken = 20;

    public void onileCarp() {
        //int privateDegisken = 10;

        for (int i = 1 ; i< 6 ; i++) {

            System.out.println(i + "*" + KapsamSinifi2.this.privateDegisken + " = " + (i * KapsamSinifi2.this.pr

```

- Bu şekilde kullanırsam istediğim değişkeni kullanabilirim. `Kapsamsinifi2`nın değişkenini kullanır.

```

public KapsamSinifi2() {
}
public void dahiliSinifKontrol() {

    DahiliSinif d = new DahiliSinif();

    System.out.println("Kontrol ediliyor... " + d.a);
}

public class DahiliSinif {
    private int privateDegisken = 20;
    private int a = 3;
}

```

- İki scope arasında olan tüm değerlere erişebilirsin. Bir üst sınıf olan kapsamSınıfı2'de alt sınıf olan private a değişkenine erişebildik. Yani inner sınıflarda alt sınıfın değerlerine üst sınıflarda erişebiliriz.

120.Access Modeifierlar(Erişim Belirleyiciler)-public,private,protected,default

```

/*
Access Modifiers- Erişim Belirleyiciler

Erişim Belirleyiciler 2 düzeyde erişimi belirlerler.

Sınıf Düzeyinde Erişim Belirleyiciler

public : Bir class eğer public yazılmışsa bu classa paketin içindeki ve paketin dışındaki
tüm classlar tarafından erişilir.

default: default erişim belirleyici kullanmamak anlamına gelir.
Eğer bir class hiçbir erişim belirleyici kullanmadan yazılmışsa, bu class'a sadece aynı paketin içindeki
classlar erişebilir. Paketin dışındaki classlar bu class'a erişemez.

*** protected, private erişim belirleyiciler classlar için kullanılamazlar.

```

Metod ve Özellik Düzeyinde Erişim Belirleyiciler

I
public: Eğer bir metod veya özellik(class member veya alan) public olarak tanımlanmışsa ,
bu alana paketin içindeki ve dışındaki tüm classlar erişebilir.
Ancak tabii ki bu alanın içinde bulunduğu classın da erişilebilir olması gereklidir.

default: Eğer bir metod veya özellik tanımlanırken hiçbir erişim belirleyici kullanılmamışsa (default),
bu alana sadece kendi paketindeki classlar erişebilir.

protected : Eğer bir metod veya özellik(class member veya alan) public olarak tanımlanmışsa,
bu alana aynı paketin içindeki diğer classlar tarafından erişilebilir. Paketin dışındaki classlar ise
sadece ve sadece bu alanın bulunduğu classın bir alt classıysa erişebilir.

private: Eğer bir alan private olarak tanımlanmışsa bu alana sadece kendi classı erişebilir.

Daha önceden öğrendiğimiz gibi bir metodun içinde tanımlanmış değişkenler lokal değişken olarak tanımlanırlar.Yani,
bu değişkenlere diğer metodlardan ve classlardan erişilemez. Bundan dolayı bu değişkenlere access modifier eklenemez.

- Protected farklı paketten kullanabilmen için extends yapman lazım.

121.Static Anahtar Kelimesi

- Static tanımlanan bir methoda veya değişkene her yerden obje üretilmeden erişilebilir.(Class'ı public ise)
- Statik ifadeler sadece bir yerde çalışır ve sadece bir kere çalışır. İşin en basit hali bu. Static bir değişken herhangi bir class yapısına bağlı değildir. Bellekte tek bir yerde tutulur.
- Static metodların çalışma durumlarıyla aynıdır. Static olan verileri yönetirler. Ve aynı zamanda girdi noktaları olarak kullanılırlar. Fakat onları normal metodların içinde çağrılabılır.
- Static değişkenler bellekte sadece bir kere oluşur.

122.Final anahtar Kelimesi

A screenshot of an IDE showing Java code. The code defines a class named FinalTest with a public final integer variable 'objeSayisi'. A tooltip window is open over the variable declaration, displaying the error message: "variable obje_sayisi not initialized in the default constructor" and "(Alt-Enter shows hints)".

```
public class FinalTest {
    public final int obje_sayisi;
```

Database Şifreleri gibi değerler final yapılabilir.

- Final sınıf değişkenleri: Final olan bir sınıf değişkenine sadece bir kere değer ataması yapılabilir ve bu atama sınıf konstruktöründe gerçekleşebilir.

A screenshot of an IDE showing Java code. It contains a public static final double variable 'PI' assigned the value 3.14159265358979323846.

```
public static final double PI = 3.14159265358979323846;
```

- Public diyerek her yerden erişilebilir yapıyoruz.
Static diyerek obje üretilmesine gerek kalmadan direkt erişebiliyoruz.
Final diyerek başka yerde değiştirilmesini engelliyoruz.

123.Final sınıflar, Final Metodlar ve Final Parametreler

A screenshot of an IDE showing Java code. It defines a class named Database with a public void method 'baglantiKur'. The method takes two final String parameters: 'username' and 'password'. Inside the method, it prints both parameters to the console using System.out.println.

```
public class Database {
    public void baglantiKur(final String username,final String password) {
        System.out.println(username);
        System.out.println(password);
    }
}
```

- Final metot parametreleri: Final olarak tanımlanmış bir metot parametresine sadece bir kere değer atanabilir. Metot parametrelerinin tamamen final olarak tanımlamış olmalarında büyük fayda vardır. Bu şekilde parametrenin metot bünyesinde değişikliğe uğrama tehlikesi ortadan kaldırılmış olur.

```
public class Database {  
    public final void baglantiKur(String username, String password) {  
        System.out.println(username);  
        System.out.println(password);  
    }  
}
```

- Final metodlar: Final olan bir metot ne alt sınıflarca yeniden yüklenebilir (method overloading) ne de saklı (hidden) tutulabilir.

```
public final class Database {  
    public void baglantiKur(String username, String password) {  
        System.out.println(username);  
        System.out.println(password);  
    }  
}
```

- Final sınıflar: Final olan bir sınıf genişletilerek bir alt sınıf oluşturulamaz.(extend edilemezler.)

124. Interfacelere Özellik veya Metod Ekleme Yolları

- Bir sınıf birden fazla interface ile implement (çağırılabilir) edilebilir.
- Interface'e değişken veya metod eklemek istiyorsan tanımlandığı yerde değerini vermelisin ve java bunu direkt static çeviriyor.
- Interface, anlık değişken içermezler. Bu yüzden interface üzerinde tanımlanan değişkenler, interface ile türetilen sınıflar tarafından değiştirilemezler.

```
1  
2  public interface IDeneme {  
3  
4      public int a = 4;  
5  
6  }  
7  
8
```

- Tanımlamalarını gerçekleştirirken sadece public veya default erişim belirleyici kullanabiliriz. Burada a'yı static tanımlamadığımız halde main içinde IDeneme.a diyerek erişebiliyoruz. Çünkü interfacelerden obje oluşturamayız.
- Tanımladığımız Interface'i, birden fazla sınıf çağrıarak kullanabilir.
- Interface üzerinde tanımlanan metodlar gövdesizdir. Bu yapısı ile abstract metodlara benzerlik gösterir.

- Interface sınıfını kullanarak nesne üretemeyiz.
- Interfacelerde static kullanmak saçma çünkü direkt erişebiliyoruz.

Bölüm 14:Java Collection Framework

125.Collection Framework nedir?

Framework Nedir ?

- Framework, yazılım geliştiricilerinin kullandığı önceden hazırlanmış kütüphanelerin bulunduğu ve bunlara yenilerini ekleyebileceğи yapıların adıdır.
- Gelişmiş frameworklerde form kontrolü, veri tabanı bağlantısı, kullanıcı giriş çıkış, mail atma, tema motoru gibi kütüphaneler mevcuttur.
- Ayrıca bu kütüphaneler kendi içinde bir çok Somut Sınıf(Concrete Class), Soyut Sınıf(Abstract Class), Interface ve metod barındırır.

Collection Framework Nedir ?

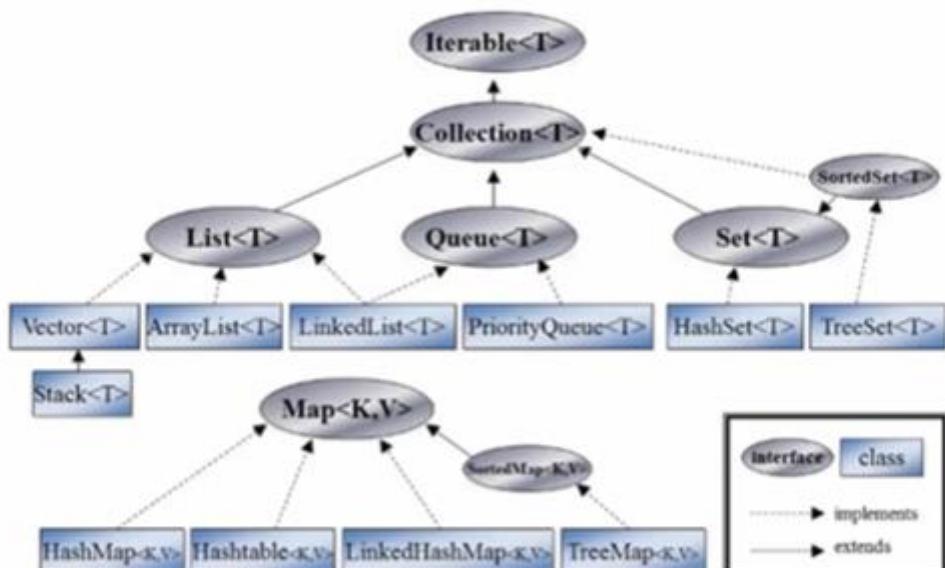
- Benzer verileri grup halinde tutmak ve onlar üzerinde işlemler yapmak yazılım geliştirirken bizim sıkça karşılaştığımız problemlerdir.
- Örnek: Çok sayıda tamsayıyı sıralamak
- Collection Framework bu gibi problemleri ve işlemleri kolayca yapmamızı sağlayan, içinde bir çok somut sınıf, soyut sınıf, interface ve metod bulunduran Java geliştiricileri tarafından tasarlanmış bir frameworktür.

Collection Objesi

- Collection Objesi içinde başka objeler barındıran bir objedir ve Collection Frameworkte bulunan Collection bir interface olup içinde benzer türden objeleri liste halinde, sıralı halde ve benzersiz olacak şekilde barındıracak objelerin temel metodlarını(add, addAll,remove,clear) belirler.

Collection Framework Özellikleri

- Collection Framework içinde bir çok interface ve class bulundurur.
- Verilerin liste halinde tutulması için List interface'i, kuyruk halinde tutulması için Queue interface'i, benzersiz tutulması için Set interface'i bulunur. Bu 3 interface de Collection Interfacededen türerler.
- Ancak Map Interface'i Collection Interface ile bağlantılı olmamasına rağmen Collection Framework'ün içinde bulunan kullanışlı bir interfacedir.
- Bu interface'i uygulayan veya implemente eden somut sınıflar ile bu framework tamamlanır.(Örnek: ArrayList,LinkedList vs.)



126.ArrayList Tekrar ve List Interface

```
    }
    System.out.println("*****");
    for (String s : list) {
        System.out.println(s);
    }
}
```

- List içinde gezinmek için bunu kullanabiliriz.(List<String> list=new ArrayList<String>();)

127. List Interface , ArrayList ve LinkedList Performans Karşılaştırması

128. Set Interface, HashSet , LinkedHashSet ve TreeSet Farkları

Set Interface ----> HashSet, LinkedHashSet, TreeSet

Set Interface ile List Interface'ın farklı
List interface'i implemente eden classlar bir elementten birden fazla depolayabilirken,
Set interface'i implemente eden classlar bir elementten sadece bir tane depolarlar.

- Hepsи Collection sınıfından extends edildiği için ortak metodları fazladır.

HashSet

HashSet Sınıfı

HashSet extends AbstractSet implements Set Interface extends Collection extends Iterable

I

- 1.HashSet elementleri "hashing" yani hash table mekanizmasına uygun bir biçimde depolarlar (Her element belli bir key'e göre depolanır.)
- 2.HashSet bir element'i sadece bir defa depolar.(Set Interface)
3. Elementlerin eklemeye sırasına göre depolamaz.

LinkedHastSet

LinkedHashSet Sınıfı

LinkedHashSet extends HashSet extends AbstractSet implements Set Interface extends Collection extends Iterable

1. LinkedHashSet hem HashTable hem de Set Interface'in LinkedList implementasyonu gibi davranışır.
2. HashSette olduğu gibi bir elementi sadece bir defa depolar.(Set Interface)
3. Elementleri eklemeye sırasına göre depolar.

TreeSet

TreeSet Sınıfı

TreeSet sınıfı NavigableSet interface'ini implemente eder ve AbstractSet sınıfından miras alır.
NavigableSet interface'i de SortedSet interfaceinden miras alır.

1. Elementleri depolamak için Tree yani Ağaç yapısını kullanır.
2. Elementleri alfabetik olarak sıralarlar.

Farkları

HashSet vs LinkedHashSet vs TreeSet

HashSet, Hash Table mekanizmasını uyguladığı için elementler sıralı değildir. Ekleme, Çıkarma ve Arama metodları sabit zamanda(Time Complexity : O(1)) çalışır.

TreeSet, elementleri tree yapısına yani ağaç yapısına uygun depolar. Ekleme, Çıkarma ve Arama metodları O(log(n)) complexity'si ile çalışır.

LinkedHashSet sınıfı hashtable ile linked list yapısını kullanarak elementleri depolar. Bu yüzden, elementler ekleme sırasında göre depolanır. Ekleme, Çıkarma ve Arama metodları sabit zamanda(Time Complexity : O(1)) çalışır.

```
/*
Set<String> set1 = new HashSet<String>();
Set<String> set2 = new LinkedHashSet<String>();
Set<String> set3 = new TreeSet<String>();
```

- Bu şekilde setlerimizi oluşturuyoruz.

```
set1.add("Java");
set1.add("Python");
set1.add("C++");
set1.add("Javascript");
set1.add("Rpp");
```

- .add diyerek elemanları ekliyoruz. Diğerlerinde de aynı şekilde 1 yerine 2 ve 3 yazıyoruz.

```
}
System.out.println("LinkedHashSet*****");
for (String s: set2){
    System.out.println(s);
}
```

- Bu şekilde ekrana bastırıyoruz.(Foreach döngüsü)

```
run:
HashSet*****
Java
C++
Javascript
Php
Python
LinkedHashSet*****
Java
Python
C++
Javascript
Php
TreeSet*****
C++
Java
```

- Çıktığa baktığımızda HashSet karışık sıralanırken LinkedHastSet verdiğimiz sırada çıktı verdi. TreeSet ise alfabetik sıraya göre çıktı verdi.
- Setler aynı elemanı sadece bir kere depolar. Aynı elemanı birden fazla depolamazlar.
- **.contains:** İçine verilen değerin olup olmadığını kontrol eder. Boolean metoddur.

The screenshot shows an IDE interface with a code editor and a terminal window. The code in the editor is:

```
    System.out.println(set1.contains("Go"));
    System.out.println(set1.contains("Java"));

}

}

ut - CollectionFramework_Sets (run) ×
```

The terminal window below shows the output of the run command:

```
run:
false
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

A large play button icon is overlaid on the terminal window.

- **.isEmpty:** Verilen kısmın boş olup olmadığını kontrol eder. Boolean metoddur.

```
System.out.println(set1.isEmpty());
```

- **.remove:** Elamanı kaldırmak için kullanılır.

A screenshot of an IDE showing Java code. The code removes the string "Java" from a set and then prints the remaining elements. The output shows the set contains C++, Javascript, Php, and Python.

```
set1.remove("Java");
for (String s: set1){
    System.out.println(s);
}

}

it - CollectionFramework_Sets (run) X
run:
C++
Javascript
Php
Python
BUILD SUCCESSFUL (total time: 0 seconds)
```

- **.removeAll:** 2 set arasında karşılaştırma yapar. Ortak olanları çıkartıp ortak olan olduğu için true döner.

```
Set<String> set1 = new HashSet<String>();
Set<String> set2 = new HashSet<String>();

set1.add("Java");
set1.add("C++");
set1.add("Python");
set1.add("Javascript");
set1.add("Php");

set2.add("Go");
set2.add("Java"); []
set2.add("CSS");
```

```
Set<String> fark = new HashSet<String>(set2);

System.out.println(fark.removeAll(set1));
System.out.println(fark);

}

it - CollectionFramework_Sets (run) X
```

```
run:
true
[CSS, Go]
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Öncelikle fark setini set2'ye eşitledik. removeAll ile set1 ile ortak değer olan Java'yı kaldırıp ortak değeri fark setinden sildik. Tekrar fark setini yazdırduğumda sadece CSS ve Go çıktı.
- **.retainAll:** Verilen değerlerde kesişim olup olmadığına bakar.

```
Set<String> kesisim = new HashSet<String>(set2);
System.out.println(kesisim.retainAll(set1));

System.out.println(kesisim);
```

- Burada kesişim setine önce set2 değerlerini atadık. Ondan sonra kesişim kümesindeki ortak elemanlar hariç diğerlerini sildik.

```
Output - CollectionFramework_Sets (run) ^ | 
run:
true
[Java]
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Ekrana yazdırduğumda ise retainAll true değeri döndürüp kesişim setide artık tek ortak değer olan Javayı gösterdi.
- Sıralı değerlerde treeset yapısını kullanmak çok daha mantıklıdır. Karışık değerlerde ise hashset kullanmak daha mantıklıdır.

129.HashMap Sınıfı

```
/*
HashMap Sınıfı

1. Değerleri Key(Anahtar) ve Value(Değer) olarak depolar. Her key'e karşılık
gelen bir tane değer bulunur.
2. Bir anahtar sadece bir kez varolabilir. Ancak bir değer birden fazla olabilir.
3. Elementleri tipki HashSet gibi ekleme sırasına göre depolamaz. (HashSet gibi)
```

```
/*
public static void main(String[] args) {

    HashMap<Integer, String> hashMap = new HashMap<Integer, String>();

    hashMap.put(10, "Java");
    hashMap.put(30, "Python");
    hashMap.put(50, "Php");
    hashMap.put(20, "Php");
    hashMap.put(20, "Php");

    System.out.println(hashMap);
}
```



```
Entry Previous Next Select Aa u" * it - CollectionFramework_HashMap (run) X
run:
{50=Php, 20=Php, 10=Java, 30=Python}
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Hashmap bu şekilde tanımlanır. Önce key sonra value verilir. Elemanları karışık çıkartır ve bir key sadece bir kez değer alabilir. (20 key'i 2 tane değer aldı.)
- Eleman eklemek için .put kullanılır.

```
HashMap<Integer, String> hashMap = new HashMap<Integer, String>();

hashMap.put(10, "Java");
hashMap.put(30, "Python");
hashMap.put(50, "Php");
hashMap.put(20, "Php");
hashMap.put(20, "Php");

hashMap.put(50, "Javascript");

System.out.println(hashMap);

```

```
Entry Previous Next Select Aa u" * it - CollectionFramework_HashMap (run) X
run:
{50=Javascript, 20=Php, 10=Java, 30=Python}
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Bir keye sonradan bir değer atarsak önceki değere silip yeni değeri kaydeder.(Php silip yerine JavaScript yazıldı 50. keyde)

```

System.out.println(hashMap);

System.out.println(hashMap.get(50));
System.out.println(hashMap.get(10));
System.out.println(hashMap.get(100));

```

Output - CollectionFramework_HashMap (run) X

```

run:
{50=Javascript, 20=Php, 10=Java, 30=Python}
Javascript
Java
null
BUILD SUCCESSFUL (total time: 0 seconds)

```

- .get diyerek değeri görebiliriz ve değeri olmayan bir keyi yazdırırsak null döndürür.

```

for (Map.Entry<Integer, String> entry : hashMap.entrySet()) {
    System.out.println("Anahtar : " + entry.getKey() + " -----> Değer: " + entry.getValue());
}

```

CollectionFramework_HashMap (run) X

```

1:
Anahat : 50 -----> Değer: Javascript []
Anahat : 20 -----> Değer: Php
Anahat : 10 -----> Değer: Java
Anahat : 30 -----> Değer: Python
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Bütün değerleri ekrana bu şekilde yazdırabiliriz. Entry dönüştürmemiz gereklidir.

130. Map Interface, HashMap , LinkedHashMap ve TreeMap Sınıflarının Farkları

- Setdeğeler ile aynı mantığa sahiptir. HashMap karışık, LinkedHashMap verdiğimiz sırada , TreeMapde keylere göre sıralı şekilde verir.

```

Map<Integer, String> hashmap = new HashMap<Integer, String>();
Map<Integer, String> linkedhashmap = new LinkedHashMap<Integer, String>();
Map<Integer, String> treemap = new TreeMap<Integer, String>();

```

- Mapleri oluşturduk.

```
public static void mapYazdir(Map<Integer, String> map) {  
  
    map.put(10, "C++");  
    map.put(5, "Java");  
    map.put(1, "Python");  
    map.put(2, "Php");  
    map.put(100, "C");  
  
    for (Map.Entry<Integer, String> entry : map.entrySet()) {  
  
        System.out.println("Key : " + entry.getKey() + "Value : " + entry.getValue());  
    }  
}
```



- Değer atama ve sıralama işlemlerini bir metod ile yaptık. Map tipinde değer gönderdik.

```
System.out.println("*****");  
mapYazdir(hashmap);  
System.out.println("*****");  
System.out.println("*****");  
mapYazdir(linkedhashmap);  
System.out.println("*****");  
System.out.println("*****");  
mapYazdir(treemap);  
System.out.println("*****");  
  
*****  
Key : 1Value : Python  
Key : 2Value : Php  
Key : 100Value : C  
Key : 5Value : Java  
Key : 10Value : C++  
*****  
*****  
Key : 10Value : C++          *****  
Key : 5Value : Java          Key : 1Value : Python  
Key : 1Value : Python        Key : 2Value : Php  
Key : 2Value : Php          Key : 5Value : Java  
Key : 100Value : C          Key : 10Value : C++ ]  
*****  
*****
```



- Çıktılarda görüldüğü gibi ilk hashmap, sonra linkedhashmap ve sonradan treemap çıktısı görülmektedir.

```

        for (Integer key : map.keySet()){
            System.out.println("Key : " + key + " Value: " + map.get(key));
        }
    }

```

put - HashMapLinkedHashMapTreeMap (run) ×

```

run:
*****
Key : 1 Value: Python
Key : 2 Value: Php
Key : 100 Value: C
Key : 5 Value: Java
Key : 10 Value: C++
*****
*****
Key : 10 Value: C++
Key : 5 Value: Java
Key : 1 Value: Python
Key : 2 Value: Php
Key : 100 Value: C
*****

```

- Bu şekil yaparakta ekrana bastırabiliriz.

131. Kendi Objelerimizi Maplerde Kullanmak , hashCode() ve equals() metodları

```

class Player {
    private String isim;

    private String id;

    public Player(String isim, String id) {
        this.isim = isim;
        this.id = id;
    }
}

```

- Böyle bir class oluşturduk.(id int olacak yanlış yazdı)

```

public String toString() {
    return "||| ID: " + id + " İsim :" + isim + " |||";
}

```

- Player classı içine Insert Code diyerek ToString metodunu seçtiğimizde (Object classından override edilir.)

```

public static void main(String[] args) {

    Set<Player> hashset = new HashSet<Player>();

    Player player1 = new Player("Mustafa",1);
    Player player2 = new Player("Mehmet",10);
    Player player3 = new Player("Emre",6);
    Player player4 = new Player("Mustafa",1);

    hashset.add(player1);
    hashset.add(player2);
    hashset.add(player3);[      ]
    hashset.add(player4);
}

```

- Main içinde hashset tanımlayıp player değerlerini içine atadık.

```

    hashset.add(player4);

    for (Player p : hashset) {
        System.out.println(p);
    }
}

```

put - HashCodeveEquals (run) X

```

run:
|||| ID: 10 İsim :Mehmet ||
|||| ID: 1 İsim :Mustafa ||
|||| ID: 1 İsim :Mustafa ||
|||| ID: 6 İsim :Emre ||
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Görüldüğü gibi ekrana yazdırduğumızda player 1 ve 4 'ün değerlerinin aynı olmasına rağmen ekrana yazdırdı. Bunun sebebi kendi yaptığımız sınıfın değer döndürünce karşılaşılacak metodların olmaması. String veya int gönderdiğimizde onların içinde default olarak bu metodlar olduğu için onlarda aynı elemanları almıyor.

```
@Override  
public int hashCode() {  
    int hash = 3;  
    hash = 23 * hash + Objects.hashCode(this.isim);  
    hash = 23 * hash + this.id;  
    return hash;  
}
```

hashCode metodu

```
@Override  
public boolean equals(Object obj) {  
    if (this == obj) {  
        return true;  
    }  
    if (obj == null) {  
        return false;  
    }  
    if (getClass() != obj.getClass()) {  
        return false;  
    }  
    final Player other = (Player) obj;  
    if (this.id != other.id)  
        return false;  
    if (getClass() != obj.getClass()) {  
        return false;  
    }  
    final Player other = (Player) obj;  
    if (this.id != other.id) {  
        return false;  
    }  
    if (!Objects.equals(this.isim, other.isim)) {  
        return false;  
    }  
    return true;  
}
```

equals metodu

- Bu 2 metod sayesinde aynı olan değerler karşımıza çıkmaz. Bu metodlara IDE ler kendileri yazabilir. Insert Code diyerek otomatik olarak ToString metodu içinde yazdık.

132. Mini Proje - Map Kullanarak Bir Cümplenin Harf Frekansını Bulma(Tekrar eden harf sayısı)

```
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

public class Main {
    public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        System.out.println("Cümleyi giriniz: ");
        String cumle=scanner.nextLine();

        Map<Character, Integer> frekans=new TreeMap<Character, Integer>();

        for(int i=0; i<cumle.length();i++){
            char c=cumle.charAt(i);
            if(frekans.containsKey(c)){
                frekans.replace(c, frekans.get(c)+1);
            }else{
                frekans.put(c, 1);
            }
        }
        for(Map.Entry<Character, Integer> entry:frekans.entrySet()){
            System.out.println("Karakter : "+ entry.getKey()+"-> "+entry.getValue()+" kere geçiyor...");
        }
    }
}
```

Cümleyi giriniz:
dasdasdasdasdasd
Karakter : a-> 5 kere geçiyor...
Karakter : d-> 6 kere geçiyor...
Karakter : s-> 5 kere geçiyor...

BUILD SUCCESS

- Öncelikle kullanıcından cümlemizi alıyoruz. Sonra map sınıfını oluşturuyoruz. Tekrar eden harfi Character sınıfını sayısını ise integera atacağımız şekilde ayarlıyoruz. Daha sonra for döngüsü oluşturarak cümlenin içinde gezmeye başlıyoruz.
- char c=cumle.charAt(i); : Cümlenin içindeki harfleri tek tek i değişkenine atıyor.
- **containsKey()**: ile HashMap nesnemiz içerisinde anahtar değeri aratarak işlem yapıyoruz. Yani cümle içindeki harf daha önce gelmişse;
- **replace()** metodu iki adet parametre alıyor. Bunlardan birincisi: değişiklik yapılacak HashMap nesnesinin key değeri, diğeri ise ilgili key değerinin karşısındaki value değeri.
- frekans.replace(c,frekans.get(c)+1); : Key değeri c(cümlenin o anki harfi) değiştirmeyerek aynı değeri koyuyoruz. Valuesine(Integer tekrar eden sayı) ise bir arttırıyoruz.
- frekans.put(c, 1); Eğer harf ilk defa geldiyse c olarak o harf atanıyor. Integer valuesı ise 1 olarak atanıyor.

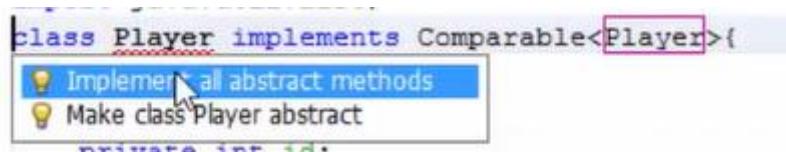
133. Listleri Sıralama ve Comparable Interface'i Kullanma

```
List<String> list_string = new ArrayList<String>();  
  
list_string.add("Java");  
list_string.add("C++");  
list_string.add("Python");  
list_string.add("Php");  
list_string.add("Go");  
  
Collections.sort(list_string);  
  
for (String s : list_string) {  
    System.out.println(s);  
}
```

t - CollectionSortveComparableInterface (run) ×

```
run:  
C++  
Go  
Java  
Php  
Python  
BUILD SUCCESSFUL (total time: 0 seconds)
```

- **Collections.sort(List<T> list):** Listeleri istediğimiz gibi sıralamamız sağlayan metodtur. Kendi yaptığımz sınıfları sıralayamaz.
- Kendi oluşturduğumuzun sınıfı sıralamak istersek eğer o sınıfta comparable interfacenı implements ederek oradaki compareTo metodunu override ederek kendimize göre modifiye etmeliyiz.



- Bu şekilde implements ederek Player sınıfını sıralayacağımızı söylüyoruz.

```
@Override  
public int compareTo(Player player) {  
  
    if (this.id < player.id) {  
  
        return -1;  
  
    }  
    else if (this.id > player.id) {  
        return 1;  
  
    }  
    return 0;  
}
```



- compareTo metodunu override ederek Player tipinde bir player objesini sıralayacağımızı söylüyoruz.-1 veya +1 yazmak zorunda değiliz. Herhangi – bir yada +bir değer yazabiliriz. Bunu yazarak küçükten büyüğe doğru sıralamak istediğimizi söylüyoruz. Eğer gelen id ile o anki id eşit ise return 0 diyerek hiçbirşey yapmıyoruz.

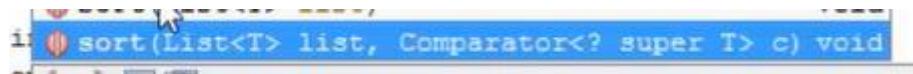
```
Collections.sort(list_player);  
  
for (Player p : list_player) {  
  
    System.out.println(p);  
  
}  
  
}  
  
}

```
CollectionSortveComparableInterface (run) ×
n:
!! ID: 1 İsim :Emre !!!
!! ID: 4 İsim :Yusuf !!!
!! ID: 5 İsim :Murat !!!
!! ID: 10 İsim :Oğuz !!!
ILD SUCCESSFUL (total time: 0 seconds)
```


```

- Bunu yaparak sıralamış oluyoruz. -Büyükür küçütür işaretlerinin yerlerini değiştirirsek büyükten küçüğe doğru sıralar. Treeset kullanıksa compareTo metoduna göre sıralar.

