**Temel Dosya İşlemleri**

Program çalışırken yani process durumundayken kullandığı veriler kalıcı değildir. Bu veriler işletim sisteminin birincil belleğinde (genel olarak RAM denilmektedir) bulunurlar. Program kapandığında ya da örneğin güç kesildiğinde bu veriler kaybolurlar. Program içerisinde bazı verilerin kalıcı olması istenebilir. Örneğin bir oyun programında o anki oyuncunun o seviyede bulunduğu ekran bilgisi, sağlık bilgisi ve diğer durumlar daha sonra yüklenmek istenebilir. Ya da örneğin bir sipariş bilgisi alındığında bu siparişe ilişkin bilgilerin ayrıca saklanması gerekebilir. Bu tip işlemler tipik olarak dosyalarda saklanarak programlar yazılır.

**Not:** Uygulamalarda kullanılan yardımcı sınıfların kodlarını dökümanın sonunda bulabilirsiniz.

**Bir Processin Çalışma Dizini**

Her processin bir çalışma dizini (current working directory) vardır. Genel olarak bir processin çalışma dizini çalışan dosyanın bulunduğu yerdir. Bir processin çalışma dizini değiştirilebilmektedir. Bir processin çalışma dizini System sınıfın getProperty metoduna “user.dir” anahtar yazısı verilerek elde edilebilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 Console.*writeLine*(System.*getProperty*(**"user.dir"**));  
 }  
}

***Anahtar Notlar:*** *Aslında System sınıfının getProperty metodu ile sisteme ilişkin bir çok bilgi alınabilmektedir. Detaylar için dökümanlar incelenebilir. Örneğin:*

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 Console.*writeLine*(System.*getProperty*(**"user.dir"**));  
 Console.*writeLine*(System.*getProperty*(**"os.arch"**));  
 Console.*writeLine*(System.*getProperty*(**"os.version"**));  
 Console.*writeLine*(System.*getProperty*(**"os.name"**));  
 Console.*writeLine*(System.*getProperty*(**"user.name"**));  
 }  
}

**Dosya Yol İfadeleri**

Bir dosyanın ya da dizinin kalıcı bellek üzerinde nerede olduğunun yazısal ifadesine yol (path) denilmektedir. Yol ifadeleri göreceli (relative) ve mutlak (absolute) olmak üzere ikiye ayrılır. Bir yol ifadesi kök dizinden itibaren bir yol belirtiyorsa bu durumda bu yol ifadesi mutlak bir yol ifadesidir. Tersi durumda yani bir yol ifadesi kök dizinden itibaren bir yol belirtmiyorsa bu durumda göreceli yol ifadesidir:

Örneğin:

“test.dat” → Göreceli yol ifadesi

“/home/testuser/test.dat” → Mutlak yol ifadesi

“[C:\Program](../../../../../C:/Program) Files\x.dat” → Mutlak yol ifadesi

**Anahtar Notlar:** Windows sistemlerinde, Unix/Linux (POSIX) sistemlerinden farklı olarak sürücü kavramı bulunmaktadır. POSIX sistemlerinde kök dizin bir sürücüden başlamaz. Ancak windows sistemlerinde örneğin diskin her parçasının bir sürücüsü bulunabilir. Hatta bir disk sürülere ayrılabilir.

Java da düz bölü ya da ters bölü karakteri yol ifadelerinde kullanılabilir. Peki göreceli yol ifadelerinde doğrudan yol bilinmediğine göre dosya işlemi yapan sınıflar bu dosyayı nerede arayacaktır? Bu durumda ilgili processin çalışma dizini ile göreceli yol ile birleştirilir ve arama o yol ifadesinde yapılır.

Java' da dosya işlemi yapan sınıflar genel olarak iki grupta incelenebilir.

- Dosyanın bütünü üzerinde işlem yapan sınıflar. Örneğin, bir dosyayı silen, kopyalayan, yer değiştiren gibi

- Dosyanın içerisindeki veriler üzerinde işlem yapan sınıflar. Örneğin, dosyaya yazma yapan, dosyadan okuma yapan sınıflar.

Java ' da dosya işlemi yapan sınıflar genel olarak java.io ve java.nio paketlerinde bulunmaktadır.

**File Sınıfı**

Bu sınıf bir dosya ya da dizine (directory) ilişkin bütünsel işlemleri yapan sınıftır. File sınıfının kullanılması için önce programcı dosya yolunu vererek bir File nesnesi oluşturur. File snıfının yol ifadesi alan String parametreli bir başlangıç metodu bulunmaktadır.

Örneğin:

**package** org.csystem.app;  
  
**import** java.io.File;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 File f = **new** File(**"test.dat"**);  
   
 *//...* }  
}

Burada “test.dat” yol ifadesi göreceli olarak verilmiştir. File sınıfının bu başlangıç metodu dosyanın varolup olmadığıyla ilgilenmez. Çünkü, belki de programcı bu dosyayı ya dizini yaratmak için bu sınıfı kullanıyor olabilir.

***Anahtar Notlar:*** *Dosya işlemleri sırasında bir çok durum oluşabilir. Örneğin, bir dosya verildiği dizinde bulunamayabilir, o anki processin o dosyaya yazma hakkı olmayabilir vs. Bu durumda genel olarak metotlar çeşitli exception lar fırlatmaktadır. Programcı bu exceptionlara dikkat etmelidir.*

Sınıfın pek çok yararlı metodu bulunmaktadır:

Sınıfın exists metodu verilen yol ifadesine ilişkin dosya ya da dizinin varlığını test etmek amaçlı kullanılabilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.File;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 String path = Console.*read*(**"Yol ifadesini giriniz:"**);  
 File f = **new** File(path);  
  
 Console.*writeLine*(f.exists() ? **"Dosya var"** : **"Dosya yok"**);  
 }  
}

Sınıfın isDirectory metodu belirtilen yol ifadesinin bir dizin belirtip belirtmediğini test eder. Benzer şekilde isFile metodu da belirtilen yol ifadesinin bir dosya olup olmadığını test eder. Bu metotlar belirtilen yol ifadesi geçersizse yine false döndürürler. Bu durumda programcı bu metotlar false döndüğünde hangi gerekçeden dolayı false döndüğünü anlayamaz. Bunun için en iyi yol öncelikle belirtilen yol ifadesinin varlığını test etmektir. Örneğin:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.File;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 String path = Console.*read*(**"Yol ifadesini giriniz:"**);  
 File f = **new** File(path);  
  
 **if** (f.exists())  
 **if** (f.isDirectory())  
 Console.*writeLine*(**"Belirtilen dizin doğru"**);  
 **else** Console.*writeLine*(**"Belirtilen dosya var"**);  
 **else** Console.*writeLine*(**"Belirtilen yol ifadesi geçersiz"**);  
 }  
}

Sınıfın delete metodu ilgili yol ifadesinde bulunan dosya ya da dizini silmektedir. delete metodu bir dizin silecekse o dizinin boş olması zorunludur. Eğer dizin doluysa delete işlemi yapılamaz. Bu durumda programcı özyinelemeli (recursive) bir algoritma ile silme işlemini yapabilir. Bu durum ileride ele alınacaktır. Bu metot eğer belirtilen yol ifadesi geçersizse false döndürecektir. Yani özetle bu metodun silme yapabilmesi için belirtilen yol bir dosya olmalı ve o dosya varolmalı, ya da belirtilen yol bir dizin olmalı ve o dizin boş olmalıdır. Bu durumlar dışında delete metodu false değer döndürmektedir. Burada programcı silinememe nedenini delete metodunun geri dönüş değerinden anlayamaz. Bu durumda sınıfın exists metodu ile önce dosyanın var olup olmadığı test edilip daha sonra silme işlemi gerçekleştirilebilir:

Örneğin:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.File;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 String dirPath = Console.*read*(**"Yol ifadesini giriniz:"**);  
 File f = **new** File(dirPath);

**if** (f.exists())  
 f.delete();  
 **else** Console.*writeLine*(**"Aranan dosya/dizin bulunamadı"**);  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
 }  
}

Bir dosyanın bir işletim sisteminde saklı (hidden), salt okunur (read only) ve çalışabilen (executable) dosya gibi bir takım bilgileri bulunmaktadır. Programcı bu bilgileri de File sınıfından elde edebilir.