134. Listleri Sıralamak ve Comparator Interface'i Kullanma



- Compare metodunun böyle kullanımı da mevcuttur. Sıralamak için compareTo() metodunu değiştirmek yerine bunu ayrı sınıf ile yapabiliriz. Büyüktен küçüğe , küçükten büyüğe veya kendi istediğimiz sıralamada yapmak istiyorsak Compare implemente eden 3 class oluşturmam gerekiyor.

```
@Override  
public int compareTo(Player player) {  
  
    if (this.id < player.id) {  
  
        return -1;  
  
    }  
    else if (this.id > player.id) {  
        return 1;  
  
    }  
    return 0;  
}
```



- Öncelikle bu compareTo metodu 02<01 ise –01>02 ise artı değer döndürüyor. Yani bu compareTo metodu şuan küçükten büyüğe doğru sıralıyor.

```
class BuyuktenKucugeString implements Comparator<String> {
```

```
@Override  
public int compare(String o1, String o2) {  
  
    I  
  
}
```



- Bu şekilde büyükten küçüğe sıralayan sınıf oluşturuyoruz. Eğer büyükten küçüğe sıralıyacaksa;

Soldaki değer sağdaki değerden büyükse -(o1>o2)

Soldaki değer sağdaki değerden küçükse +(o1<o2)

Eğer eşitlerse sıfır değer döndürmesi gereklidir.(o1=o2)

```
@Override  
public int compare(String o1, String o2) {  
  
    return -o1.compareTo(o2);  
}
```

- Bunu yazarak compareTo metodundan gelen küçükten büyüğe değerini belirten(- ve +) tersine çevirerek büyükten küçüğe doğru sıralıyoruz.(- ile çarparak)

```
Collections.sort(list, new BuyuktenKucugeString());
```

- Bu kullanımda bu şekilde yazarak belirtiyoruz.

Python

Php

Java

Go

C++

- Çıktısı bu şekildedir. Baş harfler belirtildiği gibi olur.

Peki kendi oluşturduğumuz classları bu şekilde sıralamak istersek ne yapacağız?

```
class KucuktenBuyugePlayer implements Comparator<Player> {

    public KucuktenBuyugePlayer() {
    }

    @Override
    public int compare(Player o1, Player o2) {
        if (o1.getId() < o2.getId()) {
            return -1;

        }
        else if (o1.getId() > o2.getId()) {
            return 1;
        }
        return 0;
    }
}
```

Küçükten büyüğe sıralayan class

```
class BuyuktenKucugePlayer implements Comparator<Player> {
```

```

@Override
public int compare(Player o1, Player o2) {
    if (o1.getId() < o2.getId()) {
        return 1;

    }
    else if (o1.getId() > o2.getId()) {
        return -1000;
    }
    return 0;
}

```

Büyükten küçüğe sıralayan class



```

//COLLECTIONS.SORT(list_player,new KucuktensuyugerPlayer());
Collections.sort(list_player,new BuyuktenKucugePlayer());

for (Player p : list_player) {
    System.out.println(p);
}

```

out - CollectionSortveComparatorInterface (run)

```

run:
!!!! ID: 10 İsim :Oğuz !!!
!!!! ID: 6 İsim :Murat !!!
!!!! ID: 4 İsim :Yusuf !!!
!!!! ID: 1 İsim :Emre !!!
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Kullanışı üstteki gibi olan yöntemde çıkışımız Id'lere göre sıralanışı görmektedir. Küçükten büyüğe metodunu kullandığımızda da new kucuktensuyugerPlayer yazmamız yeterli olacaktır.

İsme göre sıralayacaksak ne yapmalıyız?

```

class KucuktensuyugerStringPlayer implements Comparator<Player> {

    @Override
    public int compare(Player o1, Player o2) {

        return o1.getIsim().compareTo(o2.getIsim());
    }
}

```

- compareTo metodunuz yukarıda tanımlıydı. Bu isme göre sıralayacaktır.

```

//Collections.sort(list_player,new KucuktenBuyugePlayer());
//Collections.sort(list_player,new BuyuktenKucugePlayer());
Collections.sort(list_player, new KucuktenBuyugeStringPlayer());

for (Player p : list_player) {

    System.out.println(p);

}

```

compareTo Previous Next Select

it - CollectionSortveComparatorInterface (run) X

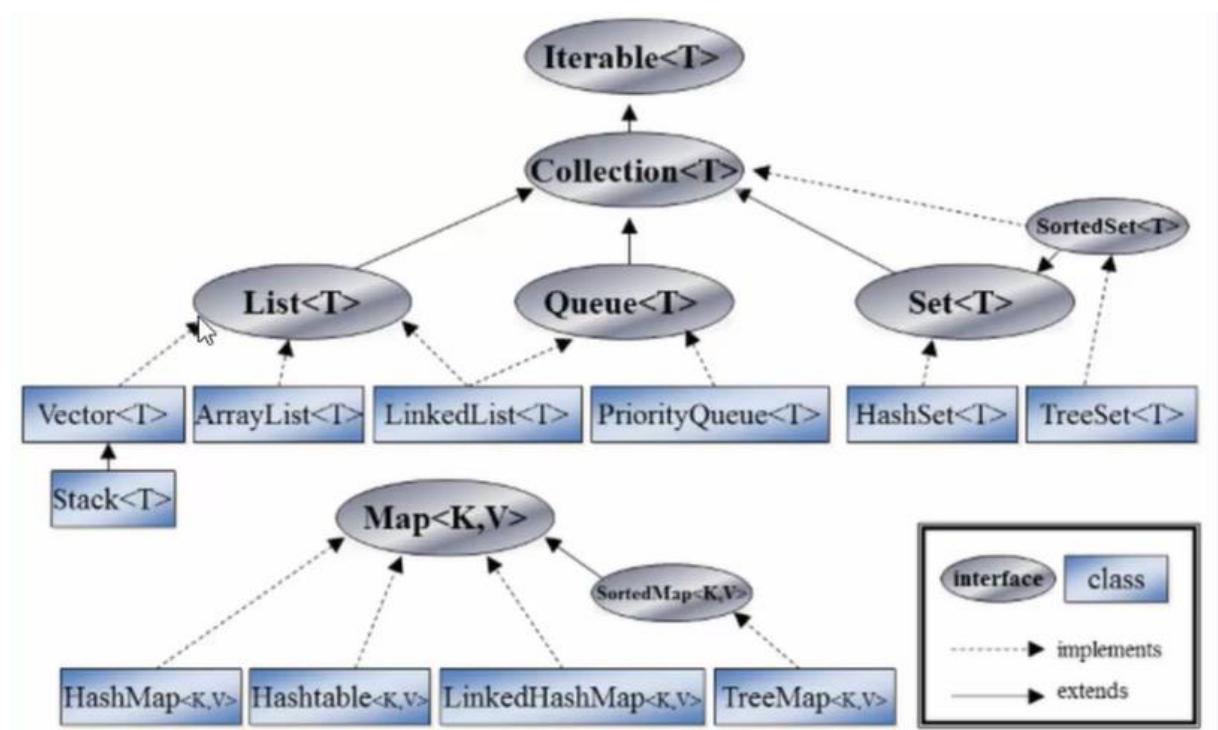
```

run:
!!!! ID: 1 İsim :Emre !!!
!!!! ID: 5 İsim :Murat !!!
!!!! ID: 10 İsim :Oğuz !!!
!!!! ID: 4 İsim :Yusuf !!!
BUILD SUCCESSFUL (total time: 0 seconds)

```

Çıktısı bu şekilde olacaktır.

135. Vectorler ve Stackler + Görsel Anlatım



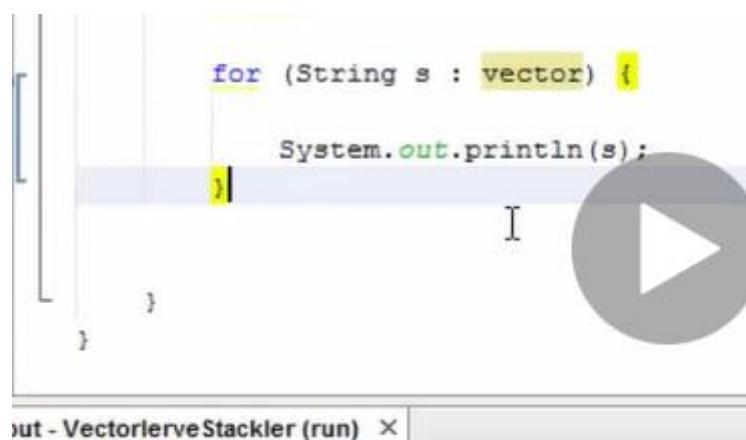
Vectorler threadlerde arraylistlere karşı çok daha güvenli ancak daha yavaş. Performans sıkıntısı varsa arraylist kullanmalıyız.

```
Vector<String> vector = new Vector<String>();  
  
vector.add("Java");  
vector.add("Python");  
vector.add("Python");  
vector.add("Php");
```

Bu şekilde vector oluşturup değerlerini atayabiliriz.

Ekrana bastırmak için birkaç farklı yol yazacağız.

1. For Döngüsü



The screenshot shows an IDE interface with Java code. The code uses a for loop to iterate over the elements of a Vector named 'vector'. Inside the loop, it prints each element using System.out.println(s). The code is as follows:

```
for (String s : vector) {  
    System.out.println(s);  
}
```

Below the code, there is a terminal window titled 'out - VectorlerveStackler (run)' showing the output of the program. The output consists of five lines: 'run:', 'Java', 'Python', 'Python', and 'Php'.

2. Iterator



The screenshot shows an IDE interface with Java code. It uses a ListIterator to iterate over the elements of the vector. The code is as follows:

```
ListIterator<String> iterator = vector.listIterator();  
  
while (iterator.hasNext()) {  
  
    System.out.println(iterator.next());  
}
```

Below the code, there is a terminal window titled 'out - VectorlerveStackler (run)' showing the output of the program. The output consists of five lines: 'run:', 'Java', 'Python', 'Python', and 'Php'.

3.Enumeration

The screenshot shows an IDE interface with two tabs: "out - VectorlerveStackler (run)" and "out - VectorlerveStackler (run) X".

The code in the editor is:

```
Enumeration<String> enumaration = vector.elements();

while (enumaration.hasMoreElements()) {

    System.out.println(enumaration.nextElement());
}

out - VectorlerveStackler (run) X
run:
Java
Python
Python
Php
```

The output in the terminal tab is:

```
İlk Eleman : Java
Son Eleman : Php
```

- `.firstElement()`=Vectorun ilk elemanını bulmayı sağlar.
- `.lastElement()`=Vectorun son elemanını bulmayı sağlar.

Stack

- Thread kısmında kullanılır.
- LIFO: Son giren ilk çıkar.
- Vectorden miras alır.

```
Stack<String> stack = new Stack<String>();
stack.push("Java");
stack.push("Python");
stack.push("Php");
stack.push("Go");
```

- Stack oluşturma ve eleman ekleme

```
for(String s:stack){  
    System.out.println("s");  
}  
Enumeration<String> enumeration=stack.elements();  
while(enumeration.hasMoreElements()){  
    System.out.println(enumeration.nextElement());  
}
```

- 2 farklı stackte gezinme yöntemi

```
System.out.println(stack.peek());
```

- **.peek**: Son elemanını gösterme yöntemi(Silersek bu elemanı siler.)
- Stack'in son elemanını gösterme yöntemi

```
System.out.println("Son eleman çıkarılıyor: "+ stack.pop());
```

- **.pop**: Stackten eleman silme yöntemi(Go'yu siler.)
- Stack'in son elemanını silme yöntemi

```
System.out.println(stack.empty());
```

- **.empty()**: Boolean methoddur. Stack boşsa true dolu ise false döndürür.

136. Mini Proje - Stack Kullanarak Palindrome Kelime Kontrolü Program

```
import java.util.Scanner;
import java.util.Stack;

public class Main {

    public static void main(String[] args) {
        // Bir Cümplenin Palindrome olup olmadığını bulma
        // kasaylabalyasak

        Scanner scanner = new Scanner(System.in);
        System.out.print("Cümleyi Giriniz: ");
        String cumle = scanner.nextLine();

        Stack<Character> stack = new Stack<Character>(); // Sirayla ahrf tutan stack

        for (int i = 0 ; i < cumle.length() / 2 ; i++) { // Cümplenin yarısına kadar harfleri tek tek alan for
            stack.push(cumle.charAt(i)); // Her harf stack atiyoruz.

        }
        if (isPalindrome(cumle, stack)) {
            System.out.println("Bu cümle Palindromdur...");
        } else {
            System.out.println("Bu cümle Palindrom değildir...");
        }

    }

    public static boolean isPalindrome(String cumle, Stack<Character> stack) { // cumleyi ve stackteki harflerin geleceğini söylüyoruz.
        for (int i = (cumle.length() / 2) + 1 ; i < cumle.length() ; i++) { // Cumplenin rotasından itibaren tek tek indis sayısını arttırıyoruz.

            if (cumle.charAt(i) != stack.pop()) { // Stack son giren harf ile cumlenin son harfi eşitse
                return false;
            }
        }
        return true;
    }
}
```

137. Queue Interface ve LinkedList'i Queue Olarak Kullanma

Queue

- FIFO=ilk giren ilk çıkar.

```
/*
 * Queue(Kuyruk) Interface'ini implemente eden LinkedList Classı FIFO(First In, First Out Mantığıyla Çalışır.)
 * add(Eleman) ----> Elemani Kuyruğa Ekler. Ekleymezse Hata Fırlatır.
 * offer(Eleman) ----> Elemani Kuyruğa Ekler. Eklense True Döner, Ekleymezse False Döner.
 * remove() ----> Kuyruğun en başındaki elemani kuyruktan çıkarır. Kuyruk boşsa hata fırlatır.
 * poll() -----> Kuyruğun en başındaki elemani kuyruktan çıkarır. Kuyruk boşsa null referans döner.
 * element() -----> Kuyruğun en başındaki elemani döner. Kuyruk boşsa , hata fırlatır.
 * peek() -----> Kuyruğun en başındaki elemani döner. Kuyruk boşsa , null referans döner.
 */
```

- Ekleme için genelde offer kullanırız.
- Genelde silmek için poll kullanılır.

```
    |
    |    /*
Queue<String> queue = new LinkedList<String>();

queue.offer("Java");
queue.offer("Python");
queue.offer("Php");
queue.offer("C++");
```

- `LinkedList` şeklinde kuyruk şeklinde tanımladık.

```
}

System.out.println("*****");
System.out.println("Eleman Çıkarılıyor : " + queue.poll());

for (String s: queue) {
    System.out.println(s);

}
```

- `Poll` metodu ile Javayı çıkarttık.

```
}

System.out.println(queue.isEmpty());

while (!queue.isEmpty()) {
    System.out.println("Eleman Çıkarılıyor : " + queue.poll());


}

raise
Eleman Çıkarılıyor : Java
Eleman Çıkarılıyor : Python
Eleman Çıkarılıyor : Php
Eleman Çıkarılıyor : C++
RMITD SOURCEKIT. [total time: 0 sec]
```

- Queuedeki tüm elemanları çıkartıp ekrana yazdırın kod biçimidir.

138. Mini Proje - Queue Kullanarak Ramazan Pidesi Kuyruğu Programı

The screenshot shows an IDE interface with two panes. The left pane displays the Java source code for the `Main` class. The right pane shows the output of the `mvn clean package` command.

```
public class Main {  
    public static void main(String[] args) throws InterruptedException {  
        Random random=new Random();  
        Queue<String> pide_kuyrugu= new LinkedList<String>();  
        pide_kuyrugu.offer("Murat");  
        pide_kuyrugu.offer("Hasan");  
        pide_kuyrugu.offer("Okan");  
        pide_kuyrugu.offer("Ayşe");  
        pide_kuyrugu.offer("Merve");  
        pide_kuyrugu.offer("Ezgi");  
        pide_kuyrugu.offer("Gizem");  
        pide_kuyrugu.offer("Nehmet");  
        pide_kuyrugu.offer("Öğuz");  
        pide_kuyrugu.offer("Azer");  
        int pide_sayisi=Math.random.nextInt(10);  
        System.out.println("Pideler çıktı: " );  
        System.out.println("Çıkan pide sayısı: "+pide_sayisi );  
        Thread.sleep(3000);  
        while(pide_sayisi!=0){  
            System.out.println(pide_kuyrugu.poll()+" pideyi aldı");//ilk murat olmak üzere sırayla herkesi çakırıldı  
            //İsmini söyle.  
            pide_sayisi--;  
            Thread.sleep(1000);  
        }  
        System.out.println("Pide kalmadı");  
    }  
}
```

Output of `mvn clean package`:

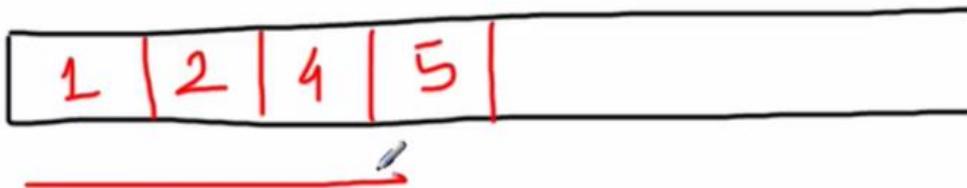
```
[INFO] --- exec-maven-plugin:3.0.0:exec (default-cli) @ PalindromBulma ---  
Pideler çıktı:  
Çıkan pide sayısı: 5  
Murat pideyi aldı  
Hasan pideyi aldı  
Okan pideyi aldı  
Ayşe pideyi aldı  
Merve pideyi aldı  
Pide kalmadı  
BUILD SUCCESS  
Total time: 9.272 s
```

139. Queue Interface ve PriorityQueue

```
public static void main(String[] args) {  
    /*  
     * Queue Interface ve PriorityQueue Sınıfı  
  
     * PriorityQueue normal Queue mantığı gibi davranmaz. Elemanlar öncelik sıralarına göre yüksek öncelik kazanıp  
     * ( )  
     * Integerlarda en yüksek öncelik en küçük sayıda, en düşük öncelik en büyük sayıdadır.  
     * Stringlerde en yüksek öncelik alfabetik olarak sözlükte en önce gelen stringte,  
     * en düşük öncelik alfabetik olarak sözlükte en son gelen stringtedir.  
  
    )  
  
    kuyrukta önlere geçer (Tıpkı Baştan edeki Acil Servisler Gibi).  
  
    add veya offer() metodları -----> Kuyruğa eleman ekler.(Önceden gördüğümüz özellikler taşırlar.)  
    clear() metodu -----> Kuyruğu Temizler.  
    contains(Object o) -----> o objesi kuyruğun içindeyse true, değilse false döner.  
    peek() -----> Kuyruğun bağındaki elemani döner.Eğer kuyruk boşsa null referans döner.  
    poll() -----> Kuyruğun bağındaki elemani çıkartır ve değer olarak döner. Eğer kuyruk boşsa null referans döner.  
    size()-----> Kuyruğun içindeki eleman sayısını döner.
```



2 - 1 - 5 - 4



- PriorityQueue sırayla 2 1 4 5 eklersek içerisinde 1 2 4 5 şekilde sıralanır.

```
    */
    Queue<Integer> queue = new PriorityQueue<Integer>();

    queue.offer(5);
    queue.offer(1);
    queue.offer(2);
    queue.offer(100);

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

it - PriorityQueue (run) X

```
run:
1
5
2
100
```

- Eleman eklediğimizde göründüğü gibi aslında yapı olarak içinde öncelikle olan önde olsa da ekrana direkt yazdırınca o şekilde görünmez.

```
    /*
    Queue<Integer> queue = new PriorityQueue<Integer>();

    queue.offer(5);
    queue.offer(1);
    queue.offer(2);
    queue.offer(100);

    while (!queue.isEmpty()) {
        System.out.println("Eleman Çıkarılıyor : " + queue.poll());
    }
    /*for (Integer i : queue) {
        System.out.println(i);
    }
}*/
```

it - PriorityQueue (run) X

```
run:
Eleman Çıkarılıyor : 1
Eleman Çıkarılıyor : 2
Eleman Çıkarılıyor : 5
Eleman Çıkarılıyor : 100
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Tek tek çıkartıp kullandığımızda asıl sırası görülüyor.

```
System.out.println(queue.peek());
```

- Kuyruğun ilk başındaki elemanı aldığımızda da 1 çıkar.

```
System.out.println(queue.contains(100));
```

- Bu methodla içinde yazılan değerin olup olmadığını görüyoruz.Boolean methoddur. İçinde yoksa false varsa true döner.

-Kendi değerlerimizin sıralamasını nasıl yapacağız?

```
class Player implements Comparable<Player>{  
    private String isim;  
    private int id;  
  
    public Player(String isim, int id) {  
        this.isim = isim;  
        this.id = id;  
    }  
  
    @Override  
    public int compareTo(Player player) {  
  
        if (this.id < player.id) {  
            return -1;  
        }  
        else if (this.id > player.id) {  
            return 1;  
        }  
        return 0;  
    }  
}
```



- Kendi sınıfımızı oluşturduk ve kendi oluşturduğumuz sınıfı sıralamak için Comparable sınıfını implemente ettik. Comparable sınıfının compareTo metodunu override ederek kendimize göre sıralamasını yaptık. id'si küçük olan daha yüksek olacak yani küçükten büyüğe doğru sıralayacak.

```
*/  
Queue<Player> queue = new PriorityQueue<Player>();  
  
queue.offer(new Player("Murat", 5));  
queue.offer(new Player("Mehmet", 1));  
queue.offer(new Player("Oğuz", 100));  
  
while (!queue.isEmpty()) {  
  
    System.out.println("Eleman Çıkarılıyor : " + queue.poll());  
}
```



- Bu şekilde ekrana yazdırılmamız gerekiyor. Ancak poll methodu referans döndürür. O yüzden kendi sınıfımızın içinde bir toString methodu yazmalıyız. Insert code diyerek toString metodumuzu yazıyoruz.

```
@Override  
public String toString() {  
    return "Player{" + "isim=" + isim + ", id=" + id + '}';  
}
```



- Bununla artık çıktımızı alabiliriz.

```
Eleman Çıkarılıyor : Player{isim=Mehmet, id=1}  
Eleman Çıkarılıyor : Player{isim=Murat, id=5}  
Eleman Çıkarılıyor : Player{isim=Oğuz, id=100}
```



- Görüldüğü gibi küçükten büyüğe sıraladı. Eğer büyükten küçüğe sıralamak istersek;

```
        if (this.id < player.id) {  
            return 1;  
  
        }  
        else if (this.id > player.id) {  
            return -1;  
        }  
        return 0;  
  
    }
```

- Böyle yapmamız yeterli olacaktır.

140. Mini Proje - PriorityQueue ve Comparable Interface ile Acil Servis Uygulaması

```
public class Hasta implements Comparable<Hasta> {

    private String isim;
    private String sikayet;
    private int oncelik;// gelen iki değerde string olduğu için oncelik diye bir değer atadık.

    public Hasta(String isim, String sikayet) {
        this.isim = isim;
        this.sikayet = sikayet;

        if(sikayet.equals("Apandisit")){//Burada hastalığın önceliğe göre sıra atıyoruz.
            this.oncelik=3;
        }else if(sikayet.equals("Yanık")){
            this.oncelik=2;
        }else if(sikayet.equals("Baş Ağrısı")){
            this.oncelik=1;
        }
    }

    @Override
    public String toString() {
        String bilgiler=
                    "Hasta Adı : "+ isim+
                    "\nSikayet : "+sikayet+
                    "\nÖncelik : "+oncelik;
        return bilgiler;
    }

    @Override
    public int compareTo(Hasta hasta) {
        if(this.oncelik>hasta.oncelik){
            return -1;
        }else if(this.oncelik<hasta.oncelik){
            return 1;
        }else
            return 0;
    }
}
```

```

import java.util.PriorityQueue;
import java.util.Queue;

public class Main {
    public static void main(String[] args) {
        Queue<Hasta> acilservis= new PriorityQueue<Hasta>();

        acilservis.offer( new Hasta("Murat Bey","Yanık"));
        acilservis.offer( new Hasta("Okan Bey","Baş Ağrısı"));
        acilservis.offer(new Hasta("Elif Hanım","Apandisit"));
        acilservis.offer( new Hasta("Öğuz Bey","Yanık"));
        acilservis.offer(new Hasta("Merve Hanım","Yanık"));
        acilservis.offer( new Hasta("Gizem Hanım","Apandisit"));

        int i=1;
        while(acilservis.isEmpty() !=true) {
            System.out.println("*****");
            System.out.println(i+".sırada");
            System.out.println(acilservis.poll());
            System.out.println("*****");
            i++;
        }
    }
}

```

141.ListIterator ve Iterator Farkları

```

// ListIterator vs Iterator

/*
ListIterator

Sadece List Interface'i implemente eden classlarda kullanılır.
next() ve previous() metodu vardır.
Ekstradan add() metodu bulunur.

Iterator:

Set , Queue ve List Interface'i implement'e eden classlarda kullanılabilir.
previous() metodu yoktur.

```

- ListIterator vektör , stack, linkedlistlerde ve arraylistte kullanılabilir. ListIteratorun kendi ekleme metodu olduğu için listiterator neredeyse oraya ekleme yapabiliyor.previous metodу vardır.
- Iteratorde previous metodу yoktur. Sadece ileri gidebiliyoruz.

```
/*
public static void main(String[] args) {
    Set<String> set = new HashSet<String>();
    List<String> list = new ArrayList<String>();

    set.add("Java");
    set.add("Python");
    set.add("C++");
    set.add("Go");

    list.add("Java");
    list.add("Python");
    list.add("C++");
    list.add("Go");
}

}
```



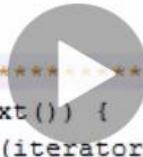
- Şu şekilde bir tane hashset ve arraylist oluşturduk.

```
Iterator<String> iterator1 = set.iterator();
Iterator<String> iterator2 = list.iterator();

while (iterator1.hasNext()) {
    System.out.println(iterator1.next());
}

System.out.println("*****");
while (iterator2.hasNext()) {
    System.out.println(iterator2.next());
}

}
```



- İkisi içinde next diyerek içindekileri görüyoruz. Iterator ikisinde de sonlarına yani Go elemanına gidiyor.

```
while (iterator2.hasNext()) {  
    System.out.println(iterator2.next());  
  
}  
System.out.println("*****");  
while (iterator2.hasNext()) {  
    System.out.println(iterator2.next());  
  
}  
}
```



- iterator ikiye ekrana yazdırduğumızda ise ekranda hiçbir şey çıkmadı. Çünkü iterator önceki yerde setin sonuna gelip orada kaldı. İkinci defa yazdırduğumızda ise hala sonunda olduğu için ekrana hiçbir şey yazdırmadı.

Output - ListIteratorveIterator (run) X

	C++
	Go
	Python

	Java
	Python
	C++
	Go

	BUILD SUCCESSFUL (total time: 0 seconds)

- Görüldüğü gibi sadece 2 kere yazdı.

```
-----+-----+
list.add("C++");
list.add("Go");

ListIterator<String> iterator = list.listIterator();

while (iterator.hasNext()){

    System.out.println(iterator.next());

}

System.out.println("*****");

while (iterator.hasPrevious()) {
    System.out.println(iterator.previous());
}

}

/*set.add("Java");
set.add("Python");
```

put - ListIteratorveIterator (run) X

```
Java
Python
C++
Go
*****
Go
C++
Python
Java
BUILD SUCCESSFUL (total time: 0 seconds)
```

- ListIterator de ise ekrana düz bir şekilde yazdırıldı. Ancak listIterator ondan sonra listenin sonunda olduğu için ve previous metodu olduğu için tersten de yazdırıldı.

```
5     } */
6     while (iterator.hasNext()) {
7         String value = iterator.next();
8         if (value.equals("Go")) {
9             iterator.remove();
10        }
11    }
12    for (String s : list) {
13        System.out.println(s);
14    }
15 }
```

Output - ListIteratorveIterator (run) X

run:
Java
Python
C++
BUILD SUCCESSFUL (total time: 0 seconds)

- Iterator sayesinde bu şekilde çıkarabiliyoruz.

142. Kendi Iterable Sınıflarımızı Oluşturmak ve Iterable Interface

```
import java.util.ArrayList;

public class Radyo {

    private ArrayList<String> kanallar_listesi = new ArrayList<String>();

    public Radyo(String[] kanallar) {
        for (String kanal : kanallar) {
            kanallar_listesi.add(kanal);
        }
    }

}
```

- Kendi classımızı oluşturup(Radyo ya) String dizi gönderiyoruz.

```
public class Main {
    public static void main(String[] args) {
        Radyo radyo = new Radyo();
        for (String s : radyo) {
            System.out.println(s);
        }
    }
}
```

- Eleman gönderdikten sonra ekrana yazdırılmaya çalıştığımızda ise hata alıyoruz. Çünkü bu foreach döngüsü kendi içinde iterator kullanıyor ve bu classı nasıl gezeceğini belirtmedi. Bunu yapmak için Iterable interface'ini implemente etmeliyiz. Bunu kullandığımızda nasıl ataması gerektiğini yazacağımız Iterator metodunu override etmeliyiz.

```
public class Radyo implements Iterable<String> {
    private ArrayList<String> kanallar_listesi = new ArrayList<String>();

    public Radyo(String[] kanallar) {
        for (String kanal : kanallar) {

```

- String bastıracağımız için String yazıyoruz.

```
        @Override
        public Iterator<String> iterator() {
            return kanallar_listesi.iterator();
        }
    }
}
```

- Bu şekilde yaparak çalıştırabiliriz. Kendimizde Iterator tanımlayabiliriz.