File nesnesi bir dizin gösteriyorsa bu durumda o dizin içerisindeki dosyalar ve dizinler elde edilebilir. Bir dizin içerisindeki dosyaları ve dizinlerin isimleri list metoduyla elde edilebilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.File;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 String dirPath = Console.*read*(**"Yol ifadesini giriniz:"**);  
 File f = **new** File(dirPath);  
  
 **if** (f.exists()) {  
 String[] files = f.list();  
  
 **for** (String file : files)  
 Console.*writeLine*(file);  
 }  
 **else** Console.*writeLine*(**"Yol ifadesi geçersiz"**);  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
 }  
}

Sınıfın listFiles metodu ile o dizin içerisindeki dosya ve dizinler File türünden elde edilebilir. File sınıfının getName isimli metodu yol ifadesi içerisindeki dosya ya da dizin ismi elde edilebilir.

**Sınıf Çalışması:** Komut satırından aşağıdaki gibi çalışan programı yazınız:

*java -jar ListDir.jar <dir path>*

Program komut satırı argümanı verilmezse klavyeden dosya ismini isteyecektir.

Program komut satırından aldığı dizin içerisinde tüm dosya ve dizinleri dizin isimlerinin yanına <DIR> yazısını ekleyerek çıkartacaktır:

**Çözüm:**

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
**import** org.csystem.util.Util;  
  
**import** java.io.File;  
  
**class** App {  
 **public static void** main(String [] args)  
 {  
 args = Util.*getCommandLines*(args, **"Argümanları giriniz:"**);  
  
 **if** (args.**length** != 1) {  
 Console.Error.*writeLine*(**"Kullanım şekli: java App <directory> name VEYA java App"**);  
 System.*exit*(-1);  
 }  
  
 File dir = **new** File(args[0]);  
  
 **if** (dir.isDirectory()) {  
 Console.*writeLine*(**"%s:"**, dir.getAbsolutePath());  
 Console.*writeLine*(**". <DIR>"**);  
 Console.*writeLine*(**".. <DIR>"**);  
 **for** (File file : dir.listFiles())  
 Console.*writeLine*(**"%s%s"**, file.getName(), file.isDirectory() ? **" <DIR>"** : **""**);  
 }  
 **else** Console.Error.*writeLine*(**"Verilen yol ifadesi geçersiz veya bir dizin belirtmiyor"**);  
 }  
}

**Sınıf Çalışması:** Yukarıdaki programı verilen dizin içerisindeki yalnızca dosyaların uzunlukları toplamını bulan program biçimde yazınız.

**Çözüm:**

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
**import** org.csystem.util.Util;  
  
**import** java.io.File;  
  
**class** App {  
 **public static void** main(String [] args)  
 {  
 args = Util.*getCommandLines*(args, **"Argümanları giriniz:"**);  
  
 **if** (args.**length** != 1) {  
 Console.Error.*writeLine*(**"Kullanım şekli: java App <directory> name VEYA java App"**);  
 System.*exit*(-1);  
 }  
  
 File dir = **new** File(args[0]);  
  
 **if** (dir.isDirectory()) {  
 **long** length = 0;  
  
 **for** (File file : dir.listFiles())  
 **if** (file.isFile())  
 length += file.length();  
 Console.*writeLine*(**"Uzunluk:%s"**, length);  
 }  
 **else** Console.Error.*writeLine*(**"Verilen yol ifadesi geçersiz veya bir dizin belirtmiyor"**);  
 }  
}

**Anahtar Notlar:** Modern işletim sistemlerinde her bir dizinin iki tane default dizini bulur. Bunlar “.” ve “..” ile yazılır. “.” o anki bulunulan dizini temsil eder, “..” ise o anki dizinin bir üst dizinini (parent) temsil eder.

File sınıfının mkdir metodu eğer verilen yol ifadesi isimli bir dizin yaratır. Eğer yol ifadesine ilişkin dizin varsa bu durumda hiç birşey yapmaz. Metodun geri dönüş değeri dizinin yaratılıp yaratılmadığını gösteren boolean türden değerdir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.File;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 File f = **new** File(**"test"**);  
  
 Console.*writeLine*(f.mkdir());  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
 }  
}

mkdir metodu yaratılmak istenen dizine ilişkin yol ifadesi geçersizse dizin yaratamaz ve false değerine döner. Verilen yol ifadesine ilişkin tüm dizinlerin yaratılabilmesi için mkdirs metodu kullanılmalıdır:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.File;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 File f = **new** File(**"c:\\users\\csd\\Desktop\\test\\mest\\deneme"**);  
  
 Console.*writeLine*(f.mkdirs());  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
 }  
}

Bu metotlar erişim hakkının bulunmadığı durumlar için exception fırlatırlar.

**Path Arayüzü, Paths ve Files Sınıfları**

Path arayüzü Java 7 ile eklenmiş dosya ve dizinler üzerinde daha kolay işlem yapılmasını sağlayan bir arayüzdür. Kullanımı karışık gibi gözükse de birçok işlemi kolaylaştırmaktadır. Tipik olarak bir Path referansı elde etmek için Paths isimli bir sınıfın static metotları kullanılır. Bu sınıf adeta bir fabrika gibi Path referansı elde etmekte kullanılır. Path arayüzü ve Paths sınfı java.nio.file paketi içerisinde bildirilmiştir. Elde edilen Path referansı Files isimli bir sınıf ile kullanılır. Files sınıfı da java.nio.file paketi içerisinde bildirilmiştir. Basit bir kullanımı şu şekildedir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 Path p = Paths.*get*(**"test.txt"**);  
 **boolean** status = Files.*deleteIfExists*(p);  
  
 **if** (status)  
 Console.*writeLine*(**"Dosya silindi"**);  
 **else** Console.*writeLine*(**"Dosya yok"**);  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
 }  
}

Burada sırasıyla şu işlemler yapılmıştır:

- Paths sınıfı ile bir Path referansı elde edilmiştir.

- Files sınıfının static metotları ile alınan Path referansı kullanılmıştır.

Files sınıfının copy metotları belirli bir kaynaktan başka bir yere kopyalama yapmak amaçlı kullanılmaktadır. Bu metodun bir çok overload u bulunmaktadır. Path türünden iki argümanla çağrılabilen versiyonu birinci parametrede elde edilen path ifadesinin ikinci parametredeki path ifadesine doğrudan kopyalanmasını sağlar. Bu metot kaynak (source) path ile belirtilen dosyayı bulamazsa NoSuchFileException fırlatır. Ayrıca hedef (destination-target) path ifadesine ilişkin bir dosya mevcutsa bu durumda FileAlreadyExistsException nesnesini fırlatır. Örneğin:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.nio.file.\*;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 Path src = Paths.*get*(**"C:\\users\\csd\\Desktop\\test\\App.class"**);  
 Path dest = Paths.*get*(**"C:\\users\\csd\\Desktop\\test\\Sample.class"**);  
  
 Files.*copy*(src, dest);  
 }  
 **catch** (NoSuchFileException ex) {  
 Console.Error.*writeLine*(**"Kaynak dosya bulunamadı"**);  
 }  
 **catch** (FileAlreadyExistsException ex) {  
 Console.Error.*writeLine*(**"Hedef dosya varolduğundan kopyalama yapılamaz"**);  
 }  
 **catch** (Throwable ex) {  
 Console.Error.*writeLine*(**"Belirlenemeyen bir durum oluştu"**);  
 }  
 }  
}

copy metotlarının CopyOption … parametreli versiyonları vardır. CopyOption bir marker arayüzdür. Bu arayüzü destekleyen StandartCopyOptions isimli bir enum içerisinde REPLACE\_EXISTING, COPY\_ATTRIBUTES ve ATOMIC\_MOVE sabit elemanları vardır. Bu elemanlar copy veya diğer bazı metotlara argüman olarak geçirilebilmektedir. Örneğin:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.nio.file.\*;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 Path src = Paths.*get*(**"C:\\users\\csd\\Desktop\\test\\App.class"**);  
 Path dest = Paths.*get*(**"C:\\users\\csd\\Desktop\\test\\Sample.class"**);  
  
 Files.*copy*(src, dest, StandardCopyOption.***REPLACE\_EXISTING***);  
 }  
 **catch** (NoSuchFileException ex) {  
 Console.Error.*writeLine*(**"Kaynak dosya bulunamadı"**);  
 }  
 **catch** (Throwable ex) {  
 Console.Error.*writeLine*(**"Belirlenemeyen bir durum oluştu"**);  
 }  
 }  
}

Bu işlem dizi referansı ile de yapılabilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.nio.file.\*;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 Path src = Paths.*get*(**"C:\\users\\csd\\Desktop\\test\\App.class"**);  
 Path dest = Paths.*get*(**"C:\\users\\csd\\Desktop\\test\\Sample.class"**);  
  
 CopyOption [] copyOptions = {StandardCopyOption.***REPLACE\_EXISTING***, StandardCopyOption.***COPY\_ATTRIBUTES***};  
  
 Files.*copy*(src, dest,copyOptions);  
 }  
 **catch** (NoSuchFileException ex) {  
 Console.Error.*writeLine*(**"Kaynak dosya bulunamadı"**);  
 }  
 **catch** (Throwable ex) {  
 Console.Error.*writeLine*(**"Belirlenemeyen bir durum oluştu"**);  
 }  
 }  
}

Files sınıfının move metodu kaynak path den hedef path e taşıma işlemi yapar. copy metoduna benzer şekilde kullanılabilir. Örneğin:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.nio.file.\*;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** {  
 Path src = Paths.*get*(**"C:\\users\\csd\\Desktop\\test\\App.class"**);  
 Path dest = Paths.*get*(**"C:\\users\\csd\\Desktop\\test\\Sample.class"**);  
  
 CopyOption [] copyOptions = {StandardCopyOption.***REPLACE\_EXISTING***};  
  
 Files.*move*(src, dest,copyOptions);  
 }  
 **catch** (NoSuchFileException ex) {  
 Console.Error.*writeLine*(**"Kaynak dosya bulunamadı"**);  
 }  
 **catch** (Throwable ex) {  
 Console.Error.*writeLine*(**"Belirlenemeyen bir durum oluştu"**);  
 }  
 }  
}

**Sınıf Çalışması:** Komut satırından aşağıdaki gibi çalışan bir programı yazınız.

*java -jar CopyFile.jar <src> <dest>*

Program src ile aldığı dosyayı dest ile aldığı dosya olarak kopyalacaktır. Hedef dosya varsa hedef dosyanın ismi ve uzantısı sonuna “-bak” eklenerek yedeklenecek ve kopyalama işlemi bu işlemden sonra yapılacaktır.

**Çözüm:**

**package** org.csystem.app;  
  
**import** org.csystem.samples.copyapp.CopyApp;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 CopyApp.*run*(args);  
 }  
}

**package** org.csystem.samples.copyapp;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.IOException;  
**import** java.nio.file.\*;  
  
**import static** org.csystem.util.Util.*controlForlengthEquals*;  
**import static** org.csystem.util.Util.*getCommandLines*;  
  
**public final class** CopyApp {  
  
 **private static** Path *ms\_src*, *ms\_dest*;  
  
 **private** CopyApp() {}  
  
 **private static void** backUpAndCopy() **throws** IOException  
 {  
 Files.*move*(*ms\_dest*, Paths.*get*(*ms\_dest*.toString() + **"-bak"**));  
 Files.*copy*(*ms\_src*, *ms\_dest*);  
 Console.*writeLine*(**"Kopyalama tamamlandı dosya yedeklendi"**);  
 }  
  
 **private static void** setPaths(String [] args)  
 {  
 args = *getCommandLines*(args, **"Kaynak dosya ve hedef dosya yol ifadelerini giriniz"**);  
 *controlForlengthEquals*(args, 2, **"Kullanım şekli: java App <src> <dest>"**);  
 *ms\_src* = Paths.*get*(args[0]);  
 *ms\_dest* = Paths.*get*(args[1]);  
 }  
  
 **public static void** run(String [] args)  
 {  
 **try** {  
 **try** {  
 *setPaths*(args);  
 Files.*copy*(*ms\_src*, *ms\_dest*);  
 Console.*writeLine*(**"Kopyalama tamamlandı"**);  
 }  
 **catch** (FileAlreadyExistsException ex) {  
 *backUpAndCopy*();  
 }  
 }  
 **catch** (NoSuchFileException ex) {  
 Console.Error.*writeLine*(**"Kaynak dosya bulunamadı"**);  
 }  
 **catch** (Throwable ex) {  
 Console.Error.*writeLine*(**"Belirlenemeyen bir durum oluştu"**);  
 }  
 }  
}

**Dosya Verileri Üzerinde İşlem Yapan Sınıflar**

Dosyanın verileri üzerinde işlem yapan sınıflar genel olarak iki kategoriye ayrılmıştır. Dosyaya yazma yapan ve okuma yapan sınıflar. Aslında Java'da genel olarak tüm giriş çıkış (input/ output) işlemleri iki parça halinde yapılmaktadır.

**InputStream ve OutputStream Sınıfları**

Temel giriş çıkış işlemleri için InputStream ve OutputStream isimli iki adet sınıf bulunmaktadır. Bu sınıflar sadece dosyalar için değil network gibi diğer kavramlar için de kullanılabilmektedir.

**FileInputStream ve FileOutputStream Sınıfları**

Dosya işlemleri için temel iki sınıf FileInputStream ve FileOutputStream sınıflarıdır. Bu sınıflar AutoCloseable arayüzünü desteklediklerinden “try with resources” deyimi ile kullanılabilirler. Bir dosyanın verileri üzerinde işlem yapılmadan önce o dosyanın açılması gerekir. Dosyanın açılması işlemi bu sınıfların başlangıç metotları tarafından yapılmaktadır. Kapatılması için de close metodu kullanılmalıdır. Bilindiği gibi Java 7 ile eklenen “try with resources” deyimi close işlemini otomatik olarak yapmaktadır. Bu sınıflar java.io paketi içerisinde bildirilmişlerdir.

Bir dosya aslında byte lardan oluşan bir veriler topluluğudur. Programcı açısından dosyanın her byte'ına sıfırdan başlayan bir index numarası verilmiştir. Bir dosya içerisindeki bir byte’ın index numarasına offset denilmektedir. Bir dosya için ismine dosya göstericisi (file pointer) denilen bir bilgi tutulmaktadır. **Dosyaya yazma ve dosyadan okuma yapan metotlar dosya göstericisinin gösterdiği yerden itibaren yazma ya da okuma yaparlar ve okudukları veya yazdıkları miktar kadar bu göstericiyi ilerletirler.**

FileOutputStream sınıfının String ve File parametreli ve bir parametreli başlangıç metotları ile dosya açıldığında dosya göstericisi sıfır numaralı offseti gösterir. Sınıfın boolean parametreli başlangıç metotları dosya göstericisini sona çekerler. Bu durumda her ekleme işlemi de sona yazma yapmışolur.

FileOutputStream sınıfının write metotları dosya göstericisinin bulunduğu yere yazma yaparlar. Bu metotlar byte düzeyinde yazma yaparlar. FileOutputStream sınıfı ile byte düzeyinden başka yazma yapılamaz:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.FileNotFoundException;  
**import** java.io.FileOutputStream;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 String path = **"mest.dat"**;  
  
 **try** (FileOutputStream fos = **new** FileOutputStream(path)) {  
 **for** (**int** i = 1; i <= 10; ++i)  
 fos.write((**byte**)i);  
 }  
 **catch** (FileNotFoundException ex) {  
 Console.*writeLine*(**"%s is not a regular file"**, path);  
 }  
 **catch** (Throwable ex) {  
 Console.*writeLine*(**"Exception:%s, Message:%s"**, ex.getClass().getName(), ex.getMessage());  
 }  
 }  
}  
  
Burada 10 tane bir byte’lık bilgi yazılmıştır. Programcı bir int yazmak isterse (4 byte yazmak isterse) önce bu sayıyı byte dizisine çevirmeli daha sonra bu byte dizisini yazmalıdır. Neyse ki Java’da bu işlemi doğrudan yapan başka yardımcı sınıflar bulunmaktadır. Bu konu ileride ele alınacaktır.