```
public class RadyoIterator implements Iterator<String> {
    public RadyoIterator(Radyo radyo) {
        this.radyo = radyo;
    }
    private Radyo radyo;
    private int index = 0;
```

```
    @Override
    public boolean hasNext() {

        if (index < kanallar_listesi.size()) {
            return true;
        }
        else {
            return false;
        }
    }

    @Override
    public String next() {
        String deger = kanallar_listesi.get(index);
        index++;
        return deger;
    }

    @Override
    public Iterator<String> iterator() {
        //return kanallar_listesi.iterator();
        return new RadyoIterator();
    }
}
```

- Bu şekilde ekrana yazdırabiliriz.

143. Mini Proje - Iterable Interface'i ile Kumanda Projesi

```
4  public class Main {
5      public static void main(String[] args) {
6          System.out.println("Kumanda Programına Hoşgeldiniz...");
7          Scanner scanner= new Scanner(System.in);
8
8          String islemler="İşlemler...\n"+
9              "1. Kanalları Göster\n"+
10             "2. Kanal Ekle\n"+
11             "3. Kanal Sil\n"+
12             "4. Kanal Sayısı Öğren\n"+
13             "Çıkış için q'ya basınız";
14
15
16
17         Kumanda kumanda= new Kumanda();
18         while(true){
19             System.out.println(islemler);
20             System.out.println("İşlemi Giriniz :");
21             String islem= scanner.nextLine();
22             if(islem.equals("q")){
23                 System.out.println("Programdan çıkışılıyor...");
24                 break;
25             } else if(islem.equals("1")){
26                 kanallariGoster(kumanda);
27             }
28             else if(islem.equals("2")){
29                 System.out.println("Eklenecek Kanal: ");
30                 String kanal_ismi=scanner.nextLine();
31
32                 kumanda.kanalEkle(kanal_ismi);
33             }
34             else if(islem.equals("3")){
35                 System.out.println("Silinecek Kanal: ");
36                 String kanal_ismi=scanner.nextLine();
37
38                 kumanda.kanalSil(kanal_ismi);
39             }
40             else if(islem.equals("4")){
41                 System.out.println("Kanal Sayısı: "+ kumanda.kanalSayisi());
42             }else{
43                 System.out.println("Geçersiz İşlem");
44             }
45         }
46
47         public static void kanallariGoster(Kumanda kumanda){
48             if(kumanda.kanalSayisi()==0){
49                 System.out.println("Hiçbir kanal bulunamıyor...");
50             }else{
51                 for(String kanal: kumanda){
52                     System.out.println("Kanal"+kanal);
53                 }
54             }
55         }
56     }
57 }
```

```
import java.util.ArrayList;
import java.util.Iterator;

public class Kumanda implements Iterable<String>{
    private ArrayList<String> kanallar= new ArrayList<String>();

    public class KumandaIterator implements Iterator<String>{
        private int index;
        @Override
        public boolean hasNext() {
            if(index>=kanallar.size()){
                return false;
            }else
                return true;
        }

        @Override
        public String next() {
            String kanal=kanallar.get(index);
            index++;
            return kanal;
        }
    }

    public void kanalEkle(String kanal){
        kanallar.add(kanal);
    }

    public void kanalSil(String kanal){
        if(kanallar.contains(kanal)){
            kanallar.remove(kanal);
        }else{
            System.out.println("Böyle bir kanal bulunmuyor");
        }
    }

    public int kanalSayisi(){
        return kanallar.size();
    }

    @Override
    public Iterator<String> iterator() {
        return new KumandaIterator();
    }
}
```

Bölüm 15: Exception Handling (İstisna Yakalama)

145. Exception Handling Nedir ?

Exception ve Exception Handling Nedir ?

Exception(İstisna), programlarımızın çalışma zamanında(runtime) normal akışını bozan olaylardır.

Exception Handling(İstisna Yakalama) ise bu exceptionların yakalanması ve programlarımızın daha güvenli çalışması için geliştirilmiş bir mekanizmadır.

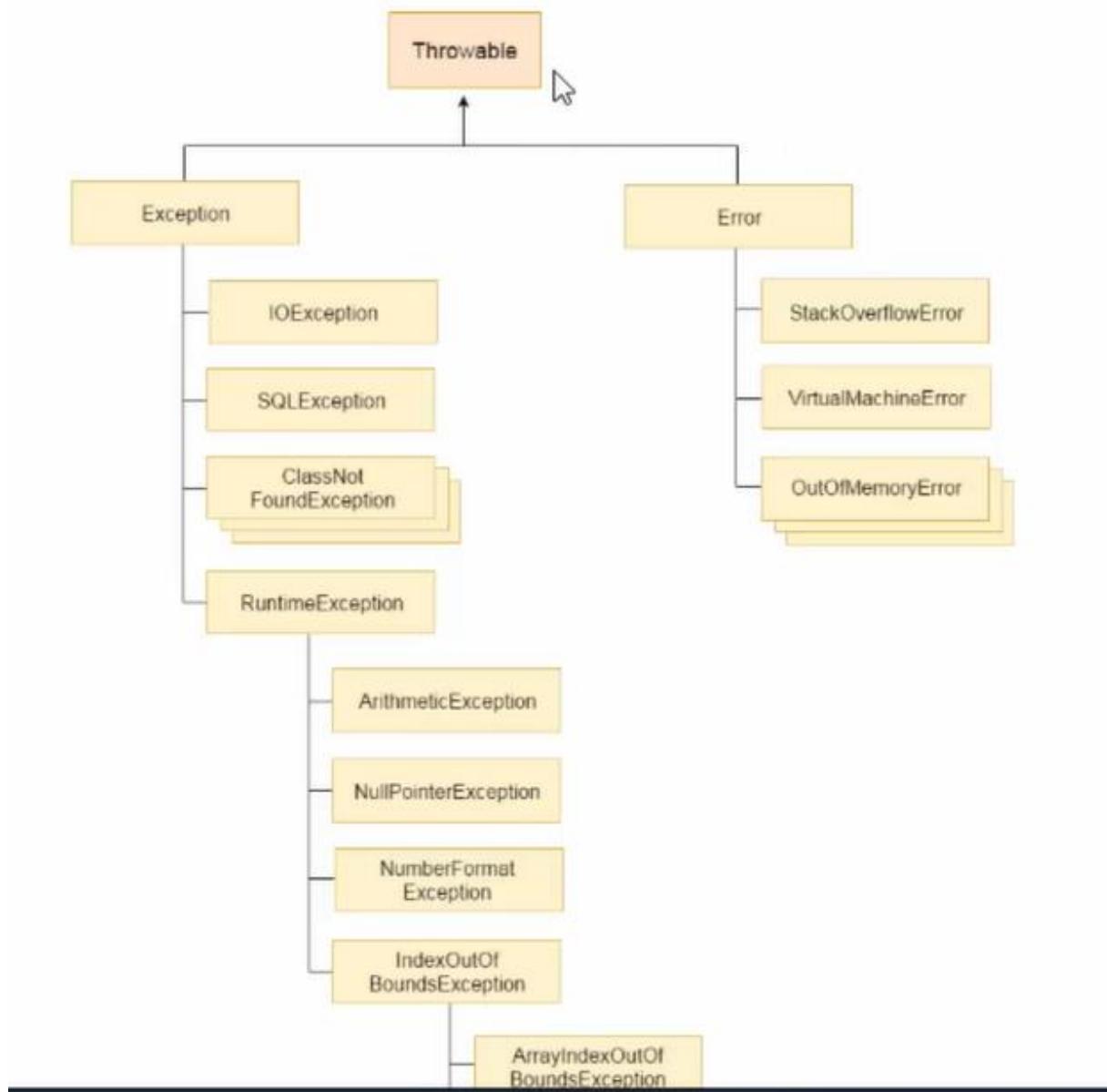
Exceptionlara örnek olarak Input Output hataları , Veritabanı bağlantı hatası ve Aritmetik hatalar verilebilir.

Exception ve Exception Handling Örneği

- Kod Parçasığı 1
- Kod Parçasığı 2
- Kod Parçasığı 3 // Exception Oluştu
- Kod Parçasığı 4
- Kod Parçasığı 5

3. Kod Parçasığında oluşabilecek herhangi bir exception sonucunda java çalışmasını durdurur ve 4. ile 5. kod parçasığını çalıştırılmaz. Ancak Exception Handling mekanizması ile bu exception'ı yakalayıp programımızı daha güvenli yazabiliriz.

Exception Hiyerarşisi



Exception Türleri

Checked Exception: Runtime Exceptionlardan türemeyen her class "Checked Exception" kapsamına girer ve Java kodların exception fırlatabileceğini öngörür. Bu hataları yakalamazsak Java bize bu hataları yakalamamızı söyler. Örnek : IOException,SQLException

UnChecked Exception: Runtime Exceptionlardan türeyen her class "Unchecked" kapsamına girer ve programı çalıştırmadan önce kontrol edilemezler. Yani Java bu exception'ı öngöremez. Hataları yakalama programcının sorumluluğundadır.
Örnek : ArithmeticException, NullPointerException

Error: Geri dönülmez ve tamir edilemeyen hatalardır. Örnek : OutOfMemoryError

Exception Örnekleri

```
String s = null;  
System.out.println(s.length()); //  
NullPointerException
```

```
int b = 12/0;//ArithmeticException
```

```
String s = "a12";  
int i=Integer.parseInt(s);//NumberFormatException
```

146. Try ve Catch Blokları ile İstisna Yakalama

- Genel mantıkları hataları handle exception ile yakalayarak kodun devam etmesini sağlar.

```
/*  
try {  
    // Exception Oluşturabilecek Kodlar  
}  
  
catch(Exception_Türü e) {  
    // Exception Durumunda Yapılacak İşlemeler  
}  
*/
```

- 30/0 bir ArithmeticExceptionidir. Bunu try catch alırsak;

```
try {
    int a = 30 / 0; // ArithmeticException
}

}
catch(ArithmeticException e){
    System.out.println("Bir sayı 0'a bölünemez.");
}

}

Joutput - TryveCatchBlokları (run) ×
run:
Bir sayı 0'a bölünemez.
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Bu şekilde try en uygun catch arar. ArithmeticException en uygun hata olduğu için ekrana bunu yazar.

```
try {
    //int a = 30 / 0; // ArithmeticException
    int[] a = {1,2,3,4,5};

    System.out.println(a[6]);

}
catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Arrayin boyunu aştınız...");
}

}

t - TryveCatchBlokları (run) ×
run:
Arrayin boyunu aştınız...
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Burada da dizi hatasını yakaladık.

```

try {
    //int a = 30 / 0; // AritmeticException
    int[] a = {1,2,3,4,5};

    System.out.println(a[6]);

}

catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Arrayin boyunu aştınız...");

}

System.out.println("Burası Çalışıyor....");
}

```

tput - TryveCatchBlokları (run) X

```

run:
Arrayin boyunu aştınız...
Burası Çalışıyor....
BUILD SUCCESSFUL (total time: 0

```

- Burada görüldüğü gibi hatayı handle exception içine aldığımda kod çalışması devam ediyor.

```

try {
    //int a = 30 / 0; // AritmeticException
    int[] a = {1,2,3,4,5};

    System.out.println(a[6]);

}

catch(Exception e){
    System.out.println("Arrayin boyunu aştınız...");

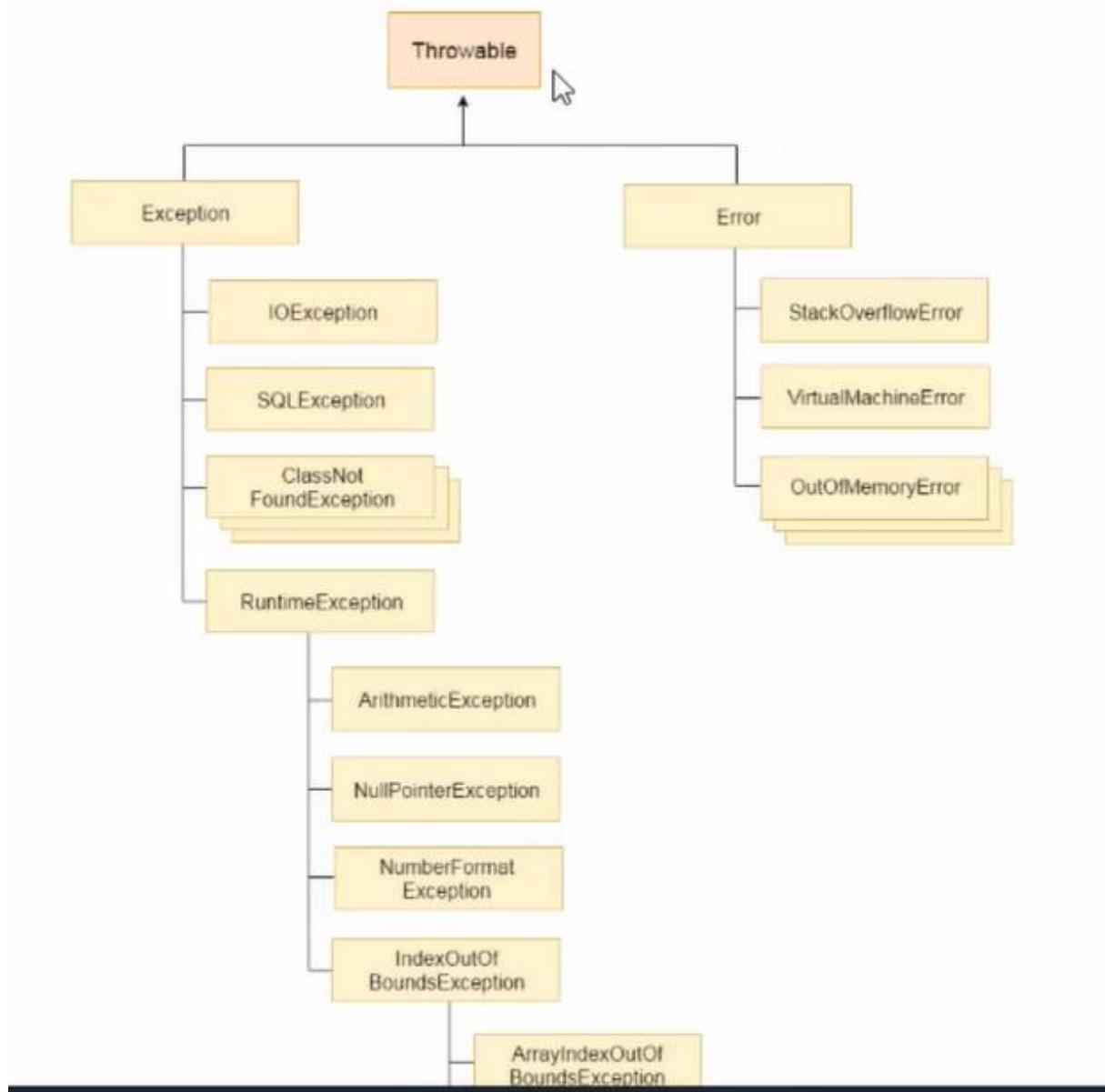
}

```



- Hata yerine Exception yazarsakta hatayı verir. Try catchları arar ve en uygununu seçer. Aşağıda görüldüğü gibi ArrayIndexofBoundsException Exceptiondan türediği için onun yerine Exception da yazabiliriz.

Exception Hiyerarşisi



```
try {  
  
    int[] a = {1,2,3,4,5};  
    System.out.println(a[6]);  
  
    int b = 30 / 0;  
  
}  
catch (Exception e) {  
    System.out.println("Bir hata oluştu...");  
}  
}
```



- Görüldüğü gibi iki farklı hata olduğu halde ikisinide exception diyerek hallettik.

```
try {  
    int b = 30 / 0;  
    int[] a = {1,2,3,4,5};  
    System.out.println(a[6]);  
  
}  
catch (ArithmaticException e) {  
    System.out.println("Bir sayı 0'a bölünemez...");  
}  
catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("Arrayin boyunu aştiniz...");  
}  
catch (Exception e) {  
    System.out.println("Bir hata oluştu...");  
}
```



- Eğer spesifik şekilde belirtmek istersek üst sınıfı alta yazmalıyız. Bu sayede try en son hiçbirşey bulamazsa ona gelecektir.Yani detaylı şekilde tanımlamak istersek hataları üst sınıfı en son yazmalıyız ki hiçbirini bulamazsa onu yazsın. Eğer ilk exceptioni görürse onda yazanı görüp bloktan çıkar ki java zaten böyle bir durumda hata verir.

147. Finally Blokları

- Exception olussa da olusmasa da bu blok mutlaka calisir. Yani eger biz kodlarimizda mutlaka yapilmasi istedigimiz bir islem varsa bunu tanimlamalıyiz. Buna örnek olarak dosya kapama veya sql baglantisi kesme gibi durumları örnek verebiliriz.

The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains the following Java code:

```
try {  
    String s = null;  
    System.out.println(s.hashCode());  
}  
catch (NullPointerException e) {  
    System.out.println("Null Referans Hatası....");  
}  
finally {  
    System.out.println("Finally bloğu çalışıyor....");  
}
```

The terminal window titled "out - FinallyBloğu (run)" displays the following output:

```
run:  
Null Referans Hatası....  
Finally bloğu çalışıyor....  
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Hata olduğu halde bloktan çıkmayarak finally metodunda olanıda çalıştırıldı.

```

        String s = "Mustafa";

        System.out.println(s.hashCode());

    }
    catch (NullPointerException e) {
        System.out.println("Null Referans Hatası.... ");
    }
    finally {
        System.out.println("Finally bloğu çalışıyor....");
    }
}

```



ut - FinallyBloğu (run) X

```

run:
-1217292781
Finally bloğu çalışıyor....

```

- Hata oluşmadığı halde finally metodunda yazan yine çalıştı.

```

try {
    int a = 30 / 0;

}

catch (NullPointerException e) {
    System.out.println("Null Referans Hatası.... ");
}
finally {

    System.out.println("Finally bloğu çalışıyor....");
}

```

ut - FinallyBloğu (run) X

```

run:
Finally bloğu çalışıyor....
Exception in thread "main" java.lang.ArithmetricException: / by zero
at Main.main(Main.java:8)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java return
BUILD FAILED (total time: 0 seconds)

```

- Exceptionu yakalamadığımız halde finally yine de çalıştı. Ancak ondan sonra program kapandı.

148. Throw Anahtar Kelimesi ile İstisna Fırlatma

- Kendimiz exception oluşturmak için kullanırız.

```
7     public static void mekan_kontrol(int yas) {  
8  
9         if (yas < 18) {  
10             throw new ArithmeticException();  
11         }  
12         else {  
13             System.out.println("Mekana hoşgeldiniz...");  
14         }  
15     }  
16 }
```

- Burada mekana 18 yaşından altı birisi girince hata fırlatan bir class yazdık.

```
| public static void main(String[] args) {  
|  
|     Scanner scanner = new Scanner(System.in);  
|     System.out.println("Lütfen yaşıınızı giriniz: ");  
|     int yas = scanner.nextInt();  
  
|     mekan_kontrol(yas);  
| }  
|  
| } // class Main
```

- Burada yaşıımızı alıyoruz 18den küçükse ;

```
16
Exception in thread "main" java.lang.ArithmaticException
    at Main.mekan_kontrol(Main.java:10)
    at Main.main(Main.java:29)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 19 seconds)
```

- Bu hatayı yakalamadığımız için(try catch içine almadık) program sonlanacak.

```

try {
    mekan_kontrol(yas);

}
catch (ArithmetricException e) {
    System.out.println("18 yaşından küçükler mekana giremez..."); 
}

```

- Bu şekilde de hatayı yakalamalıyız.

149. Throws Anahtar Kelimesi , Checked Exception ve Unchecked Exception Farkı

Unchecked Exception

- İsminden de anlaşılacağı gibi derleyici tarafından kontrol edilemeyen exceptionlardır. Derleyici tarafından kontrol edilemedikleri için kodumuz başarıyla derlenmiş olsa bile çalışma zamanında bu hatalarla karşılaşabiliriz.
- Unchecked Exceptionlar RuntimeException'ın alt sınıflarıdır. Bunlardan en yaygınları ArithmetricException , NullPointerException , ClassCastException... gibi exceptionlardır.

Checked Exception

- İsminden de anlaşılacağı üzere derleyici tarafından kontrol edilen exceptionlardır. Derleyici tarafından kontrol edildikleri için biz bu exceptionları kodumuzda belirtmek zorundayız. Aksi halde derleme işlemini başarıyla tamamlamak mümkün değildir.
- Checked Exceptionlar Exception sınıfının alt sınıflarıdır (RuntimeException hariç). En yaygınları ClassNotFoundException, IOException, SQLException... gibi exceptionlardır.

```

public static void mekan_kontrol(int yas) throws IOException {
    if (yas < 18) {
        throw new IOException();
    }
    else {
        System.out.println("Mekana hoş geldin..."); 
    }
}

```



- Burada throw kısmına IOException yazdık. Bu bir checked exception yani derleyici tarafından kontrol edilebildiği için bunu yazan kısma burasının bir IOException fırlattığını belirtmeliyiz. Bunu da throws IOException diyecek yapıyoruz.

```
    mekan_kontrol(yas);  
    }  
    A Add throws clause for java.io.IOException  
    Surround Statement with try-catch  
    Surround Block with try-catch
```

Üstteki methodda bunu belirttiğimizde ve methodu kullanmaya çalıştığımızda java hata veriyor. Görüldüğü gibi bunu try-catch içine almamızı söylüyor. Derleyici bu hatayı anladığı için ve try catch içine almamız gerektiğini bildiği için hatayı veriyor. Mutlaka yakalamamız gerektiğini söylüyor.

```
    try {  
        mekan_kontrol(yas);  
    } catch (IOException ex) {  
        System.out.println("IOException oluştu...");  
    }  
}  
  
ut - Throwsanahtarkelimesi (run) ×  
run:  
Lütfen yaşıınızı giriniz:  
17  
IOException oluştu...  
BUILD SUCCESSFUL (total time: 9 seconds)
```

Aldığımızda ise böyle oluşuyor.

```
}  
public static void main(String[] args) throws IOException {  
  
    Scanner scanner = new Scanner(System.in);  
    System.out.println("Lütfen yaşıınızı giriniz: ");  
    int yas = scanner.nextInt();  
  
    mekan_kontrol(yas);
```



Add throws clause for java.io.IOException seçenek methodun kullanıldığı kısma throws IOException ekliyor. Eğer programı bir başkası kullanacaksa (jar veya api şeklinde) burada throws kendisinin yazması gereklidir.

150. Stacklerle Beraber Exceptionların İç içe metodlarda yakalanma mantığı

151. Kendi Exception Sınıflarımızı Oluşturmak

Kendi exception sınıfımızı yazmak için bir üst sınıfından üretmeliyiz. Bu üst sınıf IOException, rumtineexception olabilir.

```
public class InvalidAgeException extends ArithmeticException {  
  
    public InvalidAgeException(String message) {  
  
        super(message);  
  
    }  
    @Override  
    public void printStackTrace() {  
        System.out.println("Bu bir invalid age hatasıdır...");  
    }  
}
```

ArithmaticException bir uncespted exception olduğu için bizim tanımladığımız InvalidAgeException da bir unexpted exceptiondır.

```
public class Main {  
    public static void mekan_kontrol(int yas) {  
  
        if (yas < 18 ) {  
  
            throw new InvalidAgeException("Invalid Age");  
        }  
        else {  
            System.out.println("Mekana Hoşgeldiniz...");  
        }  
    }  
}
```

Mekan kontrol kısmını oluşturduk.

```
public static void main(String[] args)
    Scanner scanner = new Scanner(Sy

    System.out.println("Lütfen yaşın
    int yas = scanner.nextInt();
    try
    {
        mekan_kontrol(yas);
    }
    catch(InvalidAgeException e) {

        e.printStackTrace();
    }
}
```

ut - KendiExceptionClassimiz (run) X

```
run:  
Lütfen yaşıınızı girin :  
16  
Bu bir invalid age hatasıdır...  
BUILD SUCCESSFUL (total time: 8 seconds)
```

e.printStackTrace çağrıarak kendi yazdığımız bu bir invalid age hatasıdır ekrana geliyor.

```
source History public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.println("Lütfen yaşıınızı girin : ");
    int yas = scanner.nextInt();
    try
    {
        mekan_kontrol(yas);
    }
    catch(InvalidAgeException e) {
        System.out.println(e);
    }
}
```

about_KondiExceptionClassnames(true);

```
run:
Lütfen yaşıınızı girin :
16
InvalidAgeException: Invalid Age
BUILD SUCCESSFUL (total time: 2 seconds)
```

Sadece e bastırdığımızda mekan kontrol içinde yazdığımız invalid age geldi.

```
17  public static void main(String[] args) {
18      Scanner scanner = new Scanner(System.in);
19
20      System.out.println("Lütfen yaşıınızı girin : ");
21      int yas = scanner.nextInt();
22
23      mekan_kontrol(yas);
24
25
26
27
28
29
30  }
31
32 }
33
```

Output - KendiExceptionClassımız (run) ×

```
run:
Lütfen yaşıınızı girin :
16
Exception in thread "main" InvalidAgeException: Invalid Age
    at Main.mekan_kontrol(Main.java:10)
    at Main.main(Main.java:24)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java
BUILD FAILED (total time: 2 seconds)
```

Hatayı yakalamadığımızda ise ekrana gelen hata bu şeylededir.

Bölüm 16:Java Input Output

154. Java I / O Nedir ?

Java I / O

- Java I / O , dosyalardan ve değişik kaynaklardan input almak ve dosyalara ve değişik kaynaklara output yazmak için bir API olarak bilinir.
- Input ve Output işlemlerini hızlandırmak için Java **streamleri** kullanır. Java I/O Apisinin içinde input, output işlemleri için onlarca sınıf bulunur.

Stream Nedir ?

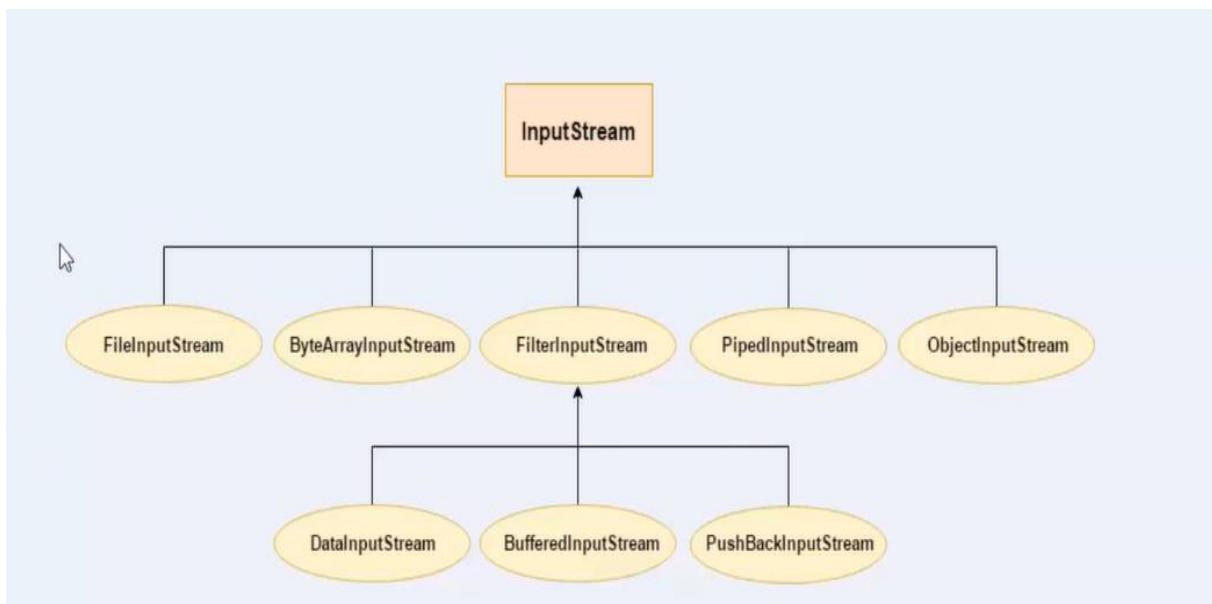
- Stream verilerin input ve output yoluyla Java projelerine aktığı bir veri dizisidir ve Javada streamler verilerin byte(1 ve 0) halinde aktığı yapılardır.
- Javadaki standard streamlerden bir tanesini aslında daha önce kullandık. Javada hazır kullanmamız için oluşturulmuş 3 tane hazır stream bulunmaktadır.
- **System.out** : Standart Output Stream
- **System.err** : Standart Error Stream
- **System.in** : Standart Input Stream

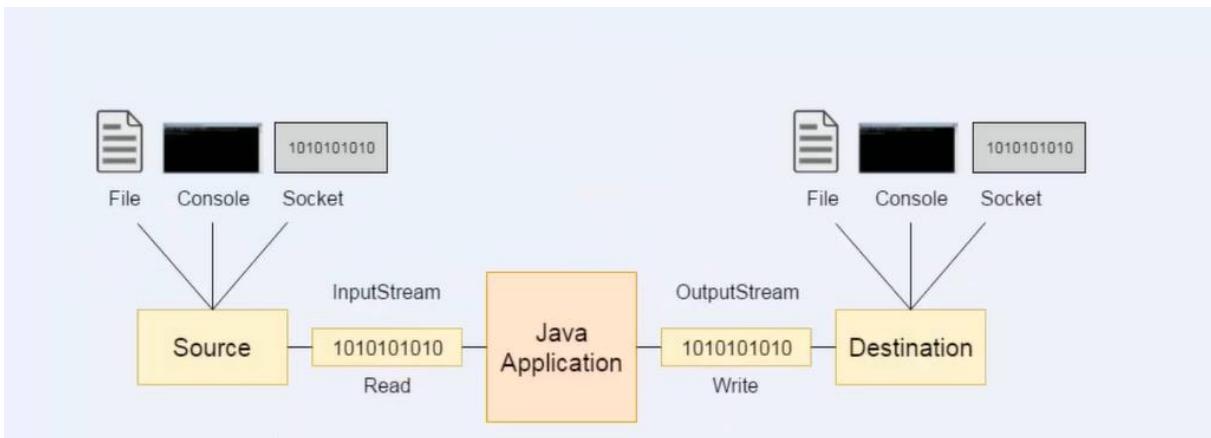
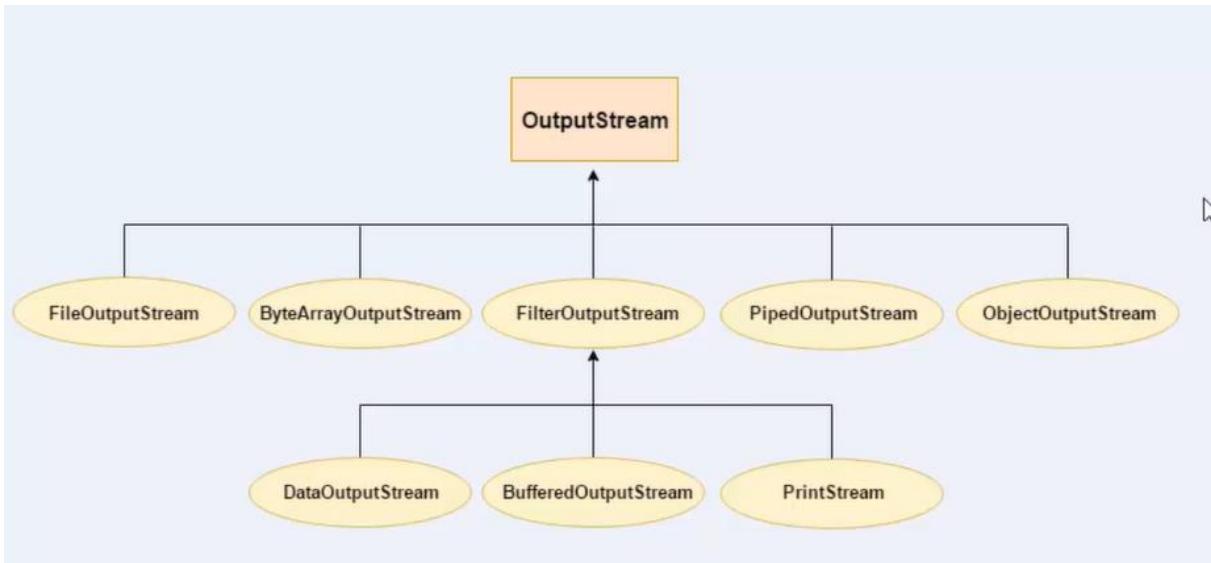
OutputStream

- Java uygulamaları bir dosyaya, bir sockete ve bağlanmış bir aygıta veriyi Stream halinde yazmak (byte halinde 1 ve 0'lar ile) için OutputStream abstract classından (soyut sınıf) türeyen bir alt sınıfı kullanır.
- **write()** metodu : Belli bir byte dizisini veya tek bir byte'ı hedefe yazar.
- **flush()** metodu : Herhangi bir bufferlanmış veri varsa bu metod sayesinde hemen hedefe yazılmasını söylemiş oluruz.
- **close()** metodu : OutputStream'i kapatır.

InputStream

- Java uygulamaları bir dosyadan, bir socketten ve bağlanmış bir aygıtın veriyi Stream halinde okumak (byte halinde 1 ve 0'lar ile) için InputStream abstract classından (soyut sınıf) türeyen bir alt sınıfı kullanır.
- **read()** metodu : Kaynaktan bir sonraki byte'i okur. Okuyacak herhangi bir byte yoksa -1 değeri döner.
- **available()** metodu : Okunabilecek byte sayısını döner.
- **close()** metodu : InputStream'i kapatır.