FileOutputStream sınıfının byte [] parametreli iki tane daha write metodu bulunmaktadır. Bu metotlar ile bir byte dizinin tamamı ya da belirli bölümleri yazılabilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.ArrayUtil;  
**import** org.csystem.util.Console;  
  
**import** java.io.FileNotFoundException;  
**import** java.io.FileOutputStream;  
**import** java.util.Random;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 String path = **"numbers.dat"**;  
   
 **try** (FileOutputStream fos = **new** FileOutputStream(path)) {  
 **byte** [] numbers = ArrayUtil.*getRandomArray*(**new** Random(100)

20, (**byte**)0, (**byte**)127);  
  
 ArrayUtil.*display*(numbers);  
 fos.write(numbers);  
 }  
 **catch** (FileNotFoundException ex) {  
 Console.*writeLine*(**"%s is not a regular file"**, path);  
 }  
 **catch** (Throwable ex) {  
 Console.*writeLine*(**"Exception:%s, Message:%s"**, ex.getClass().getName(), ex.getMessage());  
 }  
 }  
}

FileOutputStream sınıfının tek parametreli başlangıç metotları var olan dosyayı budar (truncate) yani varolan bilgiler kaybedilir. boolean parametreli başlangıç metotları ise dosyayı korurlar. Yani bilgiler yok olmaz.

FileInputStream sınıfı ile bir dosya içerisinden byte ya da byte dizisi okunabilir. Buna göre programcı bir byte dizisi okumak isterse ne kadar okuyacağına ilişkin ayrılmış bir byte dizisi vermeli ve read metotlarının bu diziyi doldurmasını sağlamalıdır.

Peki, okumak istenilen miktar okunmadan dosya sonuna gelinmişse nolur? Bu durum için byte dizisi okuyan read metotları okudukları miktarı döndürürler. Bu durumda programcı ne kadar okunduğuna bakabilir. Eğer okuma işlemi dosya sonundan yapılmaya çalışılırsa bu durumda read metodu -1 değerine döner. Öyleyse programcı dosya sonuna gelinceye kadar örneğin 4'er byte lık bilgileri (chunk) şu şekilde bir döngü ile okuyabilir:

**int** nRead;  
**byte** [] data = **new byte**[10];  
  
**while** ((nRead = fis.read(data)) > 0) {  
 **for** (**int** i = 0; i < nRead; ++i)  
 Console.*write*(**"%d "**, (**byte**) data[i]);  
}

FileInputStream sınıfının byte dizisi parametreli read metotları okudukları byte kadar miktarına dönerler. Bu durumda programcı dosyanın sonuna geldiğini read metodunun geri dönüş değerinin -1 olması durumuyla anlayabilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.ArrayUtil;  
**import** org.csystem.util.Console;  
  
**import** java.io.FileInputStream;  
**import** java.io.FileNotFoundException;  
**import** java.util.Random;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 String path = **"numbers.dat"**;  
  
 **try** (FileInputStream fis = **new** FileInputStream(path)) {  
 **byte** [] numbers = ArrayUtil.*getRandomArray*(**new** Random(100), 20, (**byte**)0, (**byte**)127);  
 Console.*writeLine*(**"Generated Numbers:"**);  
 ArrayUtil.*display*(numbers);  
  
 Console.*writeLine*(**"Read Numbers:"**);  
 **int** nRead;  
 **byte** [] data = **new byte**[10];  
  
 **while** ((nRead = fis.read(data)) > 0) {  
 **for** (**int** i = 0; i < nRead; ++i)  
 Console.*write*(**"%d "**, (**byte**) data[i]);  
 }  
  
 Console.*writeLine*();  
 }  
 **catch** (FileNotFoundException ex) {  
 Console.*writeLine*(**"%s is not a regular file"**, path);  
 }  
 **catch** (Throwable ex) {  
 Console.*writeLine*(**"Exception:%s, Message:%s"**, ex.getClass().getName(), ex.getMessage());  
 }  
 }  
}

Files sınıfının InputStream ve path parametreli copy metotları ile bir dosyadan (aslında herhangi bir stream den) okuma yapılarak ilgili dosyaya kopyalanabilir:

**package** org.csystem.app;  
  
**import** java.io.FileInputStream;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** (FileInputStream fis = **new** FileInputStream(**"mest.dat"**)) {  
 Path dest = Paths.*get*(**"test.dat"**);  
 Files.*copy*(fis, dest);  
 }  
 **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

Tersine bir Path referansına ilişkin nesneden bir OutputStream referansına da kopyalama yapılabilir:

**package** org.csystem.app;  
  
**import** java.io.FileOutputStream;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** (FileOutputStream fos = **new** FileOutputStream(**"umut.dat"**)) {  
 Path src = Paths.*get*(**"mest.dat"**);  
 Files.*copy*(src, fos);  
 }  
 **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

**Sınıf Çalışması:** Komut satırından aşağıdaki gibi çalışan programı yazınız.

*java -jar CopyFile.jar <src> <dest> <chunksize>*

- Programda Files sınfı kullanılmayacaktır.

- Program src ile belirtilen dosyayı dest ile belirtilen dosyaya kopyalacaktır.

- Dosya varsa doğrudan üzerine yazılacaktır.

- Chunk size kaynak dosyadan ne kadarlık bloklar halinde kopyalama yapılacağına ilişkin değerdir.

**Çözüm:**

**package** org.csystem.app;  
  
**import** org.csystem.samples.copyappstream.CopyApp;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 CopyApp.*run*(args);  
 }  
}

**package** org.csystem.samples.copyappstream;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.FileInputStream;  
**import** java.io.FileNotFoundException;  
**import** java.io.FileOutputStream;  
**import** java.io.IOException;  
  
**import static** org.csystem.util.Util.*controlForlengthEquals*;  
**import static** org.csystem.util.Util.*getCommandLines*;  
  
**public final class** CopyApp {  
 **private static** String *ms\_srcPath*;  
 **private static** String *ms\_destPath*;  
 **private static int** *ms\_chunkSize*;  
  
 **private** CopyApp() {}  
  
 **private static void** copyFile() **throws** IOException  
 {  
 **try** (FileInputStream fis = **new** FileInputStream(*ms\_srcPath*);  
 FileOutputStream fos = **new** FileOutputStream(*ms\_destPath*)) {  
 **byte**[] data = **new byte**[*ms\_chunkSize*];  
 **int** nRead;  
  
 **while** ((nRead = fis.read(data)) > 0)  
 fos.write(data, 0, nRead);  
  
 Console.*writeLine*(**"Kopyalama işlemi tamamlandı"**);  
 }  
 **finally** {  
  
 }  
 }  
  
 **private static void** setPaths(String [] args) **throws** IOException  
 {  
 args = *getCommandLines*(args, **"Kaynak dosya ve hedef dosya yol ifadelerini giriniz"**);  
 *controlForlengthEquals*(args, 3, **"Kullanım şekli: java App <src> <dest> <chunk size>"**);  
 **try** {  
 *ms\_chunkSize* = Integer.*parseInt*(args[2]);  
 **if** (*ms\_chunkSize* <= 0)  
 **throw new** NumberFormatException();  
  
 *ms\_srcPath* = args[0];  
 *ms\_destPath* = args[1];  
 }  
 **catch** (NumberFormatException ex) {  
 Console.*writeLine*(**"Blok uzunluğu geçersiz"**);  
 System.*exit*(-1);  
 }  
 }  
  
 **public static void** run(String [] args)  
 {  
 **try** {  
 *setPaths*(args);  
  
 *copyFile*();  
 }  
 **catch** (FileNotFoundException ex) {  
 Console.*writeLine*(**"Kaynak dosya bulunamadı"**);  
 }  
 **catch** (Throwable ex) {  
 Console.*writeLine*(**"Belirlenemeyen bir durum oluştu"**);  
 }  
 }  
}

**Sınıf Çalışması:** Komut satırından aşağıdaki gibi çalışan programı yazınız.

*java -jar ConcatFiles.jar file1 file2 file3 … fileN destfile*

- Program file1, file2, …, fileN ile belirtilen dosyaları birleştirerek destFile dosyasına byte byte eklenecektir.

- Files ve Path sınıflarını kullanabilirsiniz.

- Program kaynak dosyalara önceden bakmalı ve bir tanesi bile yoksa mesaj vererek sonlanmalıdır.

- Hedef dosya varsa silinip yenisi oluşturulacaktır.