155. FileOutputStream ile Dosyalara Veri Yazmak

```
FileOutputStream fos = null;
//File file = new File("dosya.txt");
try {
    fos = new FileOutputStream("dosya.txt");
} catch (FileNotFoundException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}

}
```

- **FileOutputStream**, bir dosyaya (file) ya da dosya belirleyiciye (file descriptor) yazan bir çıkışakımıdır. Bir dosyanın elde olması ya da yaratılması, alttaki platforma bağlıdır. Bazı platformlar, biranda dosyayı yalnızca yazmak üzere açar, bazıları hem okumaya hem yazmaya izin verir. Açık olan bir dosya tekrar açılmak istenirse kurucu başarısız olur. FileOutputStream, resim v.b. gibi raw byte yazar. Karakter yazmak için FileWriter sınıfı kullanılmalıdır.
- Aslında .txt uzantılı dosyalarda filewriter ve filereader kullanmak daha mantıklıdır.
- Bu şekilde dosya.txt oluşturuyoruz. Ancak bu dosyayı kapatmalıyız. Bunu da finally kısmında yapıyoruz.

```
fos = new FileOutputStream("dosya.txt");
} catch (FileNotFoundException ex) {
    System.out.println("File Not found exception oluştu....");
}
finally{

    try {
        fos.close();
    } catch (IOException ex) {
        System.out.println("Dosya kapatılırken bir hata oluştu...");
    }
}
```

- Bu şekilde de dosyayı kapatıyoruz.

```
//File file = new File("dosya.txt");
try {
    fos = new FileOutputStream("dosya.txt");

    fos.write(65);  f

} catch (FileNotFoundException ex) {
    System.out.println("File Not found exception oluştu....");
} catch (IOException ex) {
    System.out.println("Dosyaya yazılırken bir hata oluştu....");
}
FileOutputStream (run) X
```

- **.write:** Dosyaya veri yazmak için kullanıyoruz. Ancak bu bir IOException ahtası ürettiğinden dolayı onu da yakalayarak Dosyaya yazılırken hata oluşturma mesajını yazıyoruz. 65 sayısı A harfine denk gelmektedir. Dosya.txt'de A harfi yazılır bu sayede.

```
FileOutputStream fos = null;  
//File file = new File("dosya.txt");  
try {  
    fos = new FileOutputStream("dosya.txt");  
  
    [fos.write(65);  
    fos.write(74);  
  
} catch (FileNotFoundException ex) {  
    System.out.println("File Not found exception oluştu....");
```



- İlk null olarak tanımlayıp sonradan tanımlamamızın nedeni try catch'in içinde tanımlamazsak final içinde kapatamayız.
- Burada 74 harfi J denk gelmektedir. Bunu dosya.txt yazıyoruz ve dosya .txt de artık AJ yazıyor. Ancak eğer fos.write(74) siler ve tekrar çalıştırıldığımızda sadece A yazar. Dosyanın içeriğinin silinmemesi için;

```
//File file = new File("dosya.txt");  
try {  
    [fos = new FileOutputStream("dosya.txt",true);  
  
    fos.write(65);
```



- Dosyanın oluşturduğumuz yerin yanına true eklemeliyiz. Bu dosyanın içeriği varsa sona ekle yoksa dosyayı oluştur demektir.

1	AJJJJ
---	-------

- Fos.write(74) birkaç kere çalıştığımızda gördüğümüz gibi A silinmedi. Yanına eklendi.

```
// FILE NAME = new File("dosya.txt");
try {
    fos = new FileOutputStream("dosya.txt",true);

    byte[] array = {101,75,66,68};
    fos.write(array);
}
```



- Bu şekilde byte array oluşturarakta kullanabiliriz.



- Bu şekilde yazabiliyoruz.

```
//byte[] array = {101,75,66,68};
String s = "Mustafa Murat";

byte[] s_array = s.getBytes();

fos.write(s_array);
```

- Eğer dosyaya bir şey yazdırırmak istiyorsak bunu byte değerine dönüştürmek zorundayız. Bu şekilde de dosya.txt yazdırabiliriz.

156. FileInputStream ile Dosyalardan Veri Almak

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

        FileInputStream fis = null;

        try {
            fis = new FileInputStream("dosya.txt");
        } catch (FileNotFoundException ex) {
            System.out.println("File bulunamadi....");
        }
        finally {
            try {
                fis.close();
            } catch (IOException ex) {
                System.out.println("Dosya kapatılırken bir hata oluştu...");
            }
        }
    }
}
```

- Bu şekilde dosyamızı oluşturduk. Dosyanın içinde java programlama dili yazıyor.

```

try {
    fis = new FileInputStream("dosya2.txt");

} catch (FileNotFoundException ex) {
    System.out.println("File bulunamadi....");
}
finally {
    try {
        if (fis != null ) {
            fis.close();
        }
    }
}

} catch (IOException ex) {
    System.out.println("Dosya kapatılırken bir hata oluştu...");
```



- Eğer dosyamız bir sebepten açılmazsa hata almamak için(burada dosya2.txt yaptık hata almak için)

Bu şekilde if bloğuyla kontrol etmeliyiz.

```

try {
    fis = new FileInputStream("dosya.txt");

    System.out.println("Okunan Karakter : " + (char)(fis.read()));

} catch (FileNotFoundException ex) {
    System.out.println("File bulunamadi....");
} catch (IOException ex) {
    System.out.println("Dosya okunurken hata oluştu.");
}
finally {
```

Fis.read ile byte şekilde dosya okuduk. Bu yüzden bunu char şeklinde dönüştürüyoruz. Ancak char karakter tip olduğu için sadece 1 harf okur. Ekran çıktısı sadece dosya.txt de yazan ilk harf olur(j).

```

15
16         System.out.println("Birinci Karakter : " + (char)(fis.read()));
17         System.out.println("İkinci Karakter : " + (char)(fis.read()));

18
19
20     } catch (FileNotFoundException ex) {
21         System.out.println("File bulunamadi....");
22     } catch (IOException ex) {
23         System.out.println("Dosya okunurken hata oluştu...");
24     }
25     finally {
26         try {
27             if (fis != null ){
28                 fis.close();
```

Output - FileInputStream (run) X

```

run:
Birinci Karakter : J
İkinci Karakter : a
```

- Tekrar yaptığımızda 2. Karekter okunur(a).

```
3   try {
4       fis = new FileInputStream("dosya.txt");
5
6       fis.skip(5);
7
8       System.out.println("Okunan Karakter : " + (char)fis.read());
9      /*System.out.println("Birinci Karakter : " + (char)(fis.read()));
10     System.out.println("İkinci Karakter : " + (char)(fis.read()));
11     System.out.println("Üçüncü Karakter : " + (char)(fis.read()));*/
12
13
14
15 } catch (FileNotFoundException ex) {
16     System.out.println("File bulunamadi....");
17 } catch (IOException ex) {
18     System.out.println("Dosya okunurken hata oluştu " + ex);
19 }
```

Output - FileInputStream (run) X

```
> run:
> Okunan Karakter : P []
BUILD SUCCESSFUL (total time: 0 seconds)
```

- `.skip`: Belirli bir kısmı atlayarak ondan sonraki kısmı gösterir. Java programlama dilide 5. Karekterden sonrası yani 6. Karekter p'yi gösterdi. `fis.read()` eğer döndürecek değer bulamazsa -1 döndürür.

```
13   try {
14       fis = new FileInputStream("dosya.txt");
15
16
17       int deger;
18
19       String s = "";
20
21       while ((deger = fis.read()) != -1) {
22
23           s += (char) deger;
24
25       }
26
27
28       System.out.println("Dosya İçeriği : " + s);
29
30       //fis.skip(5);
31 }
```

Output - FileInputStream (run) X

```
> run:
> Dosya İçeriği : Java Programlama Dili []
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Eğer tüm içeriği okumak istersek böyle yapmalıyız. Fis.read hiçbir değer bulamazsa -1 döndüreceğinden -1 denk gelinceye kadar her bir harfi ve değeri s stringine ekliyoruz.

The screenshot shows an IDE interface with a code editor and a terminal window. The code in the editor is as follows:

```
int say = 0;
fis.skip(5);

while ((deger = fis.read()) != -1) {
    s += (char) deger;
    say++;
    if (say == 10) {
        break;
    }
}
System.out.println("Dosyanın 5.Yerinden İtibaren 10 karakter : " + s);

/*while ((deger = fis.read()) != -1) {
```

The terminal window below shows the output of the program:

```
in 5.Yerinden İtibaren 10 karakter : Programlam
SUCCESSFUL (total time: 0 seconds)
```

- Sadece belirli bir kısmı okumak istersek bu şekilde yapabiliriz.

157. Mini Proje - Mp3 Kopyalama

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {
    private static ArrayList<Integer> icerik = new ArrayList<Integer>();

    public static void dosyaOku() {
        try {
            FileInputStream in = new FileInputStream("marg.mp3");//mp3 aldık.
            int oku;
            while ((oku = in.read()) != -1) {
                icerik.add(oku);
            }
            //int şeklinde mp3'ü oku değişkenine aktardık ve oradan ArrayList'e gönderdik.
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IOException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void kopyala(String dosyaismi) {
        try {
            FileOutputStream out = new FileOutputStream(dosyaismi);//dosyayı dışarı aktarmak kopyalamak için bunu kullanıyoruz.

            for (int deger : icerik) {
                out.write(deger);
            }
            //ArrayList'teki değerlerin içeriğini de içeriğe gönderdik.
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IOException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void main(String[] args) {
        dosyaOku();
        long baslangic = System.currentTimeMillis();
        kopyala("marg2.mp3");
        kopyala("marg3.mp3");
        kopyala("marg4.mp3");
        long bitis = System.currentTimeMillis();
        System.out.println("3 dosyanın kopyalanması su kadar sürdü : " + ((bitis - baslangic) / 1000) + "saniye");// Dosyanın okunma süresini tuttuk.
    }
}
```

158. FileWriter ile Dosyalara Veri Yazmak

- .txt uzantılı dosyalarda bunları kullanmak daha faydalıdır.
 - FileOutPut ile dosyalara veri yazarken byte çeviriyoruz. Bunda ise öyle birşeye gerek kalmıyor.

```
FileWriter writer = null;

try {
    writer = new FileWriter("dosya.txt");
} catch (IOException ex) {
    System.out.println("Dosya açılırken IOException oluştu...");
}
finally {

    if (writer != null) {
        try {
            writer.close();
        } catch (IOException ex) {
            System.out.println("Dosya Kapatılırken hir hara oluştu...");
        }
    }
}
```



```
    } catch (IOException ex) {
        System.out.println("Dosya açılırken IOException oluştu...");
    }
}
finally {
    if (writer != null) {
        try {
            writer.close();
        } catch (IOException ex) {
            System.out.println("Dosya Kapatılırken bir hata oluştu...");
        }
    }
}
```

- Dosyamızı oluşturduk.

```
FileWriter writer = null;  
  
try {  
    writer = new FileWriter("dosya.txt");  
  
    writer.write("Mustafa Murat Coşkun");  
    writer.write("Mehmet Gençol");  
  
} catch (IOException ex) {  
    System.out.println("Dosya açılırken IOException oluştu...");
```

A screenshot of an IDE showing a code editor window. The tab bar at the top has 'Source' selected. The code in the editor is:

```
1 Mustafa Murat CoşkunMehmet Gençol
```

- İçerik olarak böyle yazdığımızda \n yazmadığımız için yanyana olacaklar ve true eklemediğimiz içimiz dosyanın içeriği sonradan silinecek.

```
FileWriter writer = null;

try {
    writer = new FileWriter("dosya.txt",true);

    writer.write("Mustafa Murat Coşkun\n");
    writer.write("Mehmet Gençol\n");

} catch (IOException ex) {
    System.out.println("Dosya açılırken IOException oluştu...");
}
```



A screenshot of an IDE showing the output of the code execution. The code has been run, and the output is displayed in the editor:

```
1 Mustafa Murat Coşkun
2 Mehmet Gençol
```

- Görüldüğü gibi düzeldi.

159. Java 7 ile Beraber Gelen Try With Resource Kullanımı

- Sürekli try-catch yapısı yazmak zorunda kalıyoruz. Bu try catch yapısının sayısını azaltabiliriz.

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

    try(FileWriter writer = new FileWriter("dosya.txt")){
        writer.write("Deneme");

    } catch (IOException ex) {
        System.out.println("Dosya oluşturulurken hata oluştu....");
    }
}
```

- Finally kısmını kendimiz yazmamıza gerek kalmadan bu şekilde kullanırsak dosya oluşturulacak ve içine Deneme yazılacaktır.

```
try(FileWriter writer1 = new FileWriter("dosya.txt");
    FileWriter writer2 = new FileWriter("dosya2.txt"))
{
```

- Bu yapıda birden çok dosya oluşturmak istersek bu şekilde yapabiliriz.

```
try(FileWriter writer1 = new FileWriter("diller.txt")){
    Scanner scanner = new Scanner(System.in);
    String dil;

    while (true) {

        System.out.println("Bir dil giriniz:");
        dil = scanner.nextLine();

        writer1.write(dil + "\n");
        if (dil.equals("-1")) {
            System.out.println("Programdan Çıkılıyor...");
            System.out.println("Dosyayı Kontrol Edin...");
            break;
    }
}
```



Writer.write(dil+"\n") break; } altında olacak bu şekilde olursa -1 yazdığımızda oda diller.txt yazılıyor.

```
26
27
Output - TryWithResource (run) X
run:
Bir dil giriniz:
Java
Bir dil giriniz:
Python
Bir dil giriniz:
Go
Bir dil giriniz:
Php
Bir dil giriniz:
Javascript
Bir dil giriniz:
-1
Programdan Çıkılıyor...
Dosyayı Kontrol Edin...
BUILD SUCCESSFUL (total time: 27 seconds)
```

```
<default config>
Main.java X diller.txt X
Source History | 
1 Java
2 Python
3 Go
4 Php
5 Javascript
6
```

- 1 deyince programdan çıkıştıktan kullanıcıdan aldığı değeri diller.txt yazan programı yazdık.

160. Mini Proje - İdman Programı Log Dosyası Oluşturma

161. FileReader, BufferedReader Okuma İşlemleri ve BufferedWriter ile Yazma İşlemleri

```
Main.java X ogrenciler.txt X
Source History | 
1 Mustafa Murat Coşkun, Bilgisayar Mühendisliği
2 Oğuz Artıran, Finansal Matematik
3 Mehmet Gençol, Bilgisayar Mühendisliği
4 Yusuf Mücahit Çetinkaya, İşletme
5 Görkem Özer, İstatistik
6 Semih Aktaş, Bilgisayar Mühendisliği
7 Hasan Bayhan, İşletme
8
```

- Öğrenciler.txt önceden oluşturuldu.

```
public static void main(String[] args) {
    try(Scanner scanner=new Scanner(new FileReader("ogrenciler.txt"))){
        while(scanner.hasNextLine()){
            System.out.println("Okunan Satır: " +scanner.nextLine());
        }
    } catch (FileNotFoundException ex) {
        System.out.println("Dosya bulunamadı");
    } catch (IOException ex) {
        System.out.println("dosya açılırken sorun oluştu.");
    }
}
```

- Bu şekilde yaprak consoledan değilde direkt txt dosyasından okuyoruz. File reader'da bütün bir satırı okuma metodu olmadığı için scanner sınıfını kullanıyoruz. .hasNextLine alt satır olup olmadığını öğrenirken nextLine da bütün bir satırı okuyor.

```

public static void main(String[] args) {
    try(Scanner scanner=new Scanner(new FileReader("ogrenciler.txt"))){
        while(scanner.hasNextLine()){
            String ogrenci_bilgisi=scanner.nextLine();

            String[] array=ogrenci_bilgisi.split(", ");
            if(array[1].equals("Bilgisayar mühendisliği")){
                System.out.println("Öğrenci bilgisi"+ogrenci_bilgisi);
            }
        }
    } catch (FileNotFoundException ex) {
        System.out.println("Dosya bulunamadı");
    } catch (IOException ex) {
        System.out.println("dosya açılırken sorun oluştu.");
    }
}

```

```

run:
Öğrenci Bilgisi: Mustafa Murat Coşkun,Bilgisayar Mühendisliği
Öğrenci Bilgisi: Mehmet Gençol,Bilgisayar Mühendisliği
Öğrenci Bilgisi: Semih Aktaş,Bilgisayar Mühendisliği
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Eğer belli bir kısmı almak istiyorsak böyle yapmalıyız. Burada sadece bilgisayar mühendisliği okuyanların adını almak istedik. O yüzden üstte görüldüğü gibi tüm öğrenciler okuduğu bölüm isimden sonra , ile ayrılmış. Önce String arrayine her virgülden önceki ve sonraki cümleyi atıyoruz. Mesela önce mustafa murat çoskun arrayin 1. indexinde olmuş oluyor. Bu bilgisayar mühendisliğine eşit olmadığı için sonraki cümle bilgisayar mühendisliğine geçiyor. O eşit olduğu için bu cümleyi ekrana yazdırıyor. BufferedReader da aynı mantıkta işliyor ancak çok daha işlevsel.
- **BufferedReader**, bir girdi akışındaki metni (bir dosya gibi) karakterleri, dizileri veya satırları sorunsuz bir şekilde okuyan karakterleri tamponlayarak okuyan Java sınıfıdır.
- FileReader ve BufferedReader arasındaki fark ise şöyle;
FileReader bir cümleyi okurken sadece bir harf okuyor sonra programa geri geliyor. Dosyaya gidip bir harf daha okuyor sonra tekrar geri geliyor.
BufferedReader ise bütün bir cümleyi okuduğu için dosyaya daha az erişim ve bu yüzden program daha hızlı oluyor.

```

try(Scanner scanner = new Scanner(new BufferedReader(new FileReader("ogrenciler.txt")))) {
    while (scanner.hasNextLine()) {

        String ogrenci_bilgisi = scanner.nextLine();

        String[] array = ogrenci_bilgisi.split(", ");
        if (array[1].equals("Bilgisayar Mühendisliği")) {
            System.out.println("Öğrenci Bilgisi: " + ogrenci_bilgisi);
        }
    }
}
derBufferedReaderveWriter (run) ×

Bilgisi: Mustafa Murat Coşkun,Bilgisayar Mühendisliği
Bilgisi: Mehmet Gençol,Bilgisayar Mühendisliği
Bilgisi: Semih Aktaş,Bilgisayar Mühendisliği
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Kullanmak için ise sadece yukarıya BufferedReader yazmamız gerekiyor.

```

try(BufferedWriter writer = new BufferedWriter(new FileWriter("ogrenciler.txt",true))){
    writer.write("Ali Ozan,İnşaat Mühendisliği\n");
}

} catch (IOException ex) {
    System.out.println("Dosya açılırken hata oluştu...");
}

```

- BufferedWriter kullanarak yazmaya böyle yapabiliriz.

162. Mini Proje - Dosyadan Okuyarak Not Hesaplama

```

Hatice Günday,70,90,20
Mustafa Akyürek,90,80,60
Ramazan Topaloğlu,60,30,50
Elif Akşit,80,40,80
Mehmet Düşenkalkar,70,20,40
Hatice Dağdaş,90,90,80
Merve Tütüncü,40,30,30
Hatice Pekkan,50,70,60
Merve Alyanak,70,60,30
Meryem Aşikoğlu,20,80,70
Süleyman Uluhan,40,10,100
Esra Ekici,10,10,30
Emine Kumcuoğlu,90,30,80
Süleyman Kumcuoğlu,50,10,60
Şerife Kumcuoğlu,40,70,70
Sude Tekand,90,50,90
Elif Nebioğlu,10,10,60
Sude Berberoğlu,10,100,50
Yasemin Durmaz,40,70,10
Fadime Akyürek,50,20,100
Fatma Akşit,60,40,90
Emre Sözeri,50,50,20
Yasemin Kurutluoğlu,40,50,40
Fatma Akman,80,20,30
Hacer Korol,80,50,90
Recep Altınbaş,40,70,80
Murat Numanoğlu,70,100,100
Fatih Fahri,40,10,10
Halil Sarıoğlu,30,70,60
Merve Arıcan,30,30,70
İsmail Kaplangı,60,50,10
Emir Eçeli.80.20.80

```

81 öğrencinin bu şekilde notlarının olduğu dosya.tt var.

```

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {
    public static String harfNotuHesapla(String isim,int vizel,int vize2,int finalnot) {
        String cikti="";
        double toplamnot=(vizel*3/10.0)+(vize2*3/10.0)+(finalnot*4/10.0);
        if(toplamnot>=90)cikti=isim+"budersten aa aldi";
        else if(toplamnot>=85)cikti=isim+"budersten ba aldi";
        else if(toplamnot>=80)cikti=isim+"budersten bb aldi";
        else if(toplamnot>=75)cikti=isim+"budersten cb aldi";
        else if(toplamnot>=70)cikti=isim+"budersten cb aldi";
        else if(toplamnot>=65)cikti=isim+"budersten dc aldi";
        else if(toplamnot>=60)cikti=isim+"budersten dd aldi";
        else if(toplamnot>=55)cikti=isim+"budersten fd aldi";
        else cikti=isim+"budersten ff aldi";

        return cikti;
    }
    public static void main(String[] args) {
try(Scanner scanner=new Scanner(new FileReader("dosya.txt"));
    FileWriter writer=new FileWriter("harfnotlari.txt")){
        while (scanner.hasNextLine()){
            String ogrencBilgileri=scanner.nextLine();

            String[] ogrenciArray=ogrencBilgileri.split(",");
            int vizel=Integer.valueOf(ogrenciArray[1]);
            int vize2=Integer.valueOf(ogrenciArray[2]);
            int finalnot=Integer.valueOf(ogrenciArray[3]);
            String cikti=harfNotuHesapla(ogrenciArray[0],vizel,vize2,finalnot);
            writer.write(cikti+"\n");
        }
    } catch (FileNotFoundException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}

```

Hatice Fekkan bu dersten DC Aldi...
 Merve Alyenak bu dersten FF Aldi...
 Meryem Açıkoğlu bu dersten FD Aldi...
 Suleyman Uluhan bu dersten FD Aldi...
 Ebru Ekici bu dersten FF Aldi...
 Enine Kumcuoğlu bu dersten DC Aldi...
 Soleyman Kumcuoğlu bu dersten FF Aldi...
 Şerife Kumcuoğlu bu dersten DC Aldi...
 Sude Tekindu bu dersten CB Aldi...
 Elif Nebioğlu bu dersten FF Aldi...
 Sude Berberoğlu bu dersten FF Aldi...
 Yasemin Durmaz bu dersten FF Aldi...
 Fadime Akyurek bu dersten DC Aldi...
 Fatma Akçit bu dersten FF Aldi...
 Emre Bozert bu dersten FF Aldi...
 Yasemin Muratlıoğlu bu dersten FF Aldi...
 Fatih Arslan bu dersten FF Aldi...
 Hacer Horol bu dersten DC Aldi...
 Recep Altınbaş bu dersten DC Aldi...
 Murat Hımanoğlu bu dersten AA Aldi...
 Fatih Fahri bu dersten FF Aldi...
 Halil Sarıoğlu bu dersten FF Aldi...
 Merve Arıcan bu dersten FF Aldi...
 İsmail Kaplanlı bu dersten FF Aldi...
 Emir Egeli bu dersten DC Aldi...
 Hayva Erbulak bu dersten FF Aldi...
 Hacer Okumus bu dersten CB Aldi...

Serialization Nedir ?

Java, **Serialization API** sayesinde **Serializable** interfaceini implemente eden classların objelerini bir byte dizisine dönüştürüp bir dosyaya kaydetme imkanı verir. Javada bu işlemeye **objelerin serileştirilmesi** yani **serialization** adı verilir. Sonradan ise bu objeleri dosyadan okumak için yaptığımız işlemeye işlemeye ise **deserialization** adı verilir.

Serialization Avantajları

- Objelerin statelerini(antalik durumlarını) ve özelliklerini daha sonra kullanmak için saklamak istiyorsak serialization kullanabiliriz. Örneğin oyunlarda kaydettiğimiz yerden devam etmek istiyorsak serialization mantiği kullanılıyor diyebiliriz.
- 2 platform arasındaki veri alışverişini objeler üzerinden yapmak istiyorsak objelerimizi serileştirip dosya transferi yapabiliriz. Örneğin bir chat uygulamasında mesajları obje olarak düşünürsek bunları serileştirme yolu ile transfer edebiliriz.
- Bir objenin olması çok uzun sürüyorsa ve daha sonradan bu objeyi kullanmak istiyorsak bir daha bu objeyi oluşturmak yerine bu objeyi serileştirerek daha sonra kullanabiliriz.

- Mesela veri tabanından veri çekmek istiyorsak ancak veriler çok uzunsa bir kere çekip mesela 10 dk bekliyoruz bunları bir objenin içine gömüp daha sonra istediğimiz yerde bu objeyi kullanıp içindeki verileri kullanıyoruz. Böylece tekrar 10 dk beklemek zorunda kalmıyoruz.

```

public class Ogrenci implements Serializable {

    private String isim;
    private int id;
    private String bolum;

    public Ogrenci(String isim, int id, String bolum) {
        this.isim = isim;
        this.id = id;
        this.bolum = bolum;
    }

    Generate
    Constructor...
    Logger...
    Getter...
    Setter...
    Getter and Setter...

}

    public Ogrenci(String isim, int id, String bolum) {
        this.isim = isim;
        this.id = id;
        this.bolum = bolum;
    }

    @Override
    public String toString() {

        String bilgiler = "Öğrenci İsmi :" + isim +
                         "\nÖğrenci Numarası : " + id +
                         "\nÖğrenci Bölüm : " + bolum;

        return bilgiler;
    }

}

```

- Eğer bir objeyi serileştirmek istiyorsan onun classını Serializable interfaceni implemente etmesi gerekiyor.

Objeyi Yazma

```
try(ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("ogrenci.bin"))){  
    Ogrenci ogrenci1 = new Ogrenci("Mustafa Murat",1234, "Bilgisayar Mühendisliği");  
    Ogrenci ogrenci2 = new Ogrenci("Öğuz",678, "Finansal Matematik");  
  
    out.writeObject(ogrenci1);  
    out.writeObject(ogrenci2);  
  
} catch (FileNotFoundException ex) {  
    System.out.println("Dosya Bulunamadı...");  
} catch (IOException ex) {  
    System.out.println("Dosya açılırken IOException Oluştu...");  
}  
}
```



- **.writeObject:** Objeleri dosyaya yazmaya yarar. Okurken de yazma sırasında göre okumalıyız. (Önce ogrenci1 yazdık sonra ogrenci2. Okurken önce ogrenci 1 okumalıyız.)
- Ogrenci1 ve ogrenci2 değerlerini öğrenci.bin'e kaydettik. Bundan sonra eğer bu bilgileri kullanmak istersek tekrar bağlanmak yerine direkt dosyadan bilgileri çekebileceğiz.

```
public class ObjeyiOku {  
    public static void main(String[] args) {  
  
        try(ObjectInputStream in = new ObjectInputStream(new FileInputStream("ogrenci.bin"))){  
            Ogrenci ogrenci1 = (Ogrenci)in.readObject();  
            Ogrenci ogrenci2 = (Ogrenci) in.readObject();  
  
            System.out.println("*****[*****");  
            System.out.println(ogrenci1);  
            System.out.println("**********");  
            System.out.println(ogrenci2);  
  
        } catch (FileNotFoundException ex) {  
            System.out.println("Dosya bulunamadı...");  
        } catch (IOException ex) {  
            System.out.println("Dosya açılırken IOException oluştu....");  
        } catch (ClassNotFoundException ex) {  
    }  
}
```

Serialization1 (run) X

run:
BUILD SUCCESSFUL (total time: 0 seconds)

- **.readObject:** Dosyadaki nesneleri çekmeye yarar.
- Dosyadan bilgileri çekmek için bu şekilde yapıyoruz ve im.readObjectden önce Ogrenci yazarak tür dönüşümü yapmalıyız.

```
1 public class Ogrenci implements Serializable {
2     private static final long serialVersionUID = 1000;
3
4     private String isim;
5     private int id;
6     private String bolum;
7
8     public Ogrenci(String isim, int id, String bolum) {
9         this.isim = isim;
10        this.id = id;
11        this.bolum = bolum;
12    }
13
14    @Override
```



- Mevcut olan projelerde veya kopyala-yapıştır yaptığımız sınıflarda bazen “serialVersionUID” gibi long tipinde bir değişken tanımladığını görürüz.Çoğu zaman kod okurken bu satırı görmezden geliriz.Bu değişken Serializable interface’ini implement eden sınıflarda genelde derleyici “Adds a default serial version ID to the selected type.” benzeri bir uyarı verir.Bu uyarı,jvm sürüm kontrolü sağlamak için bir versiyon id’ye ihtiyaç duyar ve bunu tanımlamazsa jvm runtime’da otomatik olarak bunu üretir ve hataya sebebiyet verebilir.Bu sebeple Serializable sınıfından türetilen sınıflarda SerialVersionUID değişken tanımı kullanılır.
- serialVersionUID** classımızın o anki versiyonunu veren bir değişkendir. Eğer sonradan classımızı değiştirirsek bu javaya classımızı değiştirdiğimizi söylememizin bir yoludur. Bu sayede bazı kazalardan kurtulmuş olabiliriz. Burada classımızın kodlarının bu versiyonunun 1000 olduğunu söyledik. Eğer sonradan classı değiştirirsek bu serialVersionUID değiştirmeliyiz ki javaya classın değiştirildiğini söyleyelim.

```
1 import java.io.Serializable;
2
3 public class Ogrenci implements Serializable {
4     private static final long serialVersionUID = 2000;
5
6     private String isim;
7     private int id;
8     private String bolum;
9     private String dersler;
10
11
12     public Ogrenci(String isim, int id, String bolum) {
13         this.isim = isim;
```

- Dersler diye bir değişken ekleyip serialVersionUID’yi 2000 yaparak classımızın bu kodlarla versiyonunun 2000 olduğunu söyledik.

```

10  public class ObjeyiOku {
11      public static void main(String[] args) {
12
13          try(ObjectInputStream in = new ObjectInputStream(new FileInputStream("ogrenci.bin"))){
14              Ogrenci ogrenci1 = (Ogrenci)in.readObject();
15              Ogrenci ogrenci2 = (Ogrenci) in.readObject();
16
17              System.out.println("*****");
18              System.out.println(ogrenci1);
19              System.out.println("*****");
20              System.out.println(ogrenci2);
21
22

```

Output - Serialization1 (run) ×

```

run:
Dosya açılırken IOException oluştu...
BUILD SUCCESSFUL (total time: 0 seconds)

```

Tekrardan deminki öğrencileri oluşturduğumuzda hata alıyoruz. Bu sen 1000 için oluşturduğum ObjeyiOku kodunu 2000 için çalıştırırmaya çalışıyorsun demek oluyor.

164. Arrayleri ve Collectionları Serileştirmek

```

public class ObjeyiYaz {
    public static void main(String[] args) {
        Ogrenci ogrenci1 = new Ogrenci("Mustafa Murat",1234, "Bilgisayar Mühendisliği");
        Ogrenci ogrenci2 = new Ogrenci("Öğuz",678, "Finansal Matematik");
        Ogrenci ogrenci3 = new Ogrenci("Mehmet",123, "Bilgisayar Mühendisliği");

        Ogrenci[] ogrenci_array = {ogrenci1,ogrenci2,ogrenci3};
        ArrayList<Ogrenci> ogrenci_list = new ArrayList<Ogrenci>(Arrays.asList(ogrenci_array));

        try(ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("ogrenciler.bin"))){

        } catch (FileNotFoundException ex) {
            System.out.println("Dosya bulunamadı...");
        } catch (IOException ex) {
            System.out.println("Dosya açılırken IOException Oluştı...");
        }
}

```

- Serialization2 (run) ×

- Bir array ve bir arraylist oluşturduk. Eğer bir arraydeki değerleri Arrayliste atmak istersek bunun için **.asList** komutunu kullanabiliriz. Öğrenci nesnelerini oluşturduktan sonra bunları görüldüğü gibi arraye attık. Sonradan arraykiste bunları **Arrays** sınıfının **.asList** kurucu metodunu kullanarak attık.

```
public class ObjeyiYaz {
    public static void main(String[] args) {
        Ogrenci ogrenci1 = new Ogrenci("Mustafa Murat", 1234, "Bilgisayar Mühendisliği");
        Ogrenci ogrenci2 = new Ogrenci("Oğuz", 678, "Finansal Matematik");
        Ogrenci ogrenci3 = new Ogrenci("Mehmet", 123, "Bilgisayar Mühendisliği");

        Ogrenci[] ogrenci_array = {ogrenci1, ogrenci2, ogrenci3};
        ArrayList<Ogrenci> ogrenci_list = new ArrayList<Ogrenci>(Arrays.asList(ogrenci_array));

        try(ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("ogrenciler.bin"))){

            out.writeObject(ogrenci_array);
            out.writeObject(ogrenci_list);
        } catch (FileNotFoundException ex) {
            System.out.println("Dosya bulunamadı...");
        } catch (IOException ex) {
            System.out.println("Dosya açılırken IOException Oluştu...");
        }
    }
}
```

ut - Serialization2 (run) ×

```
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Bu şekilde öğrenciler.bin oluşturup içine değerlerimizi yazdık.

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

        try(ObjectInputStream in = new ObjectInputStream(new FileInputStream("ogrenciler.bin"))){

            Ogrenci[] ogrenci_array = (Ogrenci[])in.readObject();
            ArrayList<Ogrenci> ogrenci_list = (ArrayList<Ogrenci>)in.readObject();

            System.out.println("*****");
            for (Ogrenci o : ogrenci_array) {
                System.out.println(o);
                System.out.println("-----");
            }
            System.out.println("*****");
            for (Ogrenci o : ogrenci_list) {
                System.out.println(o);
                System.out.println("-----");
            }
        }
    }
}
```

ut - Serialization2 (run) ×

- Önce .readObject ile dizimizi ve arraylistimizi okuduk ve bunları ekrana yazdırıldı.

```

Output - Serialization2 (run) ×

Öğrenci İsmi :Mustafa Murat
Öğrenci Numarası : 1234
Öğrenci Bölüm : Bilgisayar Mühendisliği
-----
Öğrenci İsmi :Oğuz
Öğrenci Numarası : 678
Öğrenci Bölüm : Finansal Matematik
-----
Öğrenci İsmi :Mehmet
Öğrenci Numarası : 123
Öğrenci Bölüm : Bilgisayar Mühendisliği
-----
*****
Öğrenci İsmi :Mustafa Murat
Öğrenci Numarası : 1234
Öğrenci Bölüm : Bilgisayar Mühendisliği
-----
Öğrenci İsmi :Oğuz
Öğrenci Numarası : 678
Öğrenci Bölüm : Finansal Matematik
-----
Öğrenci İsmi :Mehmet

```

- Çıktısı bu şekildedir.
- ArrayList Serializable sınıfını implemente ediyor. O yüzden hata vermedi.

165. Transient Anahtar Kelimesi ve Statik Alanları Serileştirmek

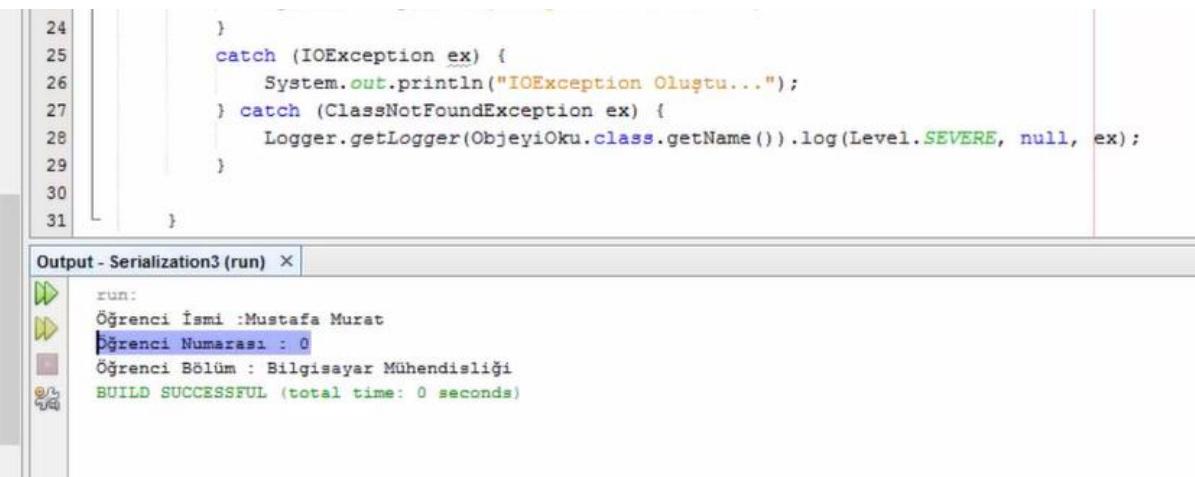
- Kelime anlamı geçici, fani olan **transient** anahtar kelimesi, önüne geldiği veri alanının serileştirme (serialization) işleminden muaf tutulmasını ve verinin dosyaya keydedilmemesini sağlar.

```

Run Debug Profile Tools Window Help
fault config > 
Source History 
1 import java.io.Serializable;
2
3 public class Ogrenci implements Serializable {
4
5     private String isim;
6     private transient int id;
7     private String bolum;
8
9     public Ogrenci(String isim, int id, String bolum) {
10        this.isim = isim;
11        this.id = id;
12        this.bolum = bolum;
13    }
14
15    @Override
16    public String toString() {
17
18        String bilgiler = "Öğrenci İsmi :" + isim +
19        "\nÖğrenci Numarası : " + id +

```

- Dosyanın boyutu çok büyük olmaması için bizim programımız için degersiz bazı bilgileri serilizasyon yapmak istemeyebiliriz. Bunun için transient kullanırız. Burada id değerinde kullanarak bunu söylüyoruz.



```

24     }
25     catch (IOException ex) {
26         System.out.println("IOException Oluştu...");
27     } catch (ClassNotFoundException ex) {
28         Logger.getLogger(ObjeyiOku.class.getName()).log(Level.SEVERE, null, ex);
29     }
30 }
31 }
```

Output - Serialization3 (run) ×

```

run:
Öğrenci İsmi :Mustafa Murat
Öğrenci Numarası : 0
Öğrenci Bölüm : Bilgisayar Mühendisliği
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Dosyayı tekrar okuduğumuzda id'yi serleştirmedigimiz için varsayılan bir değer gösteriliyor. Mesela int varsayılan değer 0 olduğu için numara sıfır olarak yazıyor. String bir değere yazsaydık null yazacaktı.
- Static bir değeri serileştiremezsin. Çünkü static alan sınıfı özgürdür. Mehoda ait değildir. O yüzden serileştirilemez.

166. Mini Proje - Serialization ile Hafıza Oyununu Kaydetme

```
|  
| import java.io.FileInputStream;  
| import java.io.FileOutputStream;  
| import java.io.IOException;  
| import java.io.ObjectInputStream;  
| import java.io.ObjectOutputStream;  
| import java.util.logging.Level;  
| import java.util.logging.Logger;  
  
|  
| public class OyunKayit {  
|  
|     public static void oyunKaydet(Kart[][] kartlar) {  
|  
|         try(ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("kayit.bin"))){  
|             System.out.println("Oyun kaydediliyor...");  
|  
|             out.writeObject(kartlar);  
|  
|         } catch (IOException ex) {  
|             Logger.getLogger(OyunKayit.class.getName()).log(Level.SEVERE, null, ex);  
|         }  
|  
|     }  
|  
|     public static Kart[][] kayittanAl() {  
|  
|         try(ObjectInputStream in = new ObjectInputStream(new FileInputStream("kayit.bin"))){  
|  
|             Kart[][] cikti = (Kart[][] ) in.readObject();  
|  
|             return cikti;  
|  
|         } catch (IOException ex) {  
|             Logger.getLogger(OyunKayit.class.getName()).log(Level.SEVERE, null, ex);  
|         } catch (ClassNotFoundException ex) {  
|             Logger.getLogger(OyunKayit.class.getName()).log(Level.SEVERE, null, ex);  
|         }  
|         return null;  
|     }  
|  
| }
```

```
| import java.io.Serializable;  
|  
|=public class Kart implements Serializable{  
|  
|     private char deger;  
|     private boolean tahmin = false;  
|  
|     public Kart(char deger) {  
|         this.deger = deger;  
|     }  
|  
|     public char getDeger() {  
|         return deger;  
|     }  
|  
|     public void setDeger(char deger) {  
|         this.deger = deger;  
|     }  
|  
|     public boolean isTahmin() {  
|         return tahmin;  
|     }  
|  
|     public void setTahmin(boolean tahmin) {  
|         this.tahmin = tahmin;  
|     }  
|  
|}
```

```
import java.io.File;
import java.util.Scanner;
public class Main {
    private static Kart[][] kartlar = new Kart[4][4];
    public static void kayittanAl() {
        File file = new File("kayit.bin");
        Scanner scanner = new Scanner(System.in);
        if (file.exists()) {
            System.out.print("Kaydedilmiş bir oyununuz var. Kayittan devam etmek ister misiniz ? (yes ya da no)");
            String cevap = scanner.nextLine();

            if (cevap.equals("yes")) {
                kartlar = OyunKayit.kayittanAl();
                return;
            }
        }
        kartlar[0][0] = new Kart('E');
        kartlar[0][1] = new Kart('A');
        kartlar[0][2] = new Kart('B');
        kartlar[0][3] = new Kart('F');
        kartlar[1][0] = new Kart('G');
        kartlar[1][1] = new Kart('A');
        kartlar[1][2] = new Kart('D');
        kartlar[1][3] = new Kart('H');
        kartlar[2][0] = new Kart('F');
        kartlar[2][1] = new Kart('C');
        kartlar[2][2] = new Kart('D');
        kartlar[2][3] = new Kart('H');
        kartlar[3][0] = new Kart('E');
        kartlar[3][1] = new Kart('G');
        kartlar[3][2] = new Kart('B');
        kartlar[3][3] = new Kart('C');
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        kayittanAl();
        while (oyunBittiMi() == false) {
            oyunTahtasi();
            System.out.print("Çıkış için q'ya basın (yes ya da no)");
            String cikis = scanner.nextLine();
            if (cikis.equals("yes")) {
                System.out.print("Oyunu kaydetmek istiyor musunuz ? (yes ya da no)");
                String kayit = scanner.nextLine();
                if (kayit.equals("yes")){
                    OyunKayit.oyunKaydet(kartlar);
                }
                else {
                    System.out.println("Oyun kaydedilmedi");
                }
                System.out.println("Programdan Çıkılıyor...");
                break;
            }
            tahminEt();
        }
    }
    public static void tahminEt() {
```

```
        }
    }

    public static void tahminEt() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Birinci Tahmin (i ve j değerlerini bir boşluklu girin...): ");
        int il = scanner.nextInt();
        int jl = scanner.nextInt();
        kartlar[il][jl].setTahmin(true);
        oyunTahtasi();
        System.out.print("İkinci Tahmin (i ve j değerlerini bir boşluklu girin...): ");
        int i2 = scanner.nextInt();
        int j2 = scanner.nextInt();
        if (kartlar[il][jl].getDeger() == kartlar[i2][j2].getDeger()) {
            System.out.println("Doğru Tahmin. Tebrikler!");
            kartlar[i2][j2].setTahmin(true);
        }
        else {
            System.out.println("Yanlış Tahmin...");
            kartlar[il][jl].setTahmin(false);
        }
    }

    public static boolean oyunBittiMi() {
        for (int i = 0 ; i < 4; i++) {
            for (int j = 0 ; j < 4 ; j++) {
                if (kartlar[i][j].isTahmin() == false) {
                    return false;
                }
            }
        }
        return true;
    }

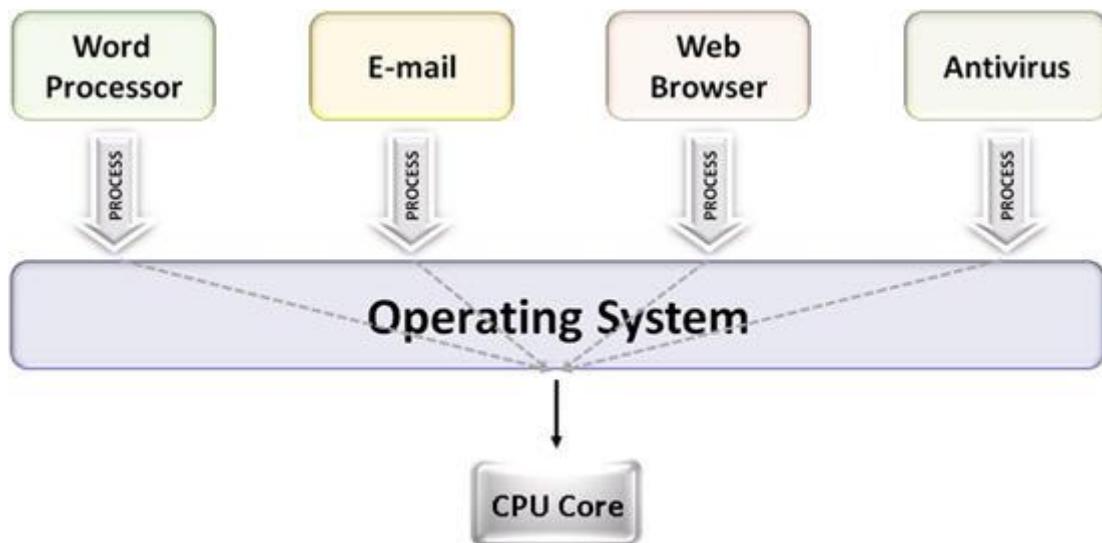
    public static void oyunTahtasi(){
        for (int i = 0 ; i < 4 ; i++) {
            System.out.println("_____");
            for (int j = 0 ; j < 4 ; j++) {

                if (kartlar[i][j].isTahmin()) {
                    System.out.print(" | " + kartlar[i][j].getDeger() + " | ");
                }
                else {
                    System.out.print(" | | ");
                }
            }
            System.out.println("");
        }
        System.out.println("_____");
    }
}
```

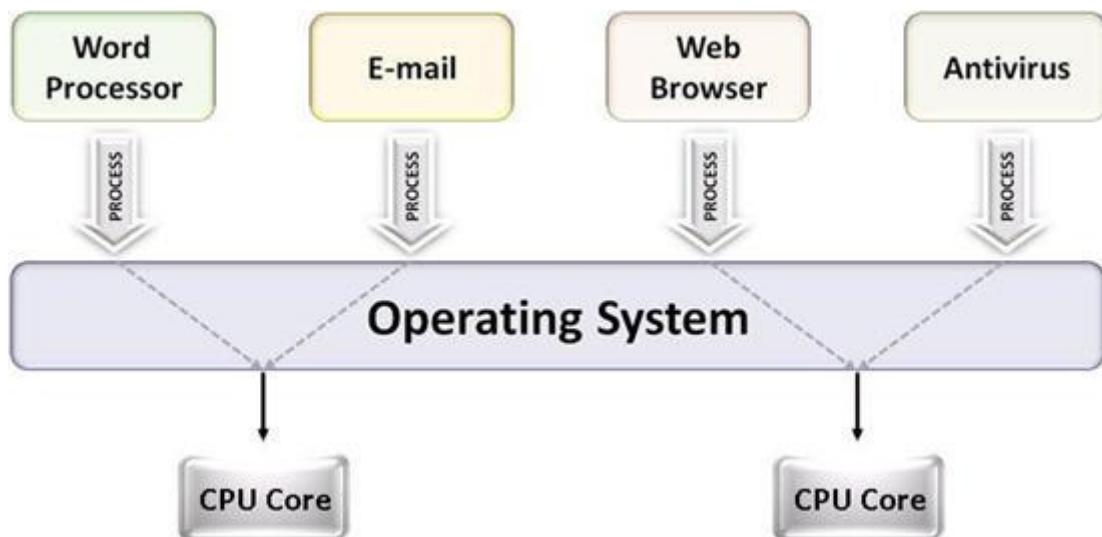
17: Multithreading ve Concurrency

167. Multitasking ve Multithreading Nedir ?

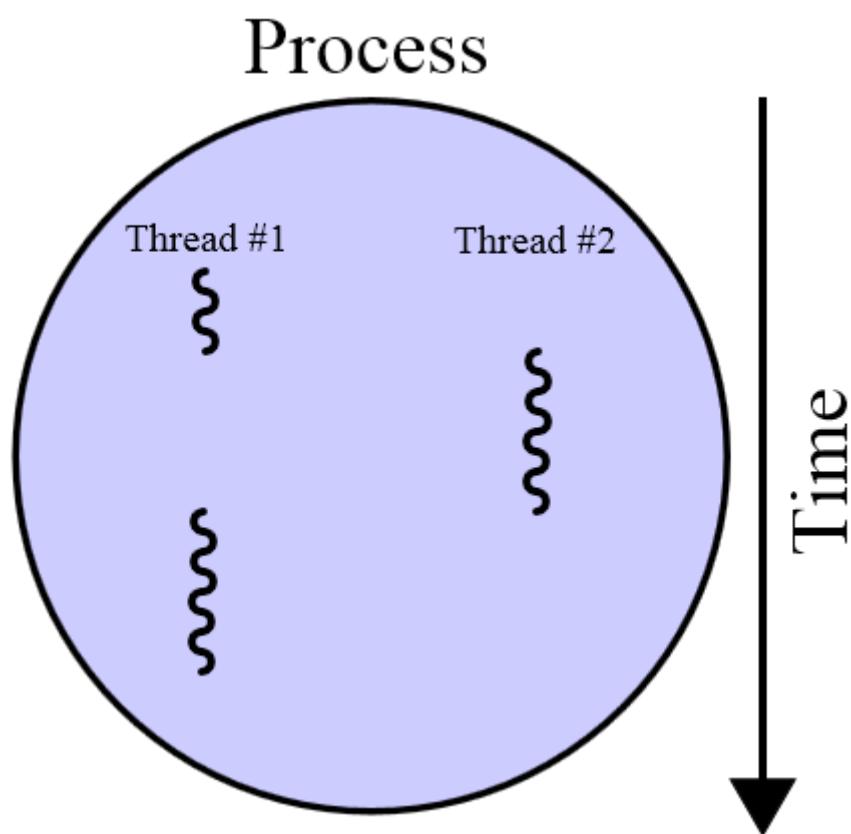
- Multitasking: Birden fazla işlemi(process) aynı anda yapabilmektir. Multitasking bir ihtiyaç olarak doğmuş. Çünkü bilgisayarda birden fazla programı çalıştırıramamak kadar kötü ne olabilir? Yani webde gezinirken Visual Studio'yu açmak için illa internet tarayıcısını kapatmamız mı gereklidir? Multitaskingden öncesinde evet kapatmamız şarttı ama multitaskinden sonra hayır. Programcılar bunu görüp işletim sistemlerini buna göre dizayn ettiler. Genel olarak birden fazla çekirdeğe sahip olan işlemciler veya birden fazla işlemciye sahip sistemlerde görülür. Peki tek çekirdekli işlemciler? Onlarda nasıl hem müzik dinleyip hem de webde gezinebiliyoruz? Bunu sağlayanda işletim sistemi. İşletim sistemi her bir işleme işlenmesi için bellir bir süre tanıyor sonra başka bir işleme süre tanıyor. Bu süre o kadar hızlıdır ki kullanıcı tarafından algılanamıyor, bir nevi sihirbazlık gibi. Kullanıcı aslında her şeyin aynı anda olduğunu zannediyor fakat aslında her zaman tek bir işlem çalışıyor. Yani aslında bilgisayar biraz müzik çalıp işlemi duraklatıyor sonra webde geziniyor sonra işlemi tekrar duraklatıp tekrar müzik çalışıyor ve böylece sürüp gidiyor. Mesela siz bu yazıyı okurken arkada Windows güncellemeleri kontrol ediyor, belki bir virüs sizin verilerinizi çalışıyor.



- Çok çekirdekli işlemcilerde ise gerçek anlamda multi tasking görülür. Herbir çekirdek birer mikro işlemci gibi davranışır ve herbir işlem (çekirdek sayısı kadar) aynı zamanda işlenebilir. (Bir nevi çoklu işlemci gibi aslında ama çoklu işlemciye göre çekirdekler birbiriyle daha hızlı etkileşebilme avantajı var ama aynı veriyolunu kullanacağından performans düşüyor.)



- Multithreading: Thread; kelime anlamı olarak ipliktir, nasıl iplikleri birleştirirince kıyafet oluşuyorsa threadleri birleştirirince işlem(process) oluşur. Multithread da multitasking gibi ihtiyaçtan doğmuştur. İşletim sistemi işlemler arası geçiş yapabiliyor fakat ya bir işlem de farklı iki işi aynı anda yapmaya kalkarsa? İşte bu olunca programlarda hatalar oluşur. Bunun önüne geçmek için işlemler threadlere ayrılıyor. Örneğin oyun oynuyorsunuz ve oynamaya devam ederken oyun otomatik olarak sizin olduğunuz yeri kaydedecek. Bunun için oyun programlanırken kaydetme işi için ayrı bir thread oluşturulur böylece işlemler paralel olarak işletilebilir.



- Özetleyecek olursak multitasking birden fazla işlemin aynı anda işlenmesi, multithreading ise bir işlemdeki birden fazla iş parçacığının aynı anda işlenmesidir.

MultiTasking ve MultiThreading Nedir ?

- MultiTasking , bilgisayarın birçok process'i (işlem) aynı anda çalıştırmasıdır. Örneğin hem Web Browser çalıştırırken aynı zamanda Spotify çalıştırması gibi
- MultiThreading ise bir process içinde bir çok çalışma ünetesi oluşturun (thread) birçok iş bir arada yapmaktadır. Örneğin, Wordde yazı yazarken aynı anda Word içinde yazdıklarımızın kontrol edilmesi buna birer örnektir.



Process ve Thread Kavramları

- **Process**, bilgisayarda çalışabilecek her türlü uygulamamızdır. Örnek: Spotify, Windows Media Player, Google Chrome
- Her process bellekte kendi **memory space'ine** (bellek alanı) sahiptir.
- Biz bir Java Uygulamasını çalıştırduğumuz zaman bu uygulama JVM(Java Virtual Machine) üzerinde çalışacak bir **process'e** dönüşür.
- Java projeleri **process'e** dönüşükleri zaman kendi **memory space'ini** veya diğer adıyla **heap'ını** oluşturur. Eğer elimizde 2 tane çalışır durumda uygulamamız(process) varsa bu uygulamalar birbirlerinin **bellek alanlarına (memory space)** ya da **heaplerine** erişemezler.

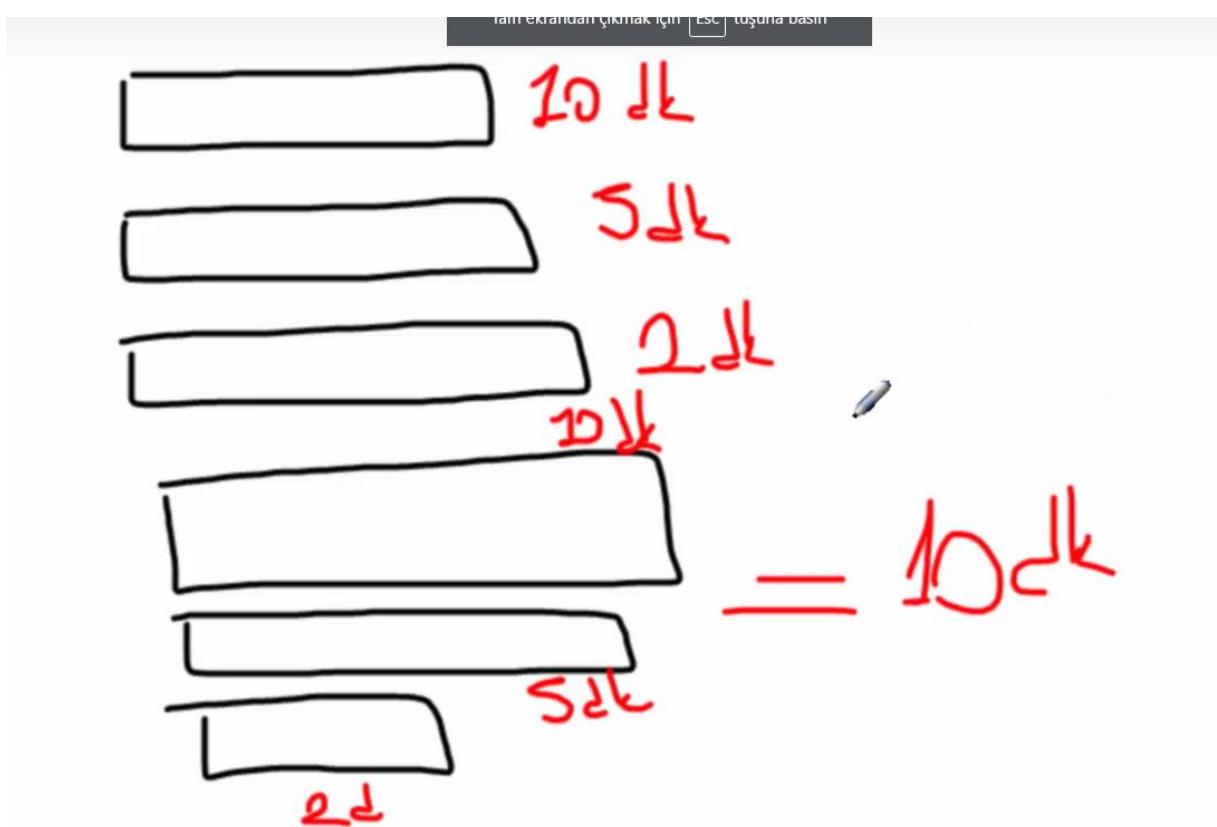


Process ve Thread Kavramları

- **Thread** ise bir process'in içinde bulunan bir çalışma ünitesidir ve her Java programı en az bir thread'e sahiptir. Eğer biz hiç **thread** oluşturmasak bile Java projeleri **main metodunu çalıştırırken** bir tane **main thread** oluşturur. Bu main threadinin yanına kendimiz değişik işlemler yaptırmak için kendi threadlerimizi oluşturabiliriz.
- Threadler processlerin içinde olduğu için processlerin oluşturduğu bellek alanına direk olarak erişim sağlayabilirler.
- Ayrıca bellek alanından ayrı olarak her threadin sadece kendinin erişebileceği bir tane **thread stack'i** bulunur.

MultiThreading'e Neden İhtiyaç Duyuyoruz ?

- Örneğin elinizde internetten bir dosya indirme gibi uzun zaman alacak bir iş var ve bu işleminden sonra yapacağınız daha bir çok iş bulunuyor. Eğer biz bu işlemi multithreading kullanmayı sadece uygulamayı **main thread** ile yazarsak, **main thread(main metod)** bu işlem bitene kadar başka herhangi bir işlem gerçekleştiremeyecek. Ancak eğer siz dosya indirme işlemini bir tane **thread** oluşturup yaparsanız, **main thread** diğer işlerine zaman ayıracaktır. Yani bizim uygulamamız bir çok işlemi paralel yapacak seviyeye gelmiş olur.
- Bunun gibi biz daha bir çok işlemimizi **threadler** yardımıyla paralel yapabiliriz ve biz **threadlerin** paralel çalışmasına **concurrency (eşzamanlık)** diyoruz.



Tek thread ile 17 dk sürecek işlem sayısını arttırdığımızda 10 dk bitiyor.

Threadlerin Çalışması

Biz programlarımızda bir çok thread oluşturduğumuzda Java Virtual Machine ve İşletim Sistemi bu threadlerin ne zaman çalışacağını ve ne zaman başlayacağını kendisi belirler. Ayrıca threadlerin çalışma sırası ve başlatılma sırası farklı olabileceği için programlarımızda ortaya çıkan çıktılar farklılık gösterebilir.

Örnek:

```
thread1.start();  
thread2.start();  
thread3.start();
```

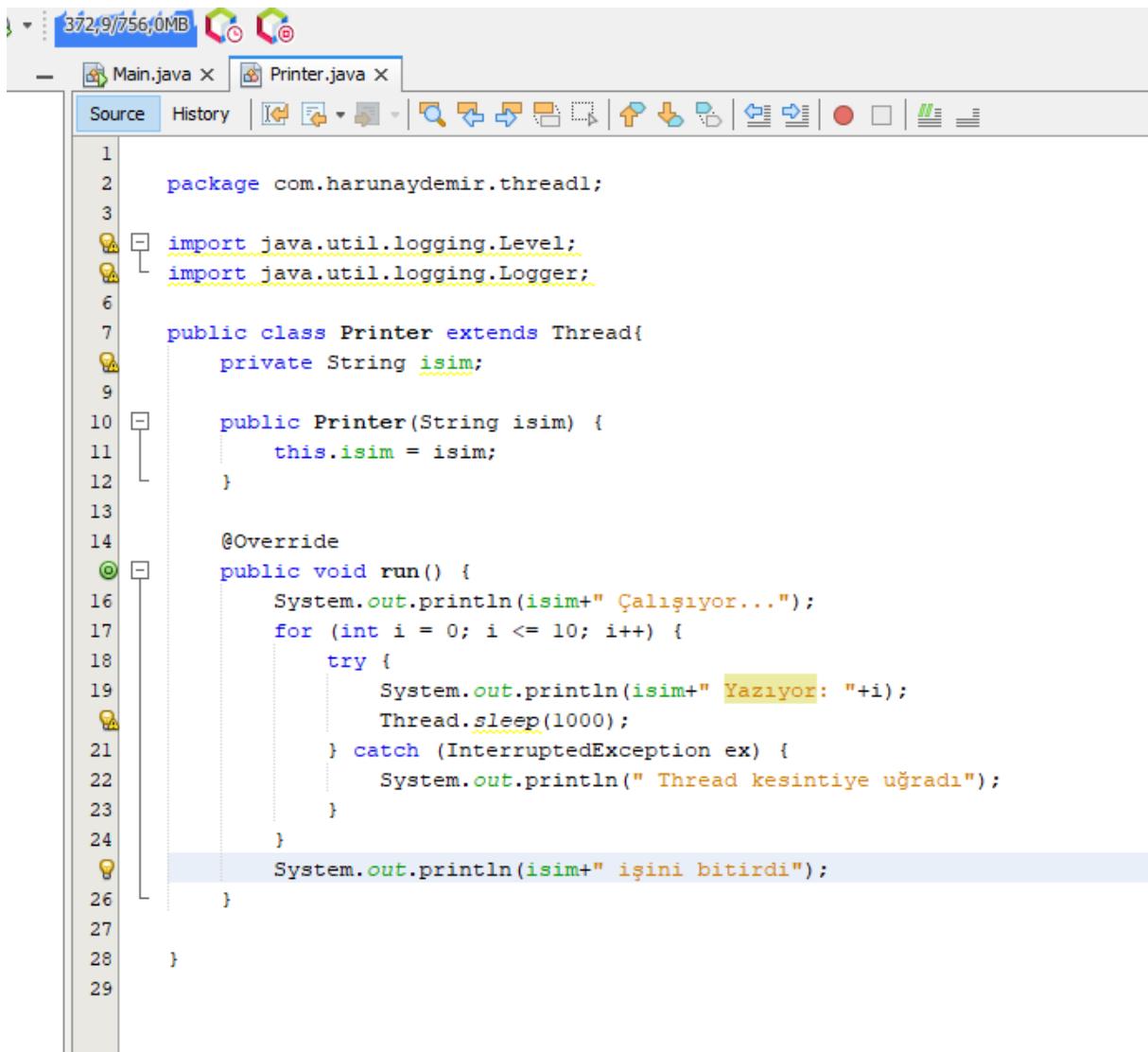
$t_2 \rightarrow t_1 \rightarrow t_3$
 $t_3 \rightarrow t_2 \rightarrow t_1$

Buradaki threadlerin işlemlerine ne zaman başlayacağı JVM'e ve İşletim Sistemine bağlıdır.

168. Thread Oluşturma Yöntemleri , Thread Sınıfı ve Runnable Interface

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.
- Thread oluşturmak için iki farklı yol izleyebiliriz. 1. Yol classı'a Thread extends etmek.



The screenshot shows a Java IDE interface with two tabs open: Main.java and Printer.java. The Printer.java tab is active, displaying the following code:

```
1 package com.harunaydemir.thread1;
2
3 import java.util.logging.Level;
4 import java.util.logging.Logger;
5
6 public class Printer extends Thread{
7     private String isim;
8
9     public Printer(String isim) {
10         this.isim = isim;
11     }
12
13
14     @Override
15     public void run() {
16         System.out.println(isim+" Çalışıyor...");
17         for (int i = 0; i <= 10; i++) {
18             try {
19                 System.out.println(isim+" Yazıyor: "+i);
20                 Thread.sleep(1000);
21             } catch (InterruptedException ex) {
22                 System.out.println(" Thread kesintiye uğradı");
23             }
24         }
25         System.out.println(isim+" işini bitirdi");
26     }
27
28 }
29
```

- Printer sınıfını Thread extends ettik. . Yani Thread sınıfını extend eden bir sınıf oluştururuz ve run metodunu override ederiz.Böylece thread tarafından çalıştırılacak kodu belirtebiliriz.

Thread.sleep=Her çalışmadan öcne 1 saniye beklemesini söylüyoruz.

The screenshot shows an IDE interface with two tabs: 'Main.java X' and 'Printer.java X'. The 'Main.java' tab is active, displaying the following code:

```
1 package com.harunaydemir.thread1;
2
3
4
5 public class Main {
6     public static void main(String[] args) {
7         Printer printer1 = new Printer("Printer1 ");
8         Printer printer2 = new Printer("Printer2 ");
9
10        printer1.start();
11        printer2.start();
12
13        System.out.println("Main thread çalışıyor");
14    }
15
16 }
```

The 'Printer.java' tab shows the definition of the Printer class:

```
1 package com.harunaydemir.thread1;
2
3
4
5 public class Printer {
6     public void start() {
7         for (int i = 1; i <= 10; i++) {
8             System.out.println(Thread.currentThread().getName() + " Yazıyor: " + i);
9         }
10    }
11 }
```

In the bottom left corner, there is a terminal window titled 'out - Run (Main)' showing the execution results:

```
Printer2 Yazıyor: 5
Printer1 Yazıyor: 6
Printer2 Yazıyor: 6
Printer1 Yazıyor: 7
Printer2 Yazıyor: 7
Printer1 Yazıyor: 8
Printer2 Yazıyor: 8
Printer1 Yazıyor: 9
Printer2 Yazıyor: 9
Printer1 Yazıyor: 10
Printer2 Yazıyor: 10
Printer1 igini bitirdi
Printer2 igini bitirdi
```

Main methodunda bu şekilde start diyerek threadleri çalıştırabiliriz. Ancak çalışma sıraları karışık olacağından belli bir sırayı takip etmez. Önce 2 veya 1 çalışabilir. Çıktı da görüldüğü gibi.

2. Yöntem ise şu şekilde;

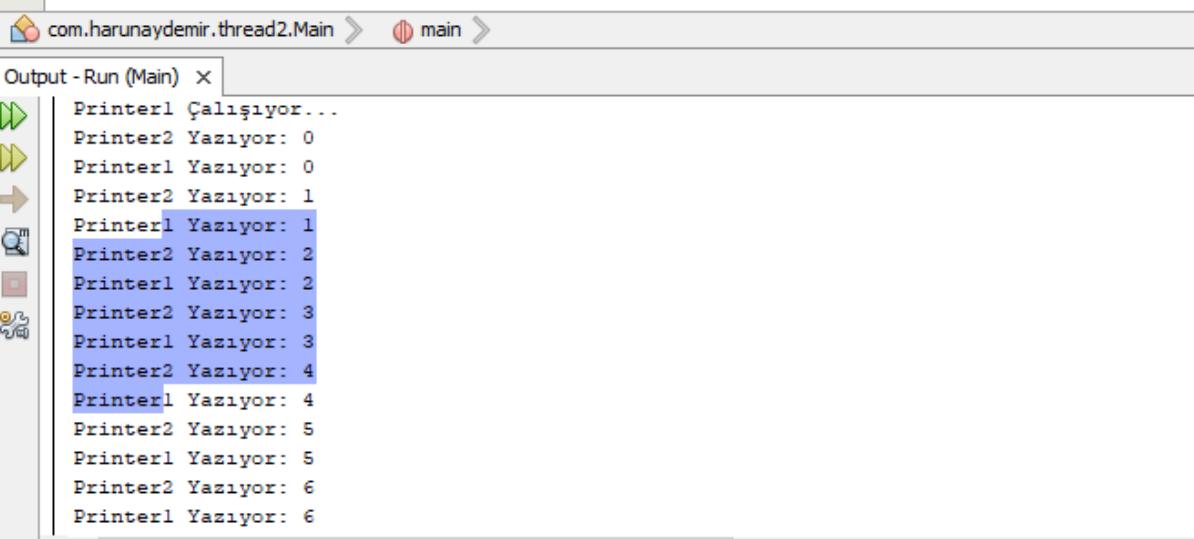
The screenshot shows a Java IDE interface with two tabs open: Main.java and Printer.java. The Printer.java tab is active, displaying the following code:

```
7  /**
8  *
9  * @author HarunAydemir
10 */
11 public class Printer implements Runnable{
12
13     private String isim;
14
15     public Printer(String isim) {
16         this.isim = isim;
17     }
18
19     @Override
20     public void run() {
21         System.out.println(isim+" Çalışıyor...");
22         for (int i = 0; i <= 10; i++) {
23             try {
24                 System.out.println(isim+" Yazıyor: "+i);
25                 Thread.sleep(1000);
26             } catch (InterruptedException ex) {
27                 System.out.println(" Thread kesintiye uğradı");
28             }
29         }
30         System.out.println(isim+" işini bitirdi");
31     }
32 }
```

- Runnable interface'ni implemente etmektir. Burada run methodunu override etmek zorundayız.

```
2 package com.harunaydemir.thread2;
3
4 public class Main {
5     public static void main(String[] args) {
6         Thread printer1 = new Thread(new Printer("Printer1"));
7         Thread printer2 = new Thread(new Printer("Printer2"));
8
9         printer1.start();
10        printer2.start();|
```

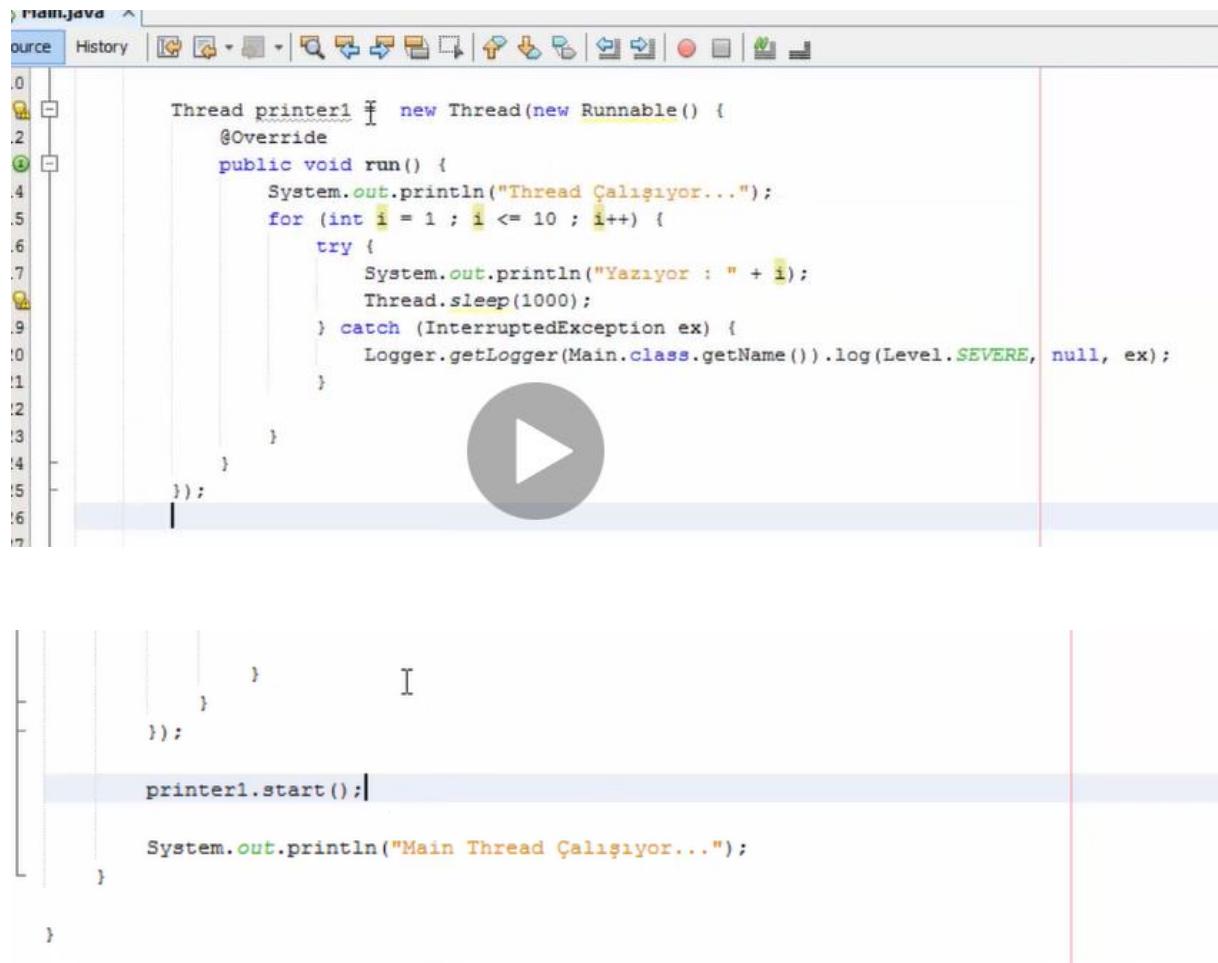
```
11
12         System.out.println("Main thread çalışıyor...");
13     }
14 }
```



```
Printer1 Çalışıyor...
Printer2 Yazıyor: 0
Printer1 Yazıyor: 0
Printer2 Yazıyor: 1
Printer1 Yazıyor: 1
Printer2 Yazıyor: 2
Printer1 Yazıyor: 2
Printer2 Yazıyor: 3
Printer1 Yazıyor: 3
Printer2 Yazıyor: 3
Printer1 Yazıyor: 4
Printer2 Yazıyor: 4
Printer1 Yazıyor: 4
Printer2 Yazıyor: 5
Printer1 Yaziyor: 5
Printer2 Yaziyor: 6
Printer1 Yaziyor: 6
```

- Main içinde ise görüldüğü gibi Thread diyerek newlememiz gerekiyor.

- Anonim sınıf olarakta yani başka bir sınıfı kullanmadan direkt thread oluşturabiliriz.



```
Main.java ^  
Source History |   
.0     Thread printer1 = new Thread(new Runnable() {  
.1         @Override  
.2         public void run() {  
.3             System.out.println("Thread Çalışıyor...");  
.4             for (int i = 1 ; i <= 10 ; i++) {  
.5                 try {  
.6                     System.out.println("Yazıyor : " + i);  
.7                     Thread.sleep(1000);  
.8                 } catch (InterruptedException ex) {  
.9                     Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
.10                }  
.11            }  
.12        }  
.13    }));  
.14  
.15    printer1.start();  
.16  
.17    System.out.println("Main Thread Çalışıyor...");  
}
```

```
run:  
Main Thread Çalışıyor...  
Thread Çalışıyor...  
Yazıyor : 1  
Yazıyor : 2  
Yazıyor : 3  
Yazıyor : 4  
Yazıyor : 5  
Yazıyor : 6  
Yazıyor : 7  
Yazıyor : 8  
Yazıyor : 9  
Yazıyor : 10  
BUILD SUCCESSFUL (total time: 10 seconds)
```

- Görüldüğü gibi direkt main içerisinde başka bir class oluşturmadan thread kulandık. Printer 1 diye isim de vermemize gere yok.

```
});/*/  
new Thread(new Runnable() {  
    @Override  
    public void run() {  
        System.out.println("Thread Çalışıyor...");  
        for (int i = 1 ; i <= 10 ; i++) {  
            try {  
                System.out.println("Yazıyor : " + i);  
                Thread.sleep(1000);  
            } catch (InterruptedException ex) {  
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
            }  
        }  
    }  
}).start();
```



- Ancak başka yerde kullanamıyoruz isim vermediğimiz için. Tek bir defa tanımlandığı yerde kullanıyoruz.

169. Threadlerin Senkronizasyonu, Synchronized Kullanımı ve Join metodu

The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer with a single package named 'com.mycompany.synchronizedanahtarkelimesi' containing a file 'ThreadSafe.java'. The code in 'ThreadSafe.java' is as follows:

```
1  package com.mycompany.synchronizedanahtarkelimesi;
2
3  public class ThreadSafe{
4      private int count =0;
5
6      public void threadleriCalistir(){
7          Thread thread1= new Thread(new Runnable() {
8              @Override
9              public void run() {
10                  for (int i = 0; i < 5000; i++) {
11                      count++;
12                  }
13              }
14          });
15          Thread thread2= new Thread(new Runnable() {
16              @Override
17              public void run() {
18                  for (int i = 0; i < 5000; i++) {
19                      count++;
20                  }
21              }
22          });
23          thread1.start();
24          thread2.start(); //Bu 2 thread ait olmayan diğer tüm işemler main threadinde
25          System.out.println("Count değişkeninin değeri: "+count);
26
27      public static void main(String[] args) {
28          ThreadSafe threadsafe= new ThreadSafe();
29
30          threadsafe.threadleriCalistir();
31
32      }
33  }
```

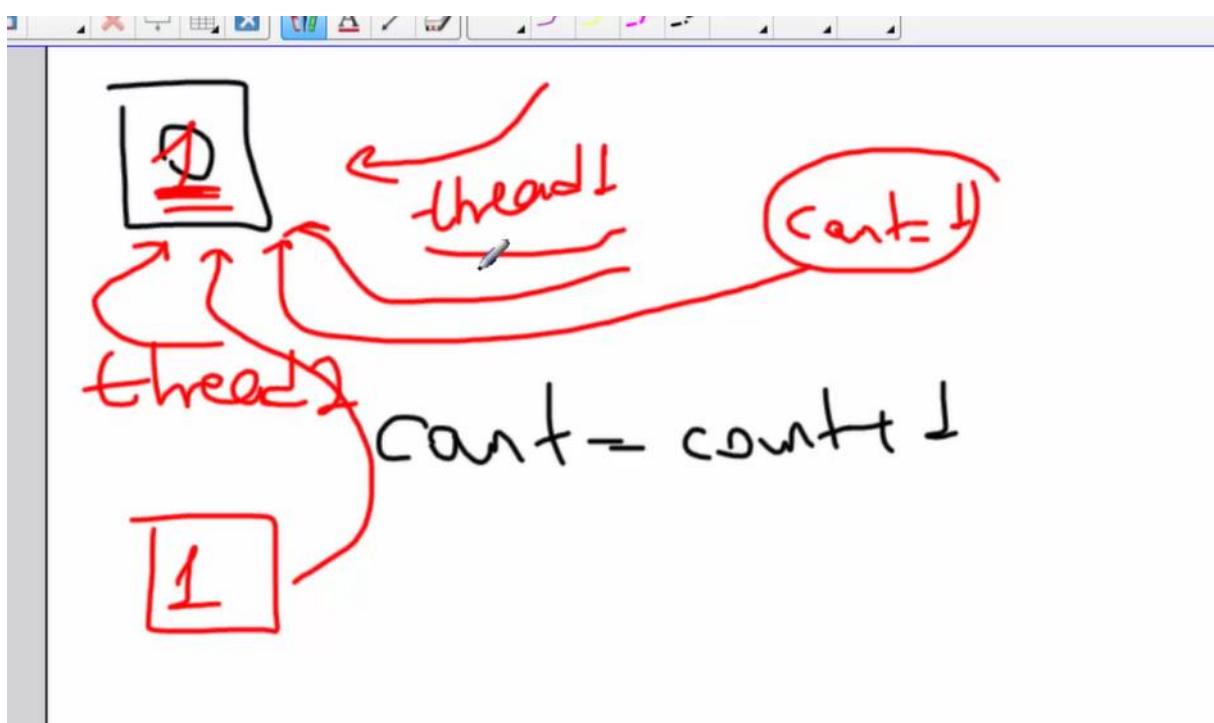
Below the code editor is the build output window:

```
|- Run (ThreadSafe)
-----< com.mycompany:SynchronizedAnahtarKelimesi >-----
Building SynchronizedAnahtarKelimesi 1.0-SNAPSHOT
-----[ jar ]-----
--- exec-maven-plugin:3.0.0:exec (default-cli) @ SynchronizedAnahtarKelimesi ---
Count değişkeninin değeri: 0
-----
BUILD SUCCESS
-----
Total time: 1.177 s
Finished at: 2021-11-13T21:59:22+03:00
|
```

- Üstte bir sınıf oluşturup sayıç tanımladık ve tanımla 2 tane threadin bu sayıçı 5000e kadar artırması gerekiyor. Ancak çalıştırduğumızda sonuç 0 çıkıyor.(Oluşturulan threadler dışında geri kalan herşey main metyhoduna aittir.) Çünkü önce main methodunun threadı çalışıyor her zaman ve main methodu olduğu için diğerlerini çalıştırmadan programı sonlandırıyor. Önce thread1 ve thread2 nin sonradan main methodunun çalışmasını isteyebiliriz. O yüzden join methodunu kullanmalıyız.
- Thread de **join** methodu, bir thread işlemini bitirmeden diğer bir thread çalışması engellenir. Eşzamanlı birden fazla thread in iş yapması önlenmiş olur.

```
1         count++;
2     }
3 }
4 );
5     thread1.start();
6     thread2.start(); //Bu 2 thread ait olmayan diğer tüm işemler main threadine ait oluyor.
7 try {
8     thread1.join();
9 } catch (InterruptedException ex) {
0     Logger.getLogger(ThreadSafe.class.getName()).log(Level.SEVERE, null, ex);
1 }
2 try {
3     thread2.join();
4 } catch (InterruptedException ex) {
5     Logger.getLogger(ThreadSafe.class.getName()).log(Level.SEVERE, null, ex);
6 }
7     System.out.println("Count değişkeninin değeri: "+count);
8 }
9
0     public static void main(String[] args) {
1         ThreadSafe threadsafe= new ThreadSafe();
2
3         threadsafe.threadleriCalistir();
4
5     }
6
7 }
```

- Bu şekilde yaptığımızda main methodu önce thread1 ve thread2 nin çalışmasını bekliyor. Çalıştırıldığında countun 10.000 olması gereklidir. Ancak her çalışmada farklı sayılar çıkıyor. (Eski java sürümünde öyle oluyor. Yeni sürümde sekronlu çalışıyorlar.) Bunun sebebi 2 threadinde aynı değişkene erişip etkilemesi. Mesela count 0 iken thread1 1 değerini ekliyor ancak aynı anda thread 2 de count değerini ekliyor. Bu durumda count 2 yerine 1 olmuş oluyor. Bu her zaman olmamalıdır. Burada bir threadin işi bitene kadar diğerinin erişimini engellememiz gerekiyor. Bunun için durumu düzeltmek için synchronized kullanmalıyız.
 - **Synchronized:** Metoda ulaşmak isteyen Threadler metoda sıra ile girerler ve bir thread metodu bitirmeden diğer thread giremez.



```
5
6     public class ThreadSafe{
7         private int count =0;
8         public synchronized void arttir(){
9             count++;
10        }
11        public void threadleriCalistir(){
12            Thread thread1= new Thread(new Runnable() {
13                @Override
14                public void run() {
15                    for (int i = 0; i < 5000; i++) {
16                        arttir();
17                    }
18                }
19            });
20            Thread thread2= new Thread(new Runnable() {
21                @Override
22                public void run() {
23                    for (int i = 0; i < 5000; i++) {
24                        arttir();
25                    }
26                }
27            });
28            thread1.start();
29            thread2.start(); //Bu 2 thread ait olmayan diğer tüm işemler main threadine ait oluyor.
30            try {
31                thread1.join();
32            } catch (InterruptedException ex) {
33                Logger.getLogger(ThreadSafe.class.getName()).log(Level.SEVERE, null, ex);
34            }
35            try {
36                thread2.join();
37            } catch (InterruptedException ex) {
38                Logger.getLogger(ThreadSafe.class.getName()).log(Level.SEVERE, null, ex);
39            }
40        }
41    }
42 }
```

- Görüldüğü gibi count ++ yi arttır methodu ile tanımladık ve ona synchronized anahtar kelimesini tanımladık.

170. Birden Çok Lock Kullanımı ve Synchronized Metod Sıkıntısı

Thread kullanmadan sadece main thread ile program yazdık.

```
8 public class ListWorker {
9     Random random= new Random();
10    ArrayList<Integer> list1=new ArrayList<Integer>();
11    ArrayList<Integer> list2=new ArrayList<Integer>();
12
13    public void list1DegerEkle(){
14        for (int i = 0; i < 2000; i++) {
15            list1.add(i);
16            try {
17                Thread.sleep(1);
18            } catch (InterruptedException ex) {
19                Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
20            }
21        }
22    }
23    public void list2DegerEkle(){
24        for (int i = 0; i < 2000; i++) {
25            list2.add(i);
26            try {
27                Thread.sleep(1);
28            } catch (InterruptedException ex) {
29                Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
30            }
31        }
32    }
33    public void degerAta(){
34        list1DegerEkle();
35        list2DegerEkle();
36        System.out.println("List1: "+list1.size()+" List2: "+list2.size());
37    }
38}
```

- Bu classta list1 ve list2 ye 2000 tane değer atadık. Ekrana yazdırıldı.

```
1
2 public class Main {
3     public static void main(String[] args) {
4         ListWorker worker=new ListWorker();
5         long baslangic=System.currentTimeMillis();
6
7         worker.degerAta();
8         long bitis=System.currentTimeMillis();
9         System.out.println("Geçen süre: "+(bitis-baslangic));
10    }
11
12
13
14}
```

- Sadece main thread kullanarak 2000 tane değer atma süresini hesapladık.

```

-----< com.mycompany:Birdencoklockkullanmak >-----
[INFO] Building Birdencoklockkullanmak 1.0-SNAPSHOT
-----[ jar ]-----

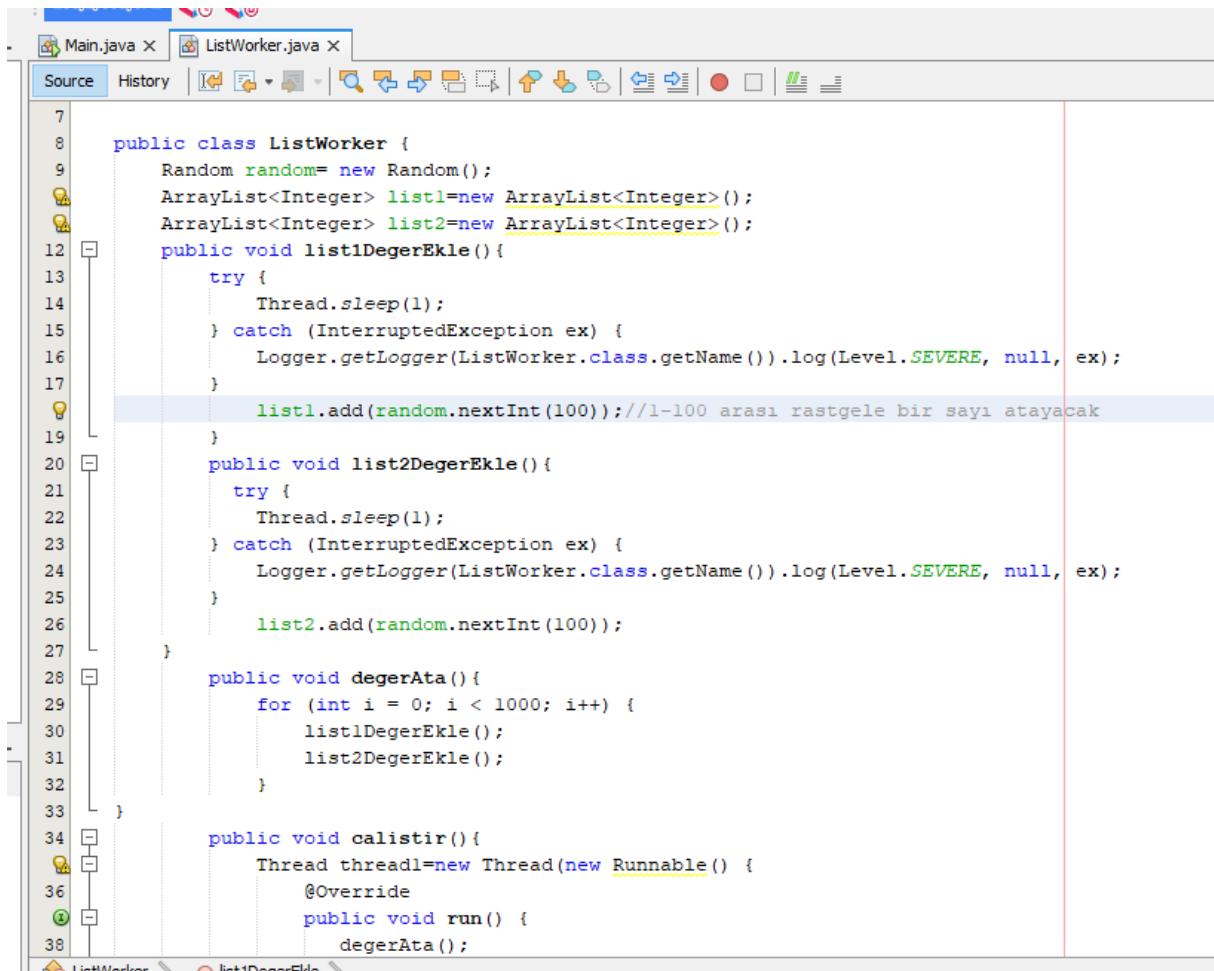
[INFO] --- exec-maven-plugin:3.0.0:exec (default-cli) @ Birdencoklockkullanmak ---
List1: 2000 List2: 2000
Geçen süre: 7745

BUILD SUCCESS

Total time: 8.559 s
Finished at: 2021-11-14T12:22:31+03:00
-----
```

- Çıktıda görüldüğü gibi değer atama süreleri 7745 milisaniye sürdü. Tek bir thread(main) kullanarak bu süreyi elde ettik.(2000 milisaniye Thread.sleep(1); kodundan geldi).

Eğer thread kullanarak yaparsak;



```

7
8     public class ListWorker {
9         Random random= new Random();
10        ArrayList<Integer> list1=new ArrayList<Integer>();
11        ArrayList<Integer> list2=new ArrayList<Integer>();
12        public void list1DegerEkle(){
13            try {
14                Thread.sleep(1);
15            } catch (InterruptedException ex) {
16                Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
17            }
18            list1.add(random.nextInt(100)); //1-100 arası rastgele bir sayı atayacak
19        }
20        public void list2DegerEkle(){
21            try {
22                Thread.sleep(1);
23            } catch (InterruptedException ex) {
24                Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
25            }
26            list2.add(random.nextInt(100));
27        }
28        public void degerAta(){
29            for (int i = 0; i < 1000; i++) {
30                list1DegerEkle();
31                list2DegerEkle();
32            }
33        }
34        public void calistir(){
35            Thread thread=new Thread(new Runnable() {
36                @Override
37                public void run() {
38                    degerAta();
```

The screenshot shows an IDE interface with two tabs at the top: 'Main.java X' and 'ListWorker.java X'. The 'Source' tab is selected for the Main.java file. The code in Main.java is as follows:

```
public void run() {
    degerAta();
}
});

Thread thread2=new Thread(new Runnable() {
    @Override
    public void run() {
        degerAta();
    }
});
long baslangic=System.currentTimeMillis();
thread1.start();
thread2.start();

try {
    thread1.join();
} catch (InterruptedException ex) {
    Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
}
try {
    thread2.join();
} catch (InterruptedException ex) {
    Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
}
System.out.println("List1 Size: "+list1.size()+" List2 Size: " +list2.size());
long bitis=System.currentTimeMillis();
System.out.println("Geçen süre: "+(bitis-baslangic));
}
```

- ListWorker classımız bu şekilde yapıyoruz.

The screenshot shows an IDE interface with two main panes. The top pane displays the code for Main.java:

```
1  public class Main {  
2      public static void main(String[] args) {  
3          ListWorker worker=new ListWorker();  
4          worker.calistir();  
5          /* long baslangic=System.currentTimeMillis();  
6  
7              worker.degerAta();  
8              long bitis=System.currentTimeMillis();  
9              System.out.println("Geçen süre: "+(bitis-baslangic));*/  
10         }  
11     }  
12  
13 }  
14  
15 }
```

The bottom pane shows the Maven build output for the 'main' profile:

```
Main > main >  
Output - Run (Main) x  
-----  
-----< com.mycompany:Birdencoklockkullanmak >-----  
Building Birdencoklockkullanmak 1.0-SNAPSHOT  
-----[ jar ]-----  
-----  
exec-maven-plugin:3.0.0:exec (default-cli) @ Birdencoklockkullanmak ---  
List1 Size: 1996 List2 Size: 1994  
Geçen süre: 3905  
-----  
BUILD SUCCESS  
-----  
Total time: 5.186 s  
Finished at: 2021-11-14T12:40:21+03:00
```

- Main içinde çalıştırduğumızda ise sürenin yarı yarıya düşüğünü görüyoruz ancak list1 ve list2 nin değer atamaları tam olmamış. Bunun sebebi önceki derste anlatıldığı gibi. Aynı yere aynı anda değer koydukları için sekronizasyon hatası oluşturuluyor.