**Çözüm:**

**package** org.csystem.samples.concatfilesapp;  
  
**public class** App {  
 **public static void** main(String[] args)  
 {  
 ConcatFiles.*run*(args);  
 }  
}

**package** org.csystem.samples.concatfilesapp;  
  
**import** org.csystem.util.Console;  
**import** org.csystem.util.Util;  
  
**import** java.io.\*;  
**import** java.util.ArrayList;  
  
**public class** ConcatFiles {  
 **private static void** appendFiles(String [] args, FileOutputStream fos) **throws** FileNotFoundException, IOException  
 {  
 **int** len = args.**length** - 1;  
 **int** chunkSize = 1024;  
 **byte**[] buf = **new byte**[chunkSize];  
  
 **for** (**int** i = 0; i < len ; ++i) {  
 **try** (FileInputStream fis = **new** FileInputStream(args[i])) {  
 **int** nRead;  
  
 **while** ((nRead = fis.read(buf)) != -1)  
 fos.write(buf, 0, nRead);  
 }  
 **finally** {  
 *//...* }  
 }  
 }  
  
 **private static void** checkForExistance(String [] args)  
 {  
 **int** len = args.**length** - 1;  
 ArrayList<String> files = **new** ArrayList<>();  
  
 **for** (**int** i = 0; i < len; ++i) {  
 **if** (!**new** File(args[i]).exists())  
 files.add(args[i]);  
 }  
  
 **if** (!files.isEmpty()) {  
 Console.*writeLine*(**"Olmayan dosyalar:"**);  
 Console.*writeLine*(files);  
 }  
 }  
  
 **private** ConcatFiles() {}  
  
 **public static void** run(String [] args)  
 {  
 args = Util.*getCommandLines*(args, **"Dosyaları giriniz:"**);  
 Util.*controlForlengthGreater*(args, 1, **"Kullanım şekli:java -jar ConcatFiles.jar file1 file2 file3 … fileN destfile"**);  
 *checkForExistance*(args);  
 **try** (FileOutputStream fos = **new** FileOutputStream(args[args.**length** - 1])) {  
 *appendFiles*(args, fos);  
 }  
 **catch** (Throwable ex) {  
 Console.*writeLine*(**"Beklenmedik bir durum oluştu"**);  
 }  
 }  
}

**Temel Türlerin Dosyaya Yazılması ve Dosyadan Okunması**

Bilindiği gibi FileInputStream ve FileOutputStream sınıfları byte veya byte türden dizilerle işlem yaparlar. Temel türleri ikili formatta dosyaya yazmak için DataOutputStream, dosyadan okumak için de DataInputStream sınıfları kullanılabilir. Bu sınıflar doğrudan dosyalara ilişkin değildir. Tüm I/O işlemleri için kullanılabilmektedir.

DataOutputStream ve DataInputStream sınıfları FileOutputStream ve FileInputStream sınıflarını sarmalayarak kullanılır. Bu durumda, DataOutputStream sınıfının writeXXX metotları temel türleri dosyaya yazılabilir. DataOutputStream sınıfının bir başlangıç metodu vardır. Bu metot OutputStream referansı alır. Bu sınıfın writeXXX metotları yazdığı türü byte dizisine çevirerek OutputStream referansının gösterdiği nesne için write metodunu çağırır. Böylelike programcı herhangi bir işlem yapmadan doğrudan istediği temel türü yazabilir:

**package** org.csystem.app;  
  
**import** java.io.DataOutputStream;  
**import** java.io.FileOutputStream;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** (FileOutputStream fos = **new** FileOutputStream(**"test.dat"**)) {  
 DataOutputStream dos = **new** DataOutputStream(fos);  
  
 **for** (**int** i = 0; i < 10; ++i)  
 dos.writeInt(i);  
 }  
 **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

Benzer şekilde DataInputStream sınıfı da temel türleri okuma amaçlı kullanılabilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.DataInputStream;  
**import** java.io.EOFException;  
**import** java.io.FileInputStream;  
**import** java.io.IOException;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** (FileInputStream fis = **new** FileInputStream(**"test.dat"**)) {  
 DataInputStream dis = **new** DataInputStream(fis);  
  
 **try** {  
 **for** (;;) {  
 **int** val = dis.readInt();  
  
 Console.*write*(**"%d "**, val);  
 Console.*writeLine*();  
 }  
 }  
 **catch** (EOFException ex) {  
 Console.*writeLine*(**"Okuma başarıyla tamamlandı"**);  
 }  
 **catch** (IOException ex) {  
 Console.Error.*writeLine*(**"Dosya formatı bozuk ya da erişilemiyor"**);  
 }  
 }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Burada nasıl bir verinin okunacağı tamamen programın bileceği birşeydir. Dosyalar yalnızca byte topluluğudur. O byte’ların ne anlama geldiği programın sorumluluğundadır:

Yazma yapan program:

**package** org.csystem.app;  
  
**import** java.io.DataOutputStream;  
**import** java.io.FileOutputStream;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 **try** (FileOutputStream fos = **new** FileOutputStream(**"test.dat"**)) {  
 DataOutputStream dos = **new** DataOutputStream(fos);  
  
 **for** (**int** i = 0; i < 10; ++i)  
 dos.writeInt(i);  
  
 dos.writeShort(67);  
 }  
 **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

Okuma Yapan Program:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.\*;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 File file = **new** File(**"test.dat"**);  
  
 **if** (file.length() % 4 != 0) {  
 Console.Error.*writeLine*(**"Dosya formatı bozuk"**);  
 System.*exit*(-1);  
 }  
  
 **try** (FileInputStream fis = **new** FileInputStream(file)) {  
 DataInputStream dis = **new** DataInputStream(fis);  
  
 **try** {  
 **for** (;;) {  
 **int** val = dis.readInt();  
  
 Console.*writeLine*(val);  
 }  
 }  
 **catch** (EOFException ex) {  
 Console.*writeLine*(**"Okuma başarıyla tamamlandı"**);  
 }  
 **catch** (IOException ex) {  
 Console.Error.*writeLine*(**"Dosya formatı bozuk ya da erişilemiyor"**);  
 }  
 }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
}

Benzer bir örnek şöyle oluşturulabilir. Dosyamız int, double ve short formatlarında data lardan oluşsun. Bir data nın hangi formatta olduğu doğrudan anlaşılamaz. Ancak her yazılan değerin başına yazılan değerin türü bir byte lık bir değer olarak yazılabilir. Aşağıdaki örnekte int, double ve short türleri için sırasıyla 1, 2 ve 3 sayıları byte olarak verilerin başına yazılmıştır:

Yazma yapan Program:

**package** org.csystem.app;  
  
**import** java.io.DataOutputStream;  
**import** java.io.FileOutputStream;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 **try** (FileOutputStream fos = **new** FileOutputStream(**"test.dat"**)) {  
 DataOutputStream dos = **new** DataOutputStream(fos);  
  
 dos.writeByte(1);  
 dos.writeInt(67);  
  
 dos.writeByte(2);  
 dos.writeDouble(67.879);  
  
 dos.writeByte(3);  
 dos.writeShort(100);  
  
 dos.writeByte(2);  
 dos.writeDouble(Math.***PI***);  
 }  
 **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

Okuma Yapan Program:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.\*;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 File file = **new** File(**"test.dat"**);  
  
 **try** (FileInputStream fis = **new** FileInputStream(file)) {  
 DataInputStream dis = **new** DataInputStream(fis);  
  
 **try** {  
 **for** (;;) {  
 **byte** type = dis.readByte();  
  
 **switch** (type) {  
 **case** 1:  
 Console.*writeLine*(dis.readInt());  
 **break**;  
 **case** 2:  
 Console.*writeLine*(dis.readDouble());  
 **break**;  
 **case** 3:  
 Console.*writeLine*(dis.readShort());  
 **break**;  
 **default**:  
 **throw new** IOException(**"Dosya formatı bozuk"**);  
 }  
 }  
 }  
 **catch** (EOFException ex) {  
 Console.*writeLine*(**"Okuma başarıyla tamamlandı"**);  
 }  
 }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Bir sınıf nesnesi de dosyaya yazılabilir. Bunun için yalın olarak saınıfın istenilen elemanları dosyaya yazılır:

**class** Product {  
 **private int m\_id**;  
 **private** String **m\_name**;  
 **private double m\_price**;  
  
 **public** Product(**int** id, String name, **double** price)  
 {  
 *//...* **m\_id** = id;  
 **m\_name** = name;  
 **m\_price** = price;  
 }  
 **public int** getId()  
 {  
 **return m\_id**;  
 }  
 **public void** setId(**int** id)  
 {  
 **m\_id** = id;  
 }  
 **public** String getName()  
 {  
 **return m\_name**;  
 }  
 **public void** setName(String name)  
 {  
 **m\_name** = name;  
 }  
 **public double** getPrice()  
 {  
 **return m\_price**;  
 }  
 **public void** setPrice(**double** price)  
 {  
 **m\_price** = price;  
 }

**public** String toString()  
 {  
 **return** String.*format*(**"[%s]-%.2f TL"**, **m\_name**, **m\_price**);  
 }  
}

Yazma yapan Program:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.DataOutputStream;  
**import** java.io.FileOutputStream;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 **try** (FileOutputStream fos = **new** FileOutputStream(**"products.dat"**, **true**)) {  
 DataOutputStream dos = **new** DataOutputStream(fos);  
  
 **for** (;;) {  
 **int** id = Console.*readInt*(**"id bilgisini giriniz:"**);  
  
 **if** (id == 0)  
 **break**;  
  
 String name = Console.*read*(**"Ürün ismini giriniz:"**);  
 **double** price = Console.*readDouble*(**"Ürün birim fiyatını giriniz:"**);  
  
 Product p = **new** Product(id, name, price);  
  
 dos.writeInt(p.getId());  
 dos.writeUTF(p.getName());  
 dos.writeDouble(p.getPrice());  
 }  
 }  
 **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

Okuma yapan program:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.DataInputStream;  
**import** java.io.EOFException;  
**import** java.io.File;  
**import** java.io.FileInputStream;  
  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 File file = **new** File(**"products.dat"**);  
  
 **try** (FileInputStream fis = **new** FileInputStream(file)) {  
 DataInputStream dis = **new** DataInputStream(fis);  
  
 **try** {  
 **for** (;;) {  
 **int** id = dis.readInt();  
 String name = dis.readUTF();  
 **double** price = dis.readDouble();  
  
 Product p = **new** Product(id, name, price);  
  
 Console.*writeLine*(p);  
 }  
 }  
 **catch** (EOFException ex) {  
 Console.*writeLine*(**"Okuma başarıyla tamamlandı"**);  
 }  
 }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}  
  
**Sınıf Çalışması:** Aşağıdaki gibi açıklanan programı yazınız.

- Program içerisinde bir menu bulunacaktır.

- Menuler sırasıyla, “Ekle”, “Listele” ve “Çıkış” şeklinde belirlenecektir.

- Ürünlere ilişkin bilgiler products.dat isimli bir dosyada saklanacaktır.

- Console sınıfı kullanarak klavyeden güvenli girişler sağlanabilir.

**Sınıf Çalışması:** Yukarıdaki programı aynı id değerinden bir daha eklenemeyecek şekilde yazınız.

**Sınıf Çalışması:** Yukarıdaki programı id değerini otomatik artıracak şekilde yeniden yazınız.

**Metin (Text) Dosyaları**

Java'da bir grup dosya sınıf metin (text) dosyalar üzerinde işlem yapabilmektedir. Metin dosyaları kolay okunabilir ve manuel değiştirilebilir olduğundan bazı uygulamalarda tercih edilmektedir. Burada BufferedWriter ve BufferedReader sınıfları ele alınacaktır. Bu sınıflar da aslında doğrudan dosyalarla ilgili değildir. Her hangi bir stream için çalışabilirler.

***Anahtar Notlar:*** *Windows sistemleri ile POSIX (Unix/Linux ve Mac OS X) sistemleri arasında text dosyalar farklılık göstermektedir. Windows sistemlerinde text dosyalarda bir alt satıra inmek için CR (carriage return) ve LF (Line feed) karakterlerinin yanyana yazılması gerekmektedir. Bu yüzden faklı sistemlerde hazırlanmış metin dosyaları ile çalışırken dikkatli olunması gerekir. Bu konu ileride detaylı olarak ele alınacaktır.*

**BufferredWriter sınıfı**

BufferedWriter sınıfı metin dosyalara yazma yapmak için kullanılabilir. Sınıfın Writer parametreli başlangıç metodu ile nesne yaratılabilir. BufferedWriter sınıfının write metotları ile dosyaya yazma yapılabilir.

Windows ve POSIX sistemlerindeki farklılık yüzünden bir alt satıra inmek için newLine metodu kullanılmalıdır. BufferedWriter sınıfı adından da anlaşılacağı gibi tamponlu şekilde çalışır. Yani yazma işlemini ya tampon dolduğunda ya da dosya kapatıldığında yapar. Eğer programcı yazma işleminin hemen gerçekleşmesini istiyorsa flush metodunu çağırmalıdır. Örneğin, klavyeden “exit” girilene kadar dosyaya girilen yazıları ayrı ayrı satırlarda yazan program şu şekildedir:

**package** org.csystem.app;  
  
  
**import** org.csystem.util.Console;  
  
**import** java.io.BufferedWriter;  
**import** java.io.FileWriter;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 **try** (BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(**"test.txt"**, **true**))) {  
 String line;  
 Console.*writeLine*(**"Yazıları giriniz:"**);  
 **for** (;;) {  
 line = Console.*readLine*();  
  
 **if** (line.equals(**"exit"**))  
 **break**;  
  
 bw.write(line);  
 bw.newLine();  
 bw.flush();  
 }  
 }  
 **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

**BufferredReader sınıfı**

BufferedReader sınıfı text bir dosyadan okuma yapmak için kullanılan bir sınıftır. Sınıfın readLine metodu satır satır okumak amaçlı kullanılmaktadır. Metot dosya sonuna geldiğinde null değeriyle geri döner. Tipik bir satır satır okuma işlemi aşağıdaki gibi yapılabilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.BufferedReader;  
**import** java.io.FileReader;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 **try** (BufferedReader br = **new** BufferedReader(**new** FileReader(**"test.txt"**))) {  
 String line;  
  
 **while** ((line = br.readLine()) != **null**)  
 Console.*writeLine*(line);  
 }  
 **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

Örneğin, bir text içerisindeki her bir satırda sayılar bulunuyor olsun. Tüm satırlardaki sayıların ayrı ayrı toplamları aşağıdaki gibi bulunabilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.BufferedReader;  
**import** java.io.FileReader;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 **try** (BufferedReader br = **new** BufferedReader(**new** FileReader(**"test.txt"**))) {  
 String line;  
  
 **int** count = 1;  
 **while** ((line = br.readLine()) != **null**) {  
 line = line.trim();  
 String [] valsStr = line.split(**"[ \t]+"**);  
  
 **int** sum = 0;  
  
 **for** (String s : valsStr)  
 sum += Integer.*parseInt*(s);  
  
 Console.*writeLine*(**"Sum of line %d = %d"**, count++, sum);  
 }  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
 }  
}

**Sınıf Çalışması:** Dosya formatı aşağıdaki gibi olan ve içerisinde öğrencinin adı soyadı, vize ve final notlarının bulunduğu bilgilerden, notu en yüksek olan öğrenci, notu en düşük olan öğrenci bilgileri ile sınıfın ortalamasını results.txt isimli bir dosyaya yazan programı yazınız.