```
public synchronized void list1DegerEkle() {
    try {
        Thread.sleep(1);
    } catch (InterruptedException ex) {
        Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
    }
    list1.add(random.nextInt(100)); //1-100 arası rastgele bir sayı atayacak
}
public synchronized void list2DegerEkle() {
    try {
        Thread.sleep(1);
    } catch (InterruptedException ex) {
        Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
    }
    list2.add(random.nextInt(100));
}
public void degerAta() {
    for (int i = 0; i < 1000; i++) {
        list1DegerEkle();
    }
}
```

- Threadlere synchronized yazdığımızda sorunu düzeltmiş oluyoruz.
- **Synchronized voidden önce yazılmalıdır.**

The screenshot shows the NetBeans IDE interface. At the top, there are tabs for 'Main.java' and 'ListWorker.java'. Below the tabs is a toolbar with various icons. The main editor area contains the following Java code:

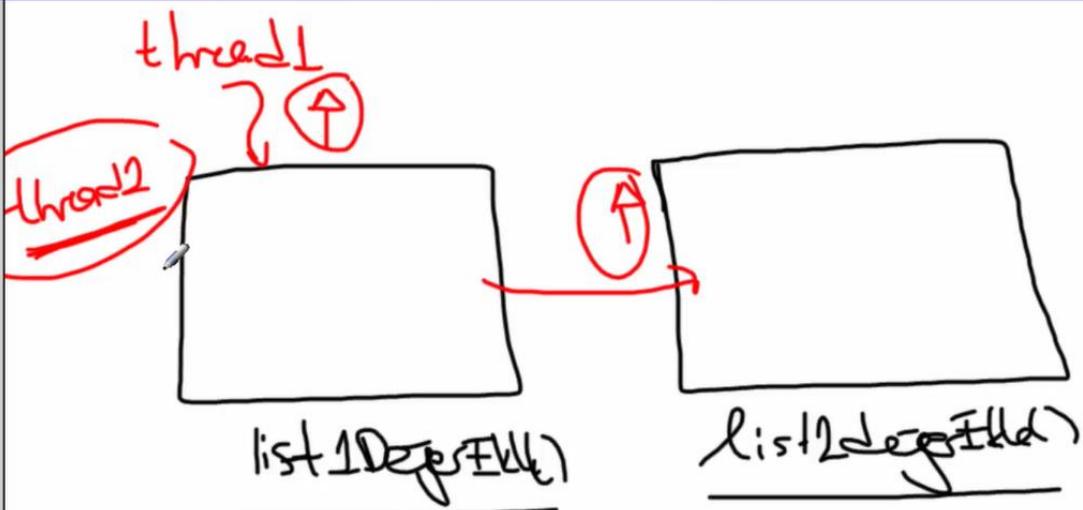
```
1 public class Main {  
2     public static void main(String[] args) {  
3         ListWorker worker=new ListWorker();  
4         worker.calistir();  
5         /* long baslangic=System.currentTimeMillis();  
6  
7             worker.degerAta();  
8             long bitis=System.currentTimeMillis();  
9             System.out.println("Geçen süre: "+(bitis-baslangic));*/  
10    }  
11 }  
12  
13  
14 }  
15
```

On the right side of the editor, there is a vertical red margin line. Below the editor is a small toolbar with icons for copy, cut, paste, and others.

At the bottom of the interface is the 'Output - Run (Main)' window, which displays the build logs:

```
cd C:\Users\HarunAydemir\Documents\NetBeansProjects\Birdencoklockkullanmak; "JAVA_HOME=C:\\Program Files\\Java\\jdk-1  
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects  
Scanning for projects...  
  
-----< com.mycompany:Birdencoklockkullanmak >-----  
Building Birdencoklockkullanmak 1.0-SNAPSHOT  
-----[ jar ]-----  
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Birdencoklockkullanmak ---  
List1 Size: 2000 List2 Size: 2000  
Geçen süre: 7947  
-----  
BUILD SUCCESS  
-----
```

- Maini çalıştırıldığımızda görüyoruz ki size sorunu düzeltmiş ancak süre tekrar 7000 milisaniyeye çıktı. Burada bir hata var.



- Her classta bir anahtar var. Thread1 o anahtarı list1dEgerEkle scyhronized dediğimizde oraya giriyor ve işlemini yapıyor. Thread2 de sırada bekliyor. Ancak thread1 işlemi bitirdiğinde anahtar hala onda ve list2DegerEkle giriyor. Bu esnada thread2 list1DegerEkle girip işlemini yapamıyor çünkü anahtar thread1 de. Thread2 o yüzden thread1in list2DegerEkle'de işini bitirmesini bekliyor ve anahtar boşça çıkışa alıp tekrar list1DegerEkle ve list2DegerEkleye giriyor. Bunun sonucunda thread2 2 kere beklemiş oluyor. Çünkü biz thread1in list1degerEkle'de işi bittiğinde thread2nin girmesini söylemedik. Bunun çözmek için 2 tane anahtara sahip olmalıyız. Bunun için metodlara scyhronized yazmayacağız.

```

public class ListWorker {
    Random random= new Random();
    ArrayList<Integer> list1=new ArrayList<Integer>();
    ArrayList<Integer> list2=new ArrayList<Integer>();
    private Object lock1= new Object();
    private Object lock2= new Object();
    public void list1DegerEkle(){
        synchronized(lock1){
            try {
                Thread.sleep(1);
            } catch (InterruptedException ex) {
                Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
            }
            list1.add(random.nextInt(100));//1-100 arası rastgele bir sayı atayacak
        }
    }
    public void list2DegerEkle(){
        synchronized(lock2){
            try {
                Thread.sleep(1);
            } catch (InterruptedException ex) {
                Logger.getLogger(ListWorker.class.getName()).log(Level.SEVERE, null, ex);
            }
            list2.add(random.nextInt(100));
        }
    }
}

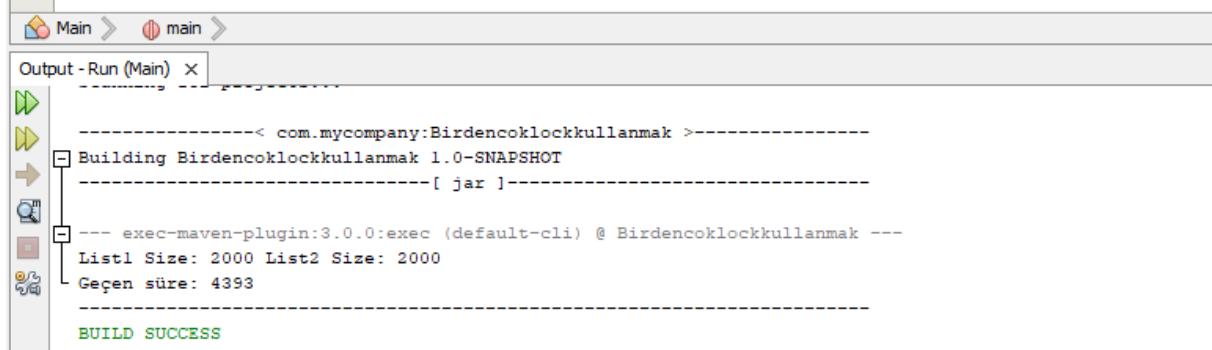
```

- Lockları(anahtarları) kendimiz tanımladık. Burada sınıfın anahtarını kullanmadık ve metodlara anahtar oluşturduk. Thread1 lock1 ile işi bittiğinde lock1 boşça çıktıgı için Thread2 onu alıp list1DegerEkle'de işlem yapabilecek.

```

1  public class Main {
2      public static void main(String[] args) {
3          ListWorker worker=new ListWorker();
4          worker.calistir();
5          /* long baslangic=System.currentTimeMillis();
6
7          worker.degerAta();
8          long bitis=System.currentTimeMillis();
9          System.out.println("Geçen süre: "+(bitis-baslangic));*/
10     }
11 }
12
13
14 }
15

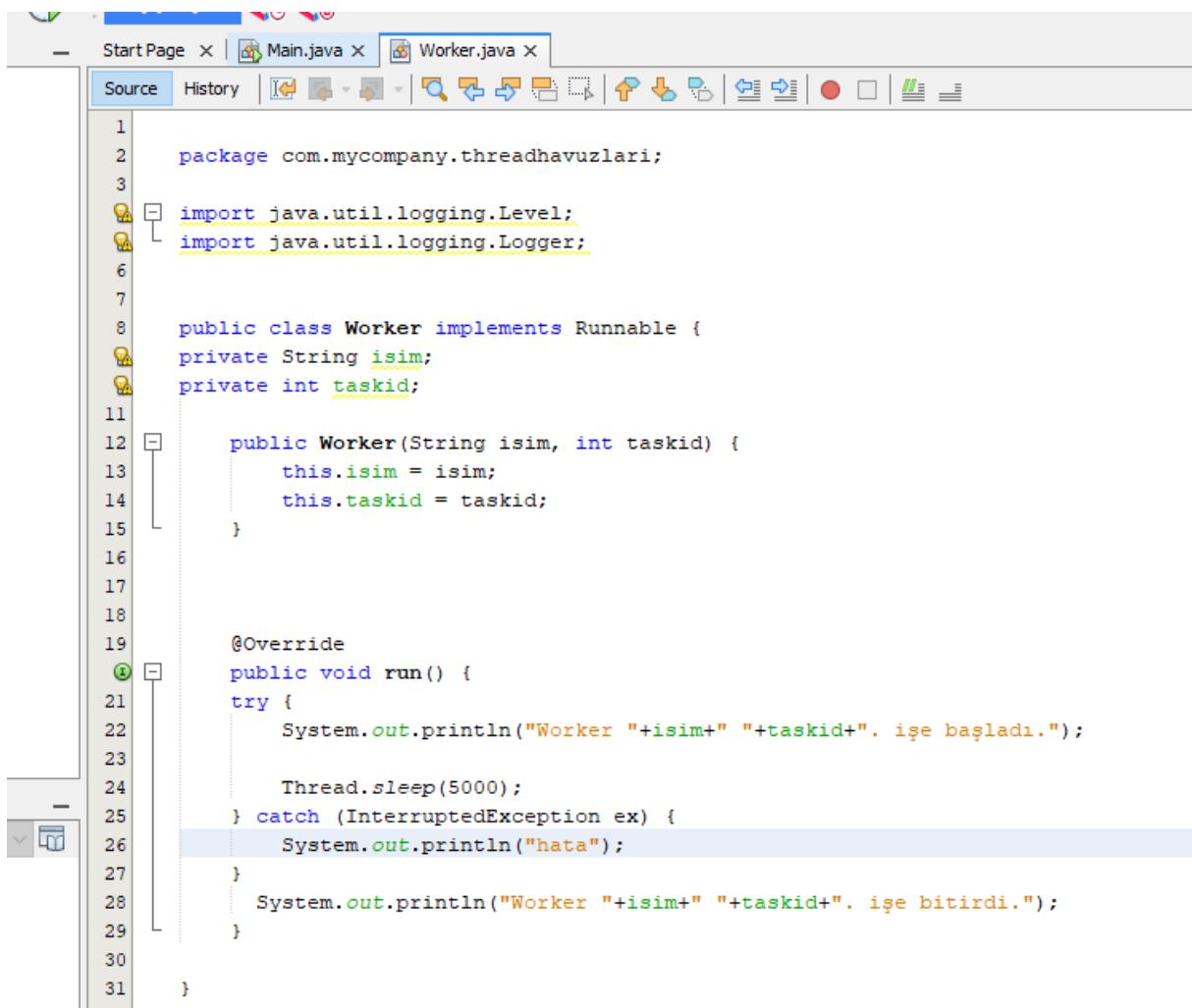
```



- Çalıştırdığımızda sürenin azaldığını ve sizein doğru değerini görmüş olduk.

171. Thread Havuzları ve ExecutorService Kullanımı

- Büyük projelerde yüzlerce thread olabilir ve programcı bunların düzenini sağlamak angarya gelebilir. O yüzden sadece ilk şu 10 thread çalışın sonra başka 10 thread çalışın gibi şeyler söyleyebiliyoruz.



The screenshot shows an IDE interface with three tabs at the top: "Start Page", "Main.java", and "Worker.java". The "Worker.java" tab is active, displaying the following Java code:

```
1 package com.mycompany.threadhavuzlari;
2
3 import java.util.logging.Level;
4 import java.util.logging.Logger;
5
6
7 public class Worker implements Runnable {
8     private String isim;
9     private int taskid;
10
11    public Worker(String isim, int taskid) {
12        this.isim = isim;
13        this.taskid = taskid;
14    }
15
16
17
18
19    @Override
20    public void run() {
21        try {
22            System.out.println("Worker "+isim+" "+taskid+". işe başladı.");
23
24            Thread.sleep(5000);
25        } catch (InterruptedException ex) {
26            System.out.println("hata");
27        }
28        System.out.println("Worker "+isim+" "+taskid+". işe bitirdi.");
29    }
30
31 }
```

- Worker sınıfını bu şekilde oluşturduk.

The screenshot shows an IDE interface with a code editor containing Java code. The code is designed to handle five threads and log errors if any thread fails to complete its task.

```
7 public class Main {  
8     public static void main(String[] args) {  
9         Thread t1=new Thread(new Worker("1",1));  
10        Thread t2=new Thread(new Worker("2",2));  
11        Thread t3=new Thread(new Worker("3",3));  
12        Thread t4=new Thread(new Worker("4",4));  
13        Thread t5=new Thread(new Worker("5",5));  
14        System.out.println("Bütün işler teslim edildi.");  
15        t1.start();  
16        t2.start();  
17        try {  
18            t1.join();  
19            t2.join();  
20        } catch (InterruptedException ex) {  
21            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
22        }  
23        t3.start();  
24        t4.start();  
25        try {  
26            t3.join();  
27            t4.join();  
28        } catch (InterruptedException ex) {  
29            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
30        }  
31        t5.start();  
32        try {  
33            t5.join();  
34        } catch (InterruptedException ex) {  
35            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
36        }  
37        System.out.println("İşlem tamamlandı");  
38    }  
39 }  
com.mvcompany.threadhavuzlari.Main >  
21     } catch (InterruptedException ex) {  
22         Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
23     }  
24     t3.start();  
25     t4.start();  
26     try {  
27         t3.join();  
28         t4.join();  
29     } catch (InterruptedException ex) {  
30         Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
31     }  
32     t5.start();  
33     try {  
34         t5.join();  
35     } catch (InterruptedException ex) {  
36         Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
37     }  
38     System.out.println("İşlem tamamlandı");  
39 }  
40 }  
41 }
```

- Main sınıfımıza bu şekilde oluşturduk.

```
32
com.mycompany.threadhavuzlari.Worker > run > try > catch InterruptedException ex >
Output - Run (Main) X
--- exec-maven-plugin:3.0.0:exec (default-cli) @ ThreadHavuzlari ---
Bütün işler teslim edildi.
Worker 1 1. işe başladı.
Worker 2 2. işe başladı.
Worker 1 1. işe bitirdi.
Worker 2 2. işe bitirdi.
Worker 3 3. işe başladı.
Worker 4 4. işe başladı.
Worker 3 3. işe bitirdi.
Worker 4 4. işe bitirdi.
Worker 5 5. işe başladı.
Worker 5 5. işe bitirdi.
İşlem tamamlandı
```

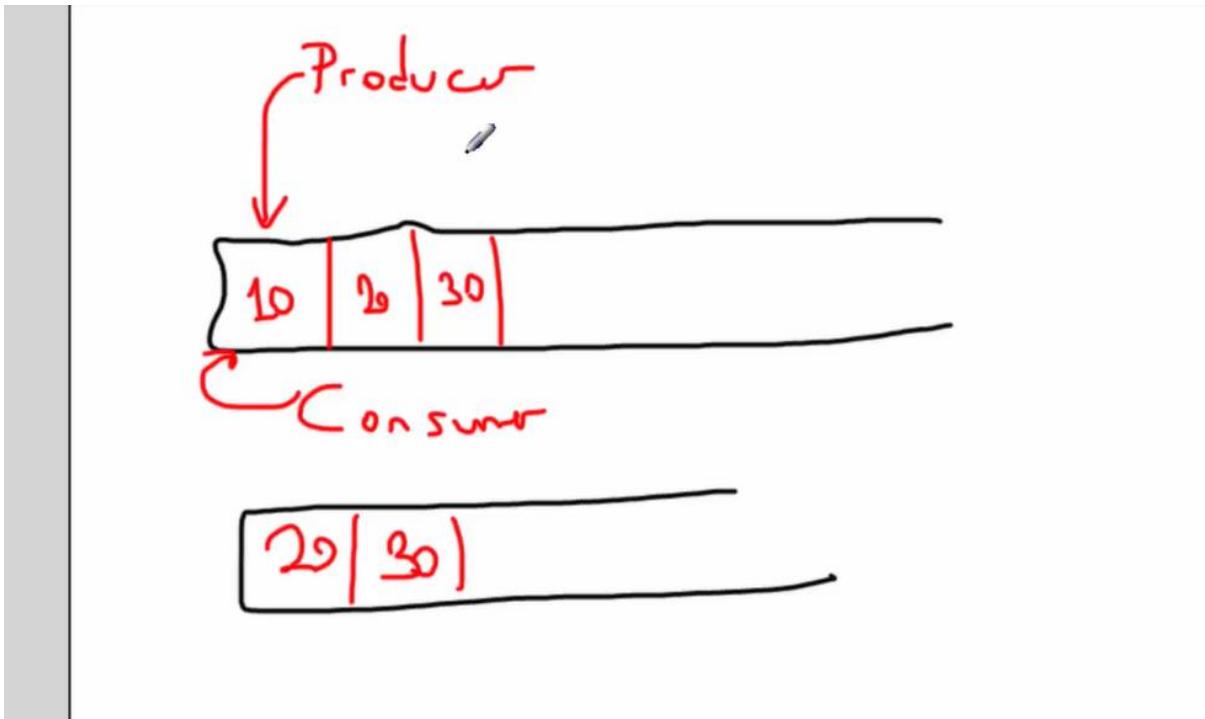
- Çıktısı ise bu şekilde oldu ve işlem uzun sürdü. Çünkü önce 1 ve 2 sonra 3 ve 4 sonra 5. Threadi bekledik. Bunu çözmek için thread havuzları ve ExecutorService kullanıcaz.
- **ExecuteService**: ExecutorService, asenkron işler çalıştırımızı kolaylaştıran JDK tarafından sunulan bir interface'dir. Bu interface aracı ile, thread seviyesinde ele almamız gereken işler ile (Thread oluşturma, hata sonucu sonlanan threadi tekrar çalışma vb) ilgilenmeden Thread havuzu oluştururuz.
- ExecutorService örneği oluşturmanın iki farklı yöntemi vardır. Bunlardan ilki Executors sınıfında bulunan static factory metodlardır. İkincisi ise direkt ThreadPoolExecutor sınıfının yapılandırıcısını kullanmaktadır. Executors sınıfının metodlarına bakarsak, ThreadPoolExecutor yapılandırıcısının kullanıldığını görürürüz. ThreadPoolExecutor yapılandırıcısını kullanmak bize daha fazla esneklik sağlar.

```
public class Main {  
    public static void main(String[] args) {  
  
        ExecutorService executor=Executors.newFixedThreadPool(2);  
        //en fazla 2 tane thread çalışacağımızı söylediğimiz (2) ile  
        /*Dahilinde azami 10 thread barındıran bir thread havuzu ve  
        thread havuzunu besleyen, yani çalıştırılması için gönderilen  
        işleri tutan üst limiti olmayan bir queue oluşturulur. Static  
        metotun koduna bakar isek, ThreadPoolExecutor yapılandırıcısını görürüz.*/  
        for(int i=1;i<=5;i++){  
            executor.submit(new Worker(String.valueOf(i),i));  
            //5 tane task atamadı yaptık.  
        }  
        executor.shutdown(); //Başka hiçbir iş yapmıyorum.Bu threadler bitene kadar  
        //ben executeri kaptırıyorum.  
        //Threadlerin işi bittiği zaman bu threadleri kapatıcam.  
        //Eğer bunu yazmazsak sonsuz döngüye girer.  
        System.out.println("Bütün işler teslim edildi.");  
        try {  
  
            executor.awaitTermination(1, TimeUnit.DAYS); //Burada şunu demek istiyoruz.  
            //Ben bu kodu en fazla 1 gün beklerim.1 gün sonra bitsin.  
            //Güvenlik açısından önemlidir.  
  
        } catch (InterruptedException ex) {  
            System.out.println("Bütün işler bitti.");  
        }  
    }  
}
```

```
--- exec-maven-plugin:3.0.0:exec (d  
Bütün işler teslim edildi.  
Worker 1 1. işe başladı.  
Worker 2 2. işe başladı.  
Worker 1 1. işe bitirdi.  
Worker 2 2. işe bitirdi.  
Worker 3 3. işe başladı.  
Worker 4 4. işe başladı.  
Worker 4 4. işe bitirdi.  
Worker 3 3. işe bitirdi.  
Worker 5 5. işe başladı.  
Worker 5 5. işe bitirdi.  
-----
```

- Çıktısı bu şekildedir.

172. Mini Proje - ArrayBlockingQueue Kullanarak Producer Consumer Problemi



- Producer threadi rastgele değer üretecek. Consumer da o değerleri rastgele alacak. 10 20 30 yazan yer q olacak ve sınırlı sayıda eleman taşıyabilecek.
- **ArrayBlockingQueue** :Merhaba , Bu yazımızda multithreading konusunu işlerken karşımıza gelen ve Thread Safe olan bir yapıdan bahsedeceğim. " ArrayBlockingQueue" bildiğimiz gibi Producer , Consumer sisteminde biz threadleri bazı kontrol mekanizmaları ile kontrol ediyorduk Reentrantlock yada wait , notify gibi. Eğer böyle yapmazsa threadlerin çalışması sırasında problemlerle karşılaşarıyorduk. Biraz açarsak ; Producer üretmeden Consumer thread'i tüketmeye kalkarsa null değer alacak ve bu da hataya sebep olacaktır. İşin özü bizim iyi bir senkronizasyon yapmamız gerekiyordu. Ancak thread safe nesnelerde ise böyle bir problem olmuyor . peki nasıl oluyor? . bakalım; örneğin: Bir sıramatik sistemimiz var. Bu sistem nasıl çalışır ? Öncelikle birisi gelir sıra alır , alınan sıra kuyruğa eklenir (bu bir Produce işlemidir) , banko görevlisi ise çağrı butonuna basar ve kuyruktan bir sıra çeker. Şimdi burada bir producer , consumer sistemi çalışıyor . Thread safe nesne kullandığımızda , Consumer butona bassa bile eğer bir sıra yoksa bu thread bekletilir. Ne zamana kadar? producer bir sıra üretene kadar. İşte Threadlerle çalışırken eğer ArrayBlockingQueue kullanırsak threadlerin kontrolünü kendimiz yapmaya mecbur olmayız thread safe olduğu için o kendi bu işi halledecektir.
- **Put:** kuyruğun sonuna eleman ekler.
- **Take:** kuyruğun başından eleman alır.
- ArrayBlockingQueue kendi içinde thread kullanır.

```
public class Main {
    public static void main(String[] args) {
        ProducerConsumer pc = new ProducerConsumer();

        Thread producer = new Thread(new Runnable() {
            @Override
            public void run() {
                pc.produce();
            }
        });
        Thread consumer = new Thread(new Runnable() {
            @Override
            public void run() {
                pc.consume();
            }
        });

        producer.start();
        consumer.start();

        try {
            producer.join();
            consumer.join();
        } catch (InterruptedException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

268,9/392,0MB

Main.java X ProducerConsumer.java X

Source History

```
11 class ProducerConsumer {
12     Random random=new Random();
13     BlockingQueue<Integer> queue= new ArrayBlockingQueue<Integer>(10);
14     //En fazla 10 eleman alabilecek ArrayBlockingQueue oluşturduk.
15 }
16
17 public void produce() {
18     while(true) {
19         try {
20             Thread.sleep(1000); //1 saniye bekledik.
21         } catch (InterruptedException ex) {
22             Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
23         }
24         Integer value=random.nextInt(100);
25         queue.put(value); //ArrayBlockingQueue ile 1 ile 100 arası rastgele sayı koyduk.
26         System.out.println("Producer Üretiliyor: " + value);
27     }
28 }
29
30 public void consume() {
31     while(true) {
32         try {
33             Thread.sleep(5000); //5 saniye bekledik.
34         } catch (InterruptedException ex) {
35             Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
36         }
37         try {
38             Integer value=queue.take();
39             System.out.println("Consumer tüketiyor: " + value);
40             System.out.println("Queue size: " + queue.size());
41         } catch (InterruptedException ex) {
42             Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
43         }
44     }
45 }
```

com.mycompany.a.ProducerConsumer > ● produce >

- Producer (Üretici) ve Consumer (Tüketicisi) yapısı hem günlük生活中 hem de programlama yaparken sıkça karşılaşabileceğimiz bir yapıdır. Producer kuyruğa bir şeyler ekler, Consumer ise bu kuyrukta bir şeyler oldukça sıra alır ve ne yapması gerekiyorsa yapar. Günlük生活中 bir örnek verecek olursak; bir bankada müşteriler için sıra numarası üreten cihaza Producer, işlem görmemiş sıra numaralı müşteri oldukça onları LED sıra göstergeleri aracılığıyla çağrııp işlem yapan vezne görevlisi de Consumer olarak düşünülebilir.
- Siz de fark etmişsinizdir, Producer ve Consumer birbirinden bağımsız iki ayrı thread halinde gerçekleştirilebilir. Ancak burada yine ortak bir veri kaynağımız olduğunu gözden kaçırımayalım: Kuyruk (Queue). Yine yardımımıza yetişen java.util.concurrent.* paketinden "thread-safe" bir yardımcı sınıf olan "BlockingQueue" oluyor. BlockingQueue aslında bir arabirim (interface) ve biz onun gerçekleştirmelerinden (implementation) biri olan "ArrayBlockingQueue" sınıfını kullanacağız. ArrayBlockingQueue tipinde bir kuyruk oluştururken bir kapasite belirtiriz. Bu kapasite dolu ise kuyruğa yeni eleman eklemek isteyen thread bekletilir. Benzer şekilde eğer kuyrukta eleman yoksa kuyruktan eleman almak isteyen thread kuyrukta eleman oluncaya kadar bekletilir.

```

Producer Üretiliyor: 15
Producer Üretiliyor: 66
Consumer tüketiyor: 77
Queue size: 3
Producer Üretiliyor: 43
Producer Üretiliyor: 96
Producer Üretiliyor: 81
Producer Üretiliyor: 41
Producer Üretiliyor: 2
Consumer tüketiyor: 95
Queue size: 7
Producer Üretiliyor: 55
Producer Üretiliyor: 13
Producer Üretiliyor: 99
Consumer tüketiyor: 15

```

```

Producer Üretiliyor: 13
Producer Üretiliyor: 99
Consumer tüketiyor: 15
Queue size: 9
Producer Üretiliyor: 5
Consumer tüketiyor: 66
Producer Üretiliyor: 44
Queue size: 10
Consumer tüketiyor: 43
Queue size: 10
Producer Üretiliyor: 40
Consumer tüketiyor: 96
Queue size: 10
Producer Üretiliyor: 80

```

- Producer her saniye değer üretirken consumer 5 saniye de bir kere değer alıyor. Consumer 10 eleman olduğunda ise producer görüldüğü gibi yer açılmasını bekliyor.

173. Wait ve Notify Metodları

```
public class Main {
    public static void main(String[] args) {
        WaitNotify wn = new WaitNotify();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {

                wn.thread1Fonksiyonu();
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                wn.thread2Fonksiyonu();
            }
        });
        thread1.start();
        thread2.start();
        try{
            thread1.join();
            thread2.join();
        }
        catch(InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Wait() Method

- Bir thread için wait() methodu çağrıldığı zaman, thread beklemeye zorlanır ta ki başka bir thread tarafından notify() yada notifyAll() methodları tetiklene kadar.

Notify() Method

- Bir thread için notify() methodu çağrıldığı zaman, beklemekte olan thread harekete geçer. Birden fazla Thread için bekleme söz konusu ise hangi Thread in uyanacağı belli değildir. Tamamen implemantasyona bağlı olarak içlerinden bir tanesi alır.

The screenshot shows an IDE interface with two tabs: 'Main.java' and 'WaitNotify.java'. The 'WaitNotify.java' tab is active, displaying the following code:

```

16     public void thread1Fonksiyonu() {
17         synchronized(this){
18             //anahtar tanımlıyoruz.
19             System.out.println("Thread 1 Çalışıyor");
20             System.out.println("Thread 1 Thread2'nin kendisini uyandırmamasını bekliyor...");
21             try {
22                 this.wait();
23                 //Burada thread1Fonksiyonuna beklemesini söyledik.
24                 //Bundan sonrasına thread2Fonksiyonu yapacak.
25             } catch (InterruptedException ex) {
26                 Logger.getLogger(WaitNotify.class.getName()).log(Level.SEVERE, null, ex);
27             }
28             System.out.println("Thread1 uyandı. Devam ediyor.");
29         }
30     }
31     public void thread2Fonksiyonu() {
32         Scanner scanner= new Scanner(System.in);
33         try {
34             Thread.sleep(2000);
35         } catch (InterruptedException ex) {
36             Logger.getLogger(WaitNotify.class.getName()).log(Level.SEVERE, null, ex);
37         }
38         synchronized(this){
39             System.out.println("Thread 2 çalışıyor.");
40             System.out.println("Devam etmek için bir tuşa basın");
41             scanner.nextLine();
42             this.notify();
43             //bununla thread2Fonksiyonunun işini bitirdiğini ve thread1Fonksiyonunun
44             //çalışmaya devam edebileceğini söylüyoruz.Bunun içinde klavyeden bir sayı girilmesi
45             //durumunu koyduk.
46             System.out.println("Uyandırdım ben gidiyorum.");
47         }
}

```

Below the code, there is a terminal window titled '[jar]' showing the execution output:

```

--- exec-maven-plugin:3.0.0:exec (default-cli) @ waitvenotify ---
Thread 1 Çalışıyor
Thread 1 Thread2'nin kendisini uyandırmamasını bekliyor...
Thread 2 çalışıyor.
Devam etmek için bir tuşa basın
fds
Uyandırdım ben gidiyorum.
Thread1 uyandı. Devam ediyor.

```

- Görüldüğü gibi thread1 çalışmayı bırakıp anahtarını thread2ye verdi. Thread2nin işini bitirdiği zaman ise thread 1 devam etti.
- Eğer bir yerde wait varsa bunu notify ile uyandırmalısın yoksa sonsuza denk çalışır.
- Thread1in çalışmaya devam etmesi için thread2 nin bitirmesi gereklidir.

```
public class WaitNotify {
    private Object lock = new Object();

    public void thread1Fonksiyonu() {
        synchronized(lock) {
            System.out.println("Thread 1 Çalışıyor....");
            System.out.println("Thread 1 Thread2'nin kendisini uyanı");

            try {
                lock.wait();
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```



- Synchronized(this) çok önerilmez. Lock oluşturmalıyız.

174. Mini Proje - Wait ve Notify Metodları ile Producer Consumer Problemi

```
public class Main {
    public static void main(String[] args) {
        ProducerConsumer pc = new ProducerConsumer();

        Thread producer = new Thread(new Runnable() {
            @Override
            public void run() {
                pc.produce();
            }
        });
        Thread consumer = new Thread(new Runnable() {
            @Override
            public void run() {
                pc.consume();
            }
        });

        producer.start();
        consumer.start();

        try {
            producer.join();
            consumer.join();
        } catch (InterruptedException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

The screenshot shows a Java IDE interface with two tabs open: Main.java and ProducerConsumer.java. The ProducerConsumer.java tab is active and displays the following code:

```
10  public class ProducerConsumer {
11      Random random = new Random();
12      Object lock = new Object();
13      Queue<Integer> queue= new LinkedList<Integer>();
14      private int limit=10;
15
16      public void produce(){
17          while (true){
18              try {
19                  Thread.sleep(1000);
20              } catch (InterruptedException ex) {
21                  Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
22              }
23              synchronized(lock){
24                  if(queue.size()==limit){
25                      try {
26                          lock.wait();
27                      } catch (InterruptedException ex) {
28                          Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
29                      }
30                  }
31                  Integer value=random.nextInt(100);
32                  queue.offer(value);
33                  System.out.println("Producer Üretiliyor: "+ value);
34                  lock.notify();
35              }
36          }
37      }
38
39      public void consume(){
40          while(true){
41              try {
42                  Thread.sleep(5000);
43              } catch (InterruptedException ex) {
44                  Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
45              }
46              synchronized(lock){
47                  if(queue.size()==0){
48                      try {
49                          lock.wait();
50                      } catch (InterruptedException ex) {
51                          Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
52                      }
53                  }
54                  Integer value=queue.poll();
55                  System.out.println("Comsumer tüketiyor. "+value);
56                  System.out.println("Queue Size: "+ queue.size());
57                  lock.notify();
58              }
59          }
60      }
61  }
62
```

The code implements a producer-consumer pattern using a queue and synchronization locks. The producer thread sleeps for 1000ms and then checks if the queue size reaches the limit (10). If it does, it waits on the lock. Once the queue size drops below the limit, it wakes up and offers a new integer value to the queue, then notifies the consumer. The consumer thread sleeps for 5000ms and then checks if the queue is empty. If it is, it waits on the lock. Once there is a value in the queue, it poll()s it, prints the value and queue size, and then notifies the producer.

The screenshot shows a Java IDE interface with two tabs open: Main.java and ProducerConsumer.java. The ProducerConsumer.java tab is active and displays the following code:

```
34          lock.notify();
35      }
36  }
37
38  public void consume(){
39      while(true){
40          try {
41              Thread.sleep(5000);
42          } catch (InterruptedException ex) {
43              Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
44          }
45          synchronized(lock){
46              if(queue.size()==0){
47                  try {
48                      lock.wait();
49                  } catch (InterruptedException ex) {
50                      Logger.getLogger(ProducerConsumer.class.getName()).log(Level.SEVERE, null, ex);
51                  }
52              }
53              Integer value=queue.poll();
54              System.out.println("Comsumer tüketiyor. "+value);
55              System.out.println("Queue Size: "+ queue.size());
56              lock.notify();
57          }
58      }
59  }
60
61
62
```

This code is identical to the one in the first screenshot, representing the same producer-consumer logic.

```
--- exec-maven-plugin:3.0.0:exec
Producer Üretiliyor: 84
Producer Üretiliyor: 2
Producer Üretiliyor: 39
Producer Üretiliyor: 85
Comsumer tüketiyor. 84
Queue Size: 3
Producer Üretiliyor: 10
Producer Üretiliyor: 25
Producer Üretiliyor: 71
Producer Üretiliyor: 56
Producer Üretiliyor: 36
Comsumer tüketiyor. 2
Queue Size: 7
Producer Üretiliyor: 69
-----
```

```
----- Producer Üretiliyor: 79
Producer Üretiliyor: 66
Comsumer tüketiyor. 39
Queue Size: 9
Producer Üretiliyor: 63
Comsumer tüketiyor. 85
Queue Size: 9
Producer Üretiliyor: 61
Comsumer tüketiyor. 10
Queue Size: 9
Producer Üretiliyor: 34
Comsumer tüketiyor. 25
Queue Size: 9
Producer Üretiliyor: 97
Comsumer tüketiyor. 71
```

175. ReentrantLock ve Condition Sınıfı , await() ve signal() Metodları

- **ReentrantLock sınıfı**, kod kilit mekanizmasını daha esnek bir şekilde sağlayan bir yapı sunar. Computer kanalları her çalışlığında kaynak üzerinde kilit sağlayarak onun değerini setler. Bu aşamada 50 ms askıya alınan thread' ler arasında kiliti elinde bulunduranlar, diğer thread' lere izin vermez. Kiliti elinde bulunduran thread en sonunda kaynağı serbest bırakarak sonlanır.
- Bir önceki yazımında Thread'ler arasında senkronizasyon sağlamak amacıyla “synchronized” anahtar kelimesinden bahsetmiştim. Bu yazıda ise “synchronized”e alternatif olan ReentrantLock sınıfından bahsedeceğim.
- ReentrantLock sınıfı ile aynı anda tek Thread'in iş yapmasını istediğimiz kod bölümlerini işaretleriz.
- Yeniden girilir kilitlerin “synhronized” kod bloklarına bir alternatif olarak nasıl kullanılabileceklerini öğrenmiş olduk. Peki önceki bölümde incelediğimiz ve bize koşula göre bir threadi bekletme ve sonrasında sürdürme işlevsellliğini kazandıran “wait()” ve “notify()” metodlarının bu alternatif yöntemdeki karşılığı nedir?
- Bunun için “lock.newCondition()” çağrısından elde edilecek bir “java.util.concurrent.locks.Condition” nesnesine ihtiyacımız olacak. “Condition” sınıfının “await()” metodu “wait()” metoduna, “signal()” metodu ise “notify()” metoduna karşılık gelmektedir.
- **Thread.sleep(1000)**: Bunu thread2 yazarsak ilk anahtarı thread1 kapmış olur. Onu ayarlıyoruz.

The screenshot shows a Java development environment with two tabs open: "Main.java X" and "ReentrantLockOrnegi.java X". The "Main.java" tab is active, displaying the following code:

```
7
8 import java.util.logging.Level;
9 import java.util.logging.Logger;
10 public class Main {
11     public static void main(String[] args) {
12         ReentrantLockOrnegi re = new ReentrantLockOrnegi();
13
14         Thread thread1 = new Thread(new Runnable() {
15             @Override
16             public void run() {
17                 re.thread1Fonksiyonu();
18             }
19         });
20         Thread thread2 = new Thread(new Runnable() {
21             @Override
22             public void run() {
23                 re.thread2Fonksiyonu();
24             }
25         });
26         thread1.start();
27         thread2.start();
28         try {
29             thread1.join();
30             thread2.join();
31         } catch (InterruptedException ex) {
32             Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
33         }
34         re.threadOver();
35     }
36
37 }
38
```

The screenshot shows an IDE interface with two tabs open: 'Main.java' and 'ReentrantLockOrnegi.java'. The 'ReentrantLockOrnegi.java' tab is active, displaying the following Java code:

```
7 import java.util.logging.Level;
8 import java.util.logging.Logger;
9 public class ReentrantLockOrnegi {
10     private int say=0;
11     private Lock lock= new ReentrantLock();
12     //Lock sınıfı ReentrantLock implemente ettiği için
13     //bu şekilde yazıyoruz.
14     private Condition condition=lock.newCondition();
15     //Condition bir abstract sınıf olduğu direkt obje üretmemiyoruz.
16     //o yüzden referans vererek üretiyoruz.
17     public void arttir(){
18         //Normal şekilde yazıldığı synchronized yazılmadığı için her
19         //thread girebilir.
20         for (int i = 0; i < 10000; i++) {
21             say++;
22         }
23     }
24     public void thread1Fonksiyonu(){
25         lock.lock();
26         System.out.println("Thread 1 çalışıyor");
27         System.out.println("Thread1 uyandırılmayı bekliyor...");
28         try {
29             //Bununla anahtarı kitledik ve başka bir threadin giremeyeceğini söyledik.
30             condition.await();
31         } catch (InterruptedException ex) {
32             Logger.getLogger(ReentrantLockOrnegi.class.getName()).log(Level.SEVERE, null, ex);
33         }
34         try{
35             //Herhangi bir hata çıktığında kodumuz sonsuz döngüye girebilir.
36             //Bu gibi durumların olmaması için finally metoduyla ahta çıkarsa
37             //threadi aç diyebiliriz.
38             System.out.println("Thread 1 uyandırıldı ve işleme devam ediyor...");
39         }
```

The screenshot shows an IDE interface with two tabs: "Main.java" and "ReentrantLockOrnegi.java". The "ReentrantLockOrnegi.java" tab is active, displaying the following Java code:

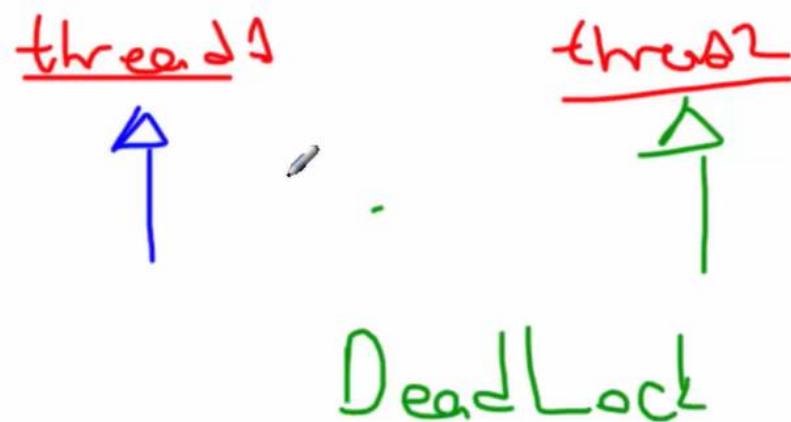
```
37     //threadi aç diyebiliriz.
38     System.out.println("Thread 1 uyandırıldı ve işleme devam ediyor...");
39     arttir();
40 }finally{
41     lock.unlock();
42 //Bununla anahtarı açarak işimizin bittiğini ve tekrar girebileceğini
43 //söylüyoruz.
44 }
45 public void thread2Fonksiyonu(){
46     try {
47         Thread.sleep(1000);
48         //İlk olarak threadin locku almasını istiyoruz o yüzden böyle yaptık.
49     } catch (InterruptedException ex) {
50         Logger.getLogger(ReentrantLockOrnegi.class.getName()).log(Level.SEVERE, null, ex);
51     }
52     Scanner scanner=new Scanner(System.in);
53     System.out.println("Thread 2 çalışıyor.");
54     lock.lock();
55     try{
56         arttir();
57         System.out.println("Devam etmek için bit tuşa basınız");
58         scanner.nextLine();
59         condition.signal();
60         //İşim bitti thread 1 alışabilir demek.
61         System.out.println("Thread 1 uyandırdım.");
62     }finally{
63         lock.unlock();
64     }
65     public void threadOver(){
66         System.out.println("Say değişkeninin değeri: "+say);
67     }
68 }
```

The screenshot shows the "Output - Run (Main)" window of the IDE. It displays the console output from the execution of the Java code. The output shows the interaction between two threads, Thread 1 and Thread 2, regarding a shared resource (lock).

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ ReentrantLock ---
Thread 1 çalışıyor
Thread1 uyandırılmayı bekliyor...
Thread 2 çalışıyor.
Devam etmek için bit tuşa basınız
fdsdf
Thread 1 uyandırdım.
Thread 1 uyandırıldı ve işleme devam ediyor...
Say değişkeninin değeri: 20000
-----
BUILD SUCCESS
-----
Total time: 6.423 s
```

176. Deadlock Oluşma Durumları ve Engelleme Yöntemleri

- Deadlock ya da kilitlenme, iki ya da daha fazla eylemin devam etmek için birbirlerinin bitmesini beklemesi ve sonuçta ikisinin de devam edememesi durumu. Genellikle "yumurta mı tavuk mu önce gelir?" gibi paradokslarda görülür.
- Bilgisayar biliminde, Coffman deadlock iki ya da daha fazla işlemin, diğerinin bir kaynağı bırakmasını beklediği ya da ikiden fazla işlemin döngüsel bir sırada birbirlerinden kaynak beklediği özel durumları belirtmek için kullanılır. Deadlock, birçok işlemin lock (kilit) olarak bilinen özel bir tür kaynağı paylaştığı çoklu işlemede sık karşılaşılan bir sorundur. Zaman paylaşımı ya da gerçek-zaman pazarında kullanılan bilgisayarlar genellikle belirli bir zaman içinde tek bir işlem erişimini garanti eden donanımsal kilit sahiptir. Yazılım kaynaklı deadlocklardan kurtulmak için genel bir çözüm olmadığı için çoğunlukla her seferinde ayrıca çözülmesi gereken bir sorundur.



- 2 lock var birini thread1 diğerini thread2 almış. Thread1 thread2deki locku beklerken thread2de thread 1 deki locku bekliyor. İkisi de diğer locku bulamadığı için ikiside beklemeye başlıyor. Bu durum deadlocktur.
- Deadlock her zaman oluşmaz.

The screenshot shows the source code for the `Hesap` class. The code defines a class with methods for checking account balance, depositing money, and transferring funds between accounts.

```
1 package com.mycompany.deadlock;
2 public class Hesap {
3     private int bakiye = 10000;
4
5     public void paraCek(int miktar) {
6         bakiye -= miktar;
7     }
8
9     public void paraYatir(int miktar) {
10        bakiye += miktar;
11    }
12
13    public static void paraTransferi(Hesap hesap1,Hesap hesap2,int miktar) {
14        hesap1.paraCek(miktar);
15        hesap2.paraYatir(miktar);
16    }
17
18
19    public int getBakiye() {
20        return bakiye;
21    }
22
23    public void setBakiye(int bakiye) {
24        this.bakiye = bakiye;
25    }
26
27 }
```

The screenshot shows the source code for the `Main` class. It creates two threads, `thread1` and `thread2`, which call methods on the `DeadlockOrnegi` object. The `thread1Fonksiyonu` method calls `thread1Fonksiyonu()` and the `thread2Fonksiyonu` method calls `thread2Fonksiyonu()`.

```
11
12 public class Main {
13     public static void main(String[] args) {
14         DeadlockOrnegi deadlock = new DeadlockOrnegi();
15
16         Thread thread1 = new Thread(new Runnable() {
17             @Override
18             public void run() {
19                 deadlock.thread1Fonksiyonu();
20             }
21         });
22         Thread thread2 = new Thread(new Runnable() {
23             @Override
24             public void run() {
25                 deadlock.thread2Fonksiyonu();
26             }
27         });
28         thread1.start();
29
30         thread2.start();
31
32         try {
33             thread1.join();
34             thread2.join();
35         } catch (InterruptedException ex) {
36             Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
37         }
38
39         deadlock.threadOver();
40     }
41 }
42 }
```

The screenshot shows an IDE interface with two main panes. The top pane displays the source code of a Java class named `DeadlockOrnegi`. The code creates two accounts (`Hesap`) and two locks (`lock1`, `lock2`). It defines two threads: `thread1Fonksiyonu` and `thread2Fonksiyonu`, each performing 5000 transfers between the accounts. The bottom pane shows the Maven build output for the project `Deadlock`. The build was successful, producing a `[jar]` file. The console output shows the initial balances of the accounts and the final total balance.

```
public class DeadlockOrnegi {
    private Hesap hesap1=new Hesap();
    private Hesap hesap2=new Hesap();
    private Random random =new Random();
    private Lock lock1= new ReentrantLock();
    private Lock lock2= new ReentrantLock();

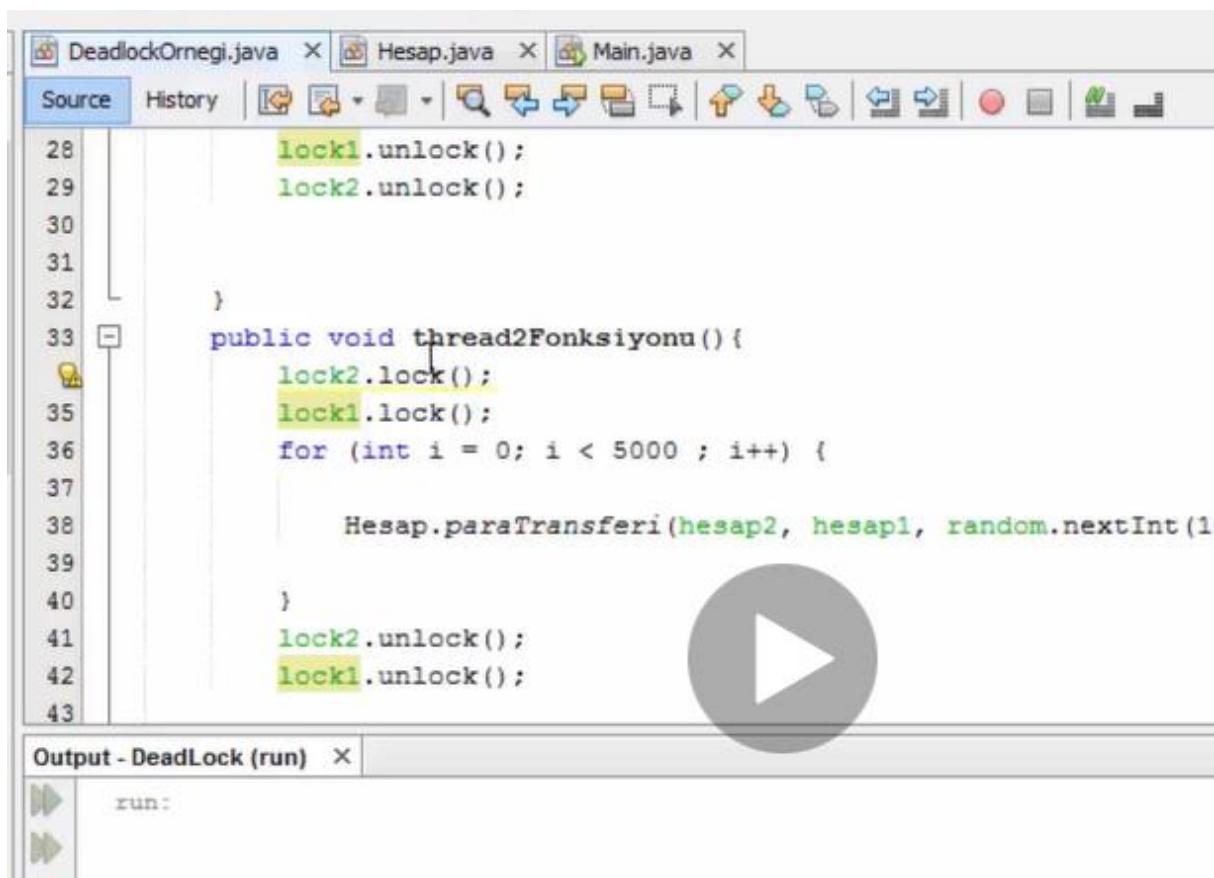
    public void thread1Fonksiyonu(){
        lock1.lock();
        lock2.lock();
        for (int i = 0; i < 5000; i++) {
            Hesap.paraTransferi(hesap1, hesap2, random.nextInt(100));
        }
        lock1.unlock();
        lock2.unlock();
    }

    public void thread2Fonksiyonu(){
        lock1.lock();
        lock2.lock();
        for (int i = 0; i < 5000; i++) {
            Hesap.paraTransferi(hesap2, hesap1, random.nextInt(100));
        }
        lock1.unlock();
        lock2.unlock();
    }

    public void threadOver(){
        System.out.println("Hesap1 Bakiye: "+ hesap1.getBakiye()+" Hesap2 Bakiye: "+hesap2.getBakiye());
        System.out.println("Toplam Bakiye: "+(hesap1.getBakiye()+hesap2.getBakiye()));
    }
}

-----< com.mycompany:Deadlock >-----
[INFO] Building Deadlock 1.0-SNAPSHOT
-----[ jar ]-----
[INFO] --- exec-maven-plugin:3.0.0:exec (default-cli) @ Deadlock ---
Hesap1 Bakiye: 9377 Hesap2 Bakiye: 10623
Toplam Bakiye: 20000
-----
[BUILD SUCCESS]
-----
Total time: 1.333 s
Finished at: 2021-11-15T19:25:21+03:00
-----
```

- Normalde böyle doğru bir şekilde çalışan programda;



```
28         lock1.unlock();
29         lock2.unlock();
30
31
32     }
33     public void thread2Fonksiyonu(){
34         lock2.lock();
35         lock1.lock();
36         for (int i = 0; i < 5000 ; i++) {
37
38             Hesap.paraTransferi(hesap2, hesap1, random.nextInt(100));
39
40         }
41         lock2.unlock();
42         lock1.unlock();
43 }
```

Output - DeadLock (run) X

run:

- Lockların yerini bu şekilde değiştirirsek program deadlock'a girebilir. Çünkü lock1'in thread1'de olduğu anda thread2 o lokumu bekleyebilir. Her zaman olmaz ancak olabileceği durumlar olur. Bunu düzeltmek için try lock kullanabiliriz.
- ReentrantLock'un sağladığı bir avantajda **tryLock** metodudur. Bu metod ile kilidin açık olup olmadığı kontrol edilir eğer kilit açıksa kod parçasına girilir ve işlemlere devam edilir eğer kilit kapalıysa beklemek yerine başka işlemler yapılabilir. Ayrıca tryLock ile bekleme süresi de tanımlayabiliriz.

The screenshot shows an IDE interface with three tabs at the top: 'DeadlockOrnegini.java', 'Hesap.java', and 'Main.java'. The 'Source' tab is selected. The code in the editor is as follows:

```
16
17     public void LocklariKontrolEt(Lock a,Lock b) {
18
19         boolean a_elde_edildi = false;
20         boolean b_elde_edildi = false;
21
22         while(true) {
23
24             a_elde_edildi = a.tryLock();
25             b_elde_edildi = b.tryLock();
26
27             if (a_elde_edildi == true && b_elde_edildi == true) {
28
29                 return;
30             }
31         }
32     }
```

Below the editor is an 'Output - DeadLock (run)' tab with the text 'run:'.

- Lock a ve b parametre olarak alan method ikisinide default olarak false yaptı. tryLock içine aldık. Aldık threadler locku kullanmıyorsa tryLock true kullanıyorsa false döner. Eğer ikisi de kullanılmıyorsa boştaysa return diyerek programı bitirdik. Eğer sadece a elde edilmiş ve b elde edilememişse ayı bırakmasını söylüyoruz. Aynısını b içinde yapıyoruz. Eğer iki threadde boş olursa program sonlanır onun dışında sonsuz döngüde kalır. Hangi threadin hangi locku bırakacağını işletim sistemi kendi belirler.

The screenshot shows an IDE interface with three tabs at the top: 'DeadlockOrnegini.java', 'Hesap.java', and 'Main.java'. The 'Source' tab is selected. The code in the editor is as follows:

```
25
26         b_elde_edildi = b.tryLock();
27
28         if (a_elde_edildi == true && b_elde_edildi == true) {
29
30             return;
31         }
32         if (a_elde_edildi == true) {
33
34             a.unlock();
35         }
36         if (b_elde_edildi == true) {
37             b.unlock();
38         }
39     }
```

Below the editor is an 'Output - DeadLock (run)' tab with the text 'run:'.

The screenshot shows a Java code editor with three tabs at the top: DeadlockOrnegi.java, Hesap.java, and Main.java. The DeadlockOrnegi.java tab is active. The code is as follows:

```
40
41
42     }
43
44     public void thread1Fonksiyonu(){
45
46         LocklariKontrolEt(lock1, lock2);
47
48         for (int i = 0; i < 5000 ; i++) {
49
50             Hesap.paraTransferi(hesap1, hesap2, random.nextInt(100));
51
52         }
53
54     }
55     lock1.unlock();
```

- Lock1.lock ve lock2.lock yerine bu metodu kullanarak deadlocku önlemiş olduk.

177. Semaphore Kullanarak Birden Çok Threading Aynı Anda Çalışması

- **Semaphore**(Semafor) paylaşılan bir veri kaynağına erişimi denetleyen bir mekanizma olarak adlandırılır. Java programlama dilinde bir semaphore bir den fazla thread arasında izin mantığına göre çalışarak senkronizasyonu sağlar. ... Şayet bir thread izin almışsa, kaynağı serbest bırakana kadar ilgili kod bloğunu çalıştırır.
- Semaforlar (Semaphores) -genelde- bir kaynağa erişimi kontrol etmek için kullanılan sayaçlardır. Kilit mekanizmaları gibi threadlerin senkronizasyonu için kullanılırlar fakat kilitlerden farklı olarak bağımsız threadler tarafından serbest bırakılabilirler (release).
- **HATIRLATMA!** Bir thread tarafından elde edilmiş kilidi başka bir thread serbest bırakmaya kalktığında "IllegalMonitorStateException" istisnası fırlatılır.

```
10
11
12     public class Main {
13         public static void main(String[] args) {
14             SemaphoreOrnegi semaphore = new SemaphoreOrnegi();
15
16             Thread thread1 = new Thread(new Runnable() {
17                 @Override
18                 public void run() {
19                     }
20             });
21             Thread thread2 = new Thread(new Runnable() {
22                 @Override
23                 public void run() {
24                     semaphore.threadFonksiyonu(2);
25                 }
26             });
27             Thread thread3 = new Thread(new Runnable() {
28                 @Override
29                 public void run() {
30                     semaphore.threadFonksiyonu(3);
31                 }
32             });
33             Thread thread4 = new Thread(new Runnable() {
34                 @Override
35                 public void run() {
36                     semaphore.threadFonksiyonu(4);
37                 }
38             });
39             Thread thread5 = new Thread(new Runnable() {
40                 @Override
```

The screenshot shows a Java code editor with two tabs: "Main.java X" and "SemaphoreOrnegi.java X". The "Main.java X" tab is active, displaying the following code:

```
34     @Override
35     public void run() {
36         semaphore.threadFonksiyonu(4);
37     }
38 }
39
40 Thread thread5 = new Thread(new Runnable() {
41     @Override
42     public void run() {
43         semaphore.threadFonksiyonu(5);
44     }
45 );
46 thread1.start();
47 thread2.start();
48 thread3.start();
49 thread4.start();
50 thread5.start();
51 try {
52     thread1.join();
53     thread2.join();
54     thread3.join();
55     thread4.join();
56 } catch (InterruptedException ex) {
57     Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
58 }
```

The screenshot shows an IDE interface with two tabs: Main.java and SemaphoreOrnegi.java. The Main.java tab is active, displaying the following code:

```
13  * @author HarunAydemir
14  */
15  public class SemaphoreOrnegi {
16      private Semaphore sem= new Semaphore(3);
17      public void threadFonksiyonu(int id){
18          try {
19              sem.acquire();
20
21              /*Bu değer sıfırsa alt tarafa giremem demek oluyor. Yani 3 olan yer 0 mi
22              diye bakıyor eğer sıfır değilse girişe izin veriyor ve 3'ü bir azaltarak 2
23              yapıyor.*/
24          } catch (InterruptedException ex) {
25              Logger.getLogger(SemaphoreOrnegi.class.getName()).log(Level.SEVERE, null, ex);
26          }
27          System.out.println("Thread Başlıyor... ID: "+ id);
28          try {
29              Thread.sleep(5000);
30          } catch (InterruptedException ex) {
31              Logger.getLogger(SemaphoreOrnegi.class.getName()).log(Level.SEVERE, null, ex);
32          }
33          System.out.println("Thread Çıkıyor... ID: "+id);
34          sem.release();
35          /*Threadin değeri tekrar bir arttırıp girilebileceğini söylüyor.*/
36      }
37  }
```

The screenshot shows the IDE's output window for the com.mycompany.semaphore.SemaphoreOrnegi project. The output log shows the following sequence of events:

```
-- exec-maven-plugin:3.0.0:exec (default-cli)
Thread Başlıyor... ID: 1
Thread Başlıyor... ID: 2
Thread Başlıyor... ID: 3
Thread Çıkıyor... ID: 2
Thread Çıkıyor... ID: 1
Thread Başlıyor... ID: 4
Thread Başlıyor... ID: 5
Thread Çıkıyor... ID: 3
Thread Çıkıyor... ID: 4
Thread Çıkıyor... ID: 5
-----
BUILD SUCCESS
-----
Total time: 12.467 s
```

- Göründüğü gibi çalıştırıldığında 3 thread aynı anda giriyor. Eğer 3 yerine Semaphore(1) yazsaydık synchronized gibi davranış olacaktır.

178. Callable ve Future Interfacelerinin Kullanımı ve Threadlerden Değer Döndürme

The screenshot shows a Java code editor with the following code:

```
1 package com.mycompany.mavenproject1;
2
3
4 import java.util.Random;
5 import java.util.concurrent.ExecutorService;
6 import java.util.concurrent.Executors;
7 import java.util.logging.Level;
8 import java.util.logging.Logger;
9
10
11 public class Main {
12     public static void main(String[] args) {
13         ExecutorService executor= Executors.newFixedThreadPool(1);
14         executor.submit(new Runnable() {
15             @Override
16             public void run() {
17                 try {
18                     Random random = new Random();
19                     System.out.println("Thread çalışıyor");
20                     int sure=random.nextInt(1000)+2000;
21                     Thread.sleep(sure);
22                 } catch (InterruptedException ex) {
23                     Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
24                 }
25                     System.out.println("Threadden çıkışıyor...");
26                 }
27             });
28         executor.shutdown();
29     }
30 }
```

The code demonstrates submitting a task to an executor service. The task is a Runnable that prints "Thread çalışıyor" and sleeps for a random duration between 2000 and 3000 milliseconds. It then prints "Threadden çıkışıyor...". The code editor shows syntax highlighting and some annotations (yellow question marks) on the random number generation and sleep statements.

- Run metodu void bir metoddur. O yüzden değer döndürmez. Ancak threadlerin değer döndürmesini istersek callableve sınıfını kullanmalıyız.
- **Callable** interface i geriye değer dönen bir thread i temsil eder. Runnable interface sine oldukça benzer olmakla birlikte aralarındaki fark Runnable geriye bir değer dönmmez iken Callable geriye bir değer döner. call() methodu tamamlandığında, return edilecek değer bir obje olarak main thread içinde saklanabilmelidir.
- Thread havuzlarını incelediğimiz bölümde görmüştük: "ExecutorService" sınıfının "submit" metoduna yapılacak işleri birer "Runnable" nesnesi şeklinde geçeriz, o da her bir "Runnable" nesnesi için bir thread oluşturur ve başlatır. "submit" metodu eklediğimiz her bir "Runnable" için bir "Future" döndürür. Bu "Future" nesneleri sayesinde asenkron işletilen bu görevlerin tamamlanıp tamamlanmadığını, yoksa iptal mi edildiklerini kontrol edebilir, hatta o görevleri iptal edebiliriz. Ayrıca yine bu "Future" nesnelerini kullanarak threadlerin dönüş değerlerini elde edebiliriz. Şaşırılmış olmalısınız, çünkü "Runnable" arabiriminin "run()" metodу değer döndürmez, yani "void"dir. Değer döndürme ihtiyacımız olduğu durumlarda Java eş zamanlılık kütüphanesinde yer alan "Callable" adlı arabirimini kullanabilir, "submit" metoduna görevlerimizi "Callable" nesneleri olarak geçebiliriz. "Callable" arabirimini jenerik tiplidir ve sahip olduğu tek metot olan "call()" metodу "Callable" jeneriği için belirtilen tipte bir değer döner.

The screenshot shows an IDE interface with two main panes. The top pane displays the `Main.java` file, which contains Java code demonstrating the use of `Callable` and `Future` for thread execution. The bottom pane shows the Maven build output for the project `com.mycompany.mavenproject1`.

```
28     }
29     System.out.println("Threadden çıkiyor...");
30   }
31 } */
```

```
32 /* Callable ve Future generic yapılardır. O yüzden <> şeklinde tanımlarız.*/
33 @Override
34 public Integer call() throws Exception {
35   Random random = new Random();
36   System.out.println("Trhead çalışıyor");
37   int sure=random.nextInt(1000)+2000;
38   try{
39     Thread.sleep(sure);
40   } catch (InterruptedException ex) {
41     Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
42   }
43   System.out.println("Threadden çıkiyor...");
44   return sure;
45 }
```

```
46 });
47 executor.shutdown();
48 System.out.println("Dönen değer: "+future.get());
49 } catch (InterruptedException ex) {
50   Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
51 } catch (ExecutionException ex) {
52   Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
53 }
```

```
54 }
55 }
56 }
```

```
57
58 }
```

Output - Run (Main) X

```
--< com.mycompany:mavenproject1 >-----
[INFO] Building mavenproject1 1.0-SNAPSHOT
[INFO] [ jar ]
[INFO] --- exec-maven-plugin:3.0.0:exec (default-cli) @ mavenproject1 ---
Trhead çalışıyor
Threadden çıkiyor...
Dönen değer2599
[INFO] BUILD SUCCESS
```