**Dosya Formatı:**

Yusuf Bıçakçıoğlu 90 85

Furkan Tüzemen 100 70

Ediz Mert Demir 80 45

Batuhan Beçik 50 40

Ozan Cıngır 20 10

**Çözüm:**

**package** org.csystem.samples.concatfilesapp;  
  
**public class** Student {  
 **private** String **m\_name**;  
 **private int m\_midtermGrade**, **m\_finalGrade**;  
  
 **public** Student(String name, **int** midtermGrade, **int** finalGrade)  
 {  
 **m\_name** = name;  
 **m\_midtermGrade** = midtermGrade;  
 **m\_finalGrade** = finalGrade;  
 }  
  
 **public** String getName()  
 {  
 **return m\_name**;  
 }  
  
 **public void** setName(String name)  
 {  
 **m\_name** = name;  
 }  
  
 **public int** getMidtermGrade()  
 {  
 **return m\_midtermGrade**;  
 }  
  
 **public void** setMidtermGrade(**int** midtermGrade)  
 {  
 **m\_midtermGrade** = midtermGrade;  
 }  
  
 **public int** getFinalGrade()  
 {  
 **return m\_finalGrade**;  
 }  
  
 **public void** setFinalGrade(**int** finalGrade)  
 {  
 **m\_finalGrade** = finalGrade;  
 }  
  
 **public double** getResultGrade()  
 {  
 **return** 0.4 \* **m\_midtermGrade** + 0.6 \* **m\_finalGrade**;  
 }  
  
 **public** String toString()  
 {  
 **double** resultGrade = getResultGrade();  
  
 **return** String.*format*(**"%s%d %d %.2f %c"**, **m\_name**, **m\_midtermGrade**, **m\_finalGrade**, resultGrade, resultGrade >= 50 ? **'S'** : **'F'**);  
 }  
}

**package** org.csystem.samples.concatfilesapp;  
  
**import** org.csystem.util.Console;  
**import** org.csystem.util.Util;  
  
**import** java.io.\*;  
  
**public class** FindGrades {  
 **private static** Student *ms\_maxStudent*, *ms\_minStudent*;  
 **private static double** *ms\_average*;  
 **private static int** *ms\_count*;  
  
 **static** {  
 *ms\_maxStudent* = **new** Student(**""**, -1, -1);  
 *ms\_minStudent* = **new** Student(**""**, 101, 101);  
 }  
  
 **private** FindGrades() {}  
  
 **private static void** writeFile(BufferedWriter bw, String [] personInfoStr, **int** midtermGrade, **int** finalGrade) **throws** IOException  
 {  
 ++*ms\_count*;  
 **int** len = personInfoStr.**length**;  
  
 **double** resultGrade = 0.4 \* midtermGrade + 0.6 \* finalGrade;  
  
 StringBuilder sb = **new** StringBuilder();  
  
 **for** (**int** i = 0; i < len - 2; ++i)  
 sb.append(personInfoStr[i]).append(**" "**);  
  
 Student s = **new** Student(sb.toString(), midtermGrade, finalGrade);  
  
 **if** (s.getResultGrade() > *ms\_maxStudent*.getResultGrade())  
 *ms\_maxStudent* = s;  
  
 **if** (s.getResultGrade() < *ms\_minStudent*.getResultGrade())  
 *ms\_minStudent* = s;  
  
 *ms\_average* += s.getResultGrade();  
  
 String statusStr = resultGrade >= 50 ? **"G"** : **"K"**;  
  
 bw.write(s.toString());  
 bw.newLine();  
 }  
  
 **private static void** calculateResults(BufferedReader br, BufferedWriter bw) **throws** IOException  
 {  
 String line;  
  
 **while** ((line = br.readLine()) != **null**) {  
 String [] personInfoStr = line.split(**"[ \t]+"**);  
 **int** len = personInfoStr.**length**;  
  
 **int** midtermGrade = Integer.*parseInt*(personInfoStr[len - 2]);  
 **int** finalGrade = Integer.*parseInt*(personInfoStr[len - 1]);  
  
 *writeFile*(bw, personInfoStr, midtermGrade, finalGrade);  
 }  
  
 *ms\_average* /= *ms\_count*;  
  
 bw.write(String.*format*(**"Max Student:%s"**, *ms\_maxStudent*));  
 bw.newLine();  
 bw.write(String.*format*(**"Min Student:%s"**, *ms\_minStudent*));  
 bw.newLine();  
 bw.write(**"Average:"** + *ms\_average*);  
 bw.newLine();  
 }  
  
 **public static void** run(String [] args)  
 {  
 args = Util.*getCommandLines*(args, **"Dosyayı giriniz:"**);  
 Util.*controlForlengthEquals*(args, 1, **"Kullanım şekli:java -jar FindGrades.jar gradesfile"**);  
  
 **try** (BufferedReader br = **new** BufferedReader(**new** FileReader(args[0]));  
 BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(**"results.txt"**))) {  
 *calculateResults*(br, bw);  
 }  
 **catch** (FileNotFoundException ex) {  
 Console.Error.*writeLine*(**"Dosya bulunamadı"**);  
 }  
 **catch** (Throwable ex) {  
 **new** File(**"results.txt"**).delete();  
 Console.Error.*writeLine*(**"Dosya formatı bozuk"**);  
 }  
 }  
}

**package** org.csystem.samples.concatfilesapp;  
  
**public class** App {  
 **public static void** main(String[] args)  
 {  
 FindGrades.*run*(args);  
 }  
}

**RandomAccessFile Sınıfı**

RandomAccessFile sınıfı ile dosya üzerinde istenilen offset’e konumlanma yapılarak yazma ya da okuma yapılabilir. RandomAccessFile sınıfı ile dosyanın açılış moduna göre okuma veya hem okuma hem yazma yapılabilir. RandomAccessFile Object sınıfından türetilmiştir. Closeable arayüzünü ve diğer bazı okuma ve yazmaya ilişkin arayüzleri desteklemektedir. Sınfın write ve writeXXX metotları ile yazma işlemi yapılabilir. Read ve readXXX metotları ile de okuma yapılabilir.

Sınıfın iki adet başlangıç metodu bulunmaktadır. Başlangıç metotlarının birinci parametreleri dosyaya ilişkin isim ya da File referansıdır. Başlangıç metotlarının ikinci parametreleri ortaktır. Bu parametreler dosyanın açılış modunu belirlemek için kullanılır. Açılış modları “r”, “rw”, “rws”, “rwd” biçiminde olabilir. Bunlar dışındaki değerler exception oluşmasına yol açar:

“r” → Yalnızca okuma amaçlı açmak için kullanılır. Dosya yoksa exception oluşur.

“rw” → Hem okuma hem de yazma amaçlı kullanılan moddur. Dosya yoksa yaratılır.

“rws” ve “rwd” açılış modları “New IO” mekanizmasına ilişkindir. Burada ele alınmayacaktır.

Sınfın seek metodu ile istenilen bir offsete konumlanılabilir. Sınıfın getFilePointer metodu ile o anki dosya göstericisinin nerede olduğu bilgisi alınabilir:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.EOFException;  
**import** java.io.RandomAccessFile;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 **try** (RandomAccessFile file = **new** RandomAccessFile(**"test.dat"**, **"rw"**)) {  
 file.seek(file.length());  
 Console.*writeLine*(**"File Pointer:%d"**, file.getFilePointer());  
 **for** (**int** i = 0; i < 10; ++i)  
 file.writeInt(i);  
  
 Console.*writeLine*(**"File Pointer:%d"**, file.getFilePointer());  
  
 file.seek(0);  
  
 Console.*writeLine*(**"File Pointer:%d"**, file.getFilePointer());  
  
 **try** {  
 **for** (;;) {  
 **int** val = file.readInt();  
 Console.*writeLine*(val);  
 }  
 }  
 **catch** (EOFException ex) {  
  
 }  
 Console.*writeLine*(**"File Pointer:%d"**, file.getFilePointer());  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
 }  
}

RandomAccessFile sınıfı hiçbir açılış modunda varolan dosyayı sıfırlamaz. Bu işlemi programcı manuel olarak yapmalıdır:

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
  
**import** java.io.EOFException;  
**import** java.io.IOException;  
**import** java.io.RandomAccessFile;  
**import** java.nio.file.Files;  
**import** java.nio.file.Paths;  
  
**class** App {  
 **public static void** main(String []args) **throws** IOException  
 {  
 Files.*deleteIfExists*(Paths.*get*(**"test.dat"**));  
  
 **try** (RandomAccessFile file = **new** RandomAccessFile(**"test.dat"**, **"rw"**)) {  
 Console.*writeLine*(**"File Pointer:%d"**, file.getFilePointer());  
 **for** (**int** i = 0; i < 10; ++i)  
 file.writeInt(i);  
  
 Console.*writeLine*(**"File Pointer:%d"**, file.getFilePointer());  
  
 file.seek(0);  
  
 Console.*writeLine*(**"File Pointer:%d"**, file.getFilePointer());  
  
 **try** {  
 **for** (;;) {  
 **int** val = file.readInt();  
 Console.*writeLine*(val);  
 }  
 }  
 **catch** (EOFException ex) {  
  
 }  
 Console.*writeLine*(**"File Pointer:%d"**, file.getFilePointer());  
 }  
 **catch** (Exception ex) {  
 Console.*writeLine*(ex.getMessage());  
 }  
 }  
}

**Dosya Formatının Kontrolü**

Bir dosyanın formatı programın anladığı bir kavramdır. Aslında dosya byte’lar topluluğundan başka bir şey değildir. Dosya formatlarının kontrolü için çeşitli yöntemler kullanılmaktadır. Bu yöntemler içerisinde yine yoğun olarak kullanılan checksum isimli bir algoritma bulunmaktadır. Bu algoritmaya göre dosyanın içerisindeki tüm byteların değerleri toplanır ve ters işraretlisi de dosyaya yazılır. Böylelikle dosya içerisindeki tüm byte değerleri toplandığında sıfır elde edilir ve kontrol yapılmış olur.

**package** org.csystem.app;  
  
**import** org.csystem.util.Console;  
**import** org.csystem.util.Util;  
  
**import** java.io.RandomAccessFile;  
  
**public class** App {  
 **public static void** main(String []args)  
 {  
 **try** (RandomAccessFile raf = **new** RandomAccessFile(**"checkSum.dat"**, **"rw"**)) {  
 raf.writeByte(0);  
  
 **for** (**int** i = 0; i < 5; ++i)  
 raf.writeInt(i \* 10);  
  
 **byte** csVal = Util.*getCheckSumVal*(raf);  
  
 raf.seek(0);  
 raf.write(-csVal);  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
  
 **try** {  
 **if** (Util.*checkSum*(**"checkSum.dat"**))  
 Console.*writeLine*(**"Doğru format"**);  
 **else** Console.Error.*writeLine*(**"Bozuk dosya formatı"**);  
 }  
 **catch** (Exception ex) {  
  
 }  
 }  
}

**Sınıf Çalışması:** Klavyeden bir n sayısı girilsin. Bu n sayısı kadar [0, 100] aralığında üretilmiş int türden sayılar sırasıyla dosyaya yazılsın. Dosyanın checkSum değerinin (byte cinsinden) yeri rasgele belirlensin. Bu rasgele belirlene yer aslında herhangi bir sayıdan sonra olarak düşünülmelidir. Buna göre ilgili değerleri ve checkSum değerini dosyaya yazan programı yazınız.

**Çözüm:**

**FileStream sınıfı**

Yukarıda da anlatıldığı gibi dosya işlemleri Java’ da biraz karmaşıktır. Dosya işlemleri ayrıca burada ele alınmayan nio paketi içerisinde bulunan bazı yardımcı sınıflarla da yapılabilmektedir. Dosya işlemlerini biraz daha kolay kullanılabilir hale getirmek için FileStream isimli bir sınıf yazılmıştır. FileStream sınıfı FileAccess enum sınıfının kodları aşağıdaki gibidir:

**package** org.csystem.io;  
  
**import** java.io.\*;  
**import** java.nio.channels.FileChannel;  
**import** java.util.Arrays;  
  
**public final class** FileStream **implements** Closeable, DataOutput, DataInput {  
 **private final** RandomAccessFile **m\_raf**;  
 **private final** BufferedReader **m\_br**;  
 **private final** BufferedWriter **m\_bw**;  
  
 **private int** findNullCharacter(**byte** [] data)  
 {  
 **for** (**int** i = 0; i < data.**length**; ++i)  
 **if** (data[i] == 0)  
 **return** i;  
  
 **return** -1;  
 }  
  
 **public** FileStream(String filePath) **throws** IOException  
 {  
 **this**(filePath, FileAccess.***READWRITE***);  
 }  
  
 **public** FileStream(File file) **throws** IOException  
 {  
 **this**(file.getAbsolutePath());  
 }  
   
 **public** FileStream(String filePath, **boolean** append) **throws** IOException  
 {  
 **this**(filePath, FileAccess.***READWRITE***);  
   
 **if** (append)  
 **m\_raf**.seek(**m\_raf**.length());  
 }  
   
 **public** FileStream(File file, **boolean** append) **throws** IOException  
 {  
 **this**(file.getAbsolutePath(), append);  
 }  
   
 **public** FileStream(String filePath, FileAccess fileAccess) **throws** IOException  
 {  
 **if** (filePath == **null** || fileAccess == **null**)  
 **throw new** IllegalArgumentException(**"null value is not accepted"**);  
   
 **m\_raf** = **new** RandomAccessFile(filePath, fileAccess == FileAccess.***READ*** ? **"r"** : **"rw"**);  
 **m\_br** = **new** BufferedReader(**new** FileReader(**m\_raf**.getFD()));  
 **m\_bw** = **new** BufferedWriter(**new** FileWriter(**m\_raf**.getFD()));  
 }  
   
 **public** FileStream(File file, FileAccess fileAccess) **throws** IOException  
 {  
 **this**(file.getAbsolutePath(), fileAccess);  
 }   
  
 **public boolean** equals(Object arg0)  
 {  
 **return m\_raf**.equals(arg0);  
 }  
  
 **public** FileChannel getChannel()  
 {  
 **return m\_raf**.getChannel();  
 }  
  
 **public** FileDescriptor getFD() **throws** IOException  
 {  
 **return m\_raf**.getFD();  
 }  
  
 **public long** getFilePointer() **throws** IOException  
 {  
 **return m\_raf**.getFilePointer();  
 }  
  
 **public** InputStream getInputStream() **throws** IOException  
 {  
 **return new** FileInputStream(**m\_raf**.getFD());  
 }  
  
 **public** OutputStream getOutputStream() **throws** IOException  
 {  
 **return new** FileOutputStream(**m\_raf**.getFD());  
 }  
  
 **public int** hashCode()  
 {  
 **return m\_raf**.hashCode();  
 }  
  
 **public long** length() **throws** IOException  
 {  
 **return m\_raf**.length();  
 }  
  
 **public int** read() **throws** IOException  
 {  
 **return m\_raf**.read();  
 }  
  
 **public int** read(**byte**[] b, **int** off, **int** len) **throws** IOException  
 {  
 **return m\_raf**.read(b, off, len);  
 }  
  
 **public int** read(**byte**[] b) **throws** IOException  
 {  
 **return m\_raf**.read(b);  
 }  
  
 **public boolean** readBoolean() **throws** IOException  
 {  
 **return m\_raf**.readBoolean();  
 }  
  
 **public byte** readByte() **throws** IOException  
 {  
 **return m\_raf**.readByte();  
 }  
  
 **public char** readChar() **throws** IOException  
 {  
 **return m\_raf**.readChar();  
 }  
  
 **public double** readDouble() **throws** IOException  
 {  
 **return m\_raf**.readDouble();  
 }  
  
 **public float** readFloat() **throws** IOException  
 {  
 **return m\_raf**.readFloat();  
 }  
  
 **public void** readFully(**byte**[] b, **int** off, **int** len) **throws** IOException  
 {  
 **m\_raf**.readFully(b, off, len);  
 }  
  
 **public void** readFully(**byte**[] b) **throws** IOException  
 {  
 **m\_raf**.readFully(b);  
 }  
  
 **public int** readInt() **throws** IOException  
 {  
 **return m\_raf**.readInt();  
 }  
  
 **public** String readLine() **throws** IOException  
 {  
 **return m\_br**.readLine();  
 }  
  
 **public long** readLong() **throws** IOException  
 {  
 **return m\_raf**.readLong();  
 }  
   
 @SuppressWarnings(**"resource"**)  
 **public** Object readObject() **throws** IOException, ClassNotFoundException  
 {   
 **return new** ObjectInputStream(**new** FileInputStream(**m\_raf**.getFD())).readObject();  
 }  
  
 **public short** readShort() **throws** IOException  
 {  
 **return m\_raf**.readShort();  
 }  
  
 **public** String readUTF() **throws** IOException  
 {  
 **return m\_raf**.readUTF();  
 }  
  
 **public** String readFixedUTF(**int** size) **throws** IOException  
 {  
 **byte** [] data = **new byte**[size];  
  
 **if** (read(data) < size)  
 **throw new** IOException();  
  
 **int** index = findNullCharacter(data);  
  
 **return** index == -1 ? **""** : **new** String(data, 0, index);  
 }  
  
 **public int** readUnsignedByte() **throws** IOException  
 {  
 **return m\_raf**.readUnsignedByte();  
 }  
  
 **public int** readUnsignedShort() **throws** IOException  
 {  
 **return m\_raf**.readUnsignedShort();  
 }  
  
 **public void** seek(**long** pos) **throws** IOException  
 {  
 **m\_raf**.seek(pos);  
 }  
  
 **public void** seekEnd() **throws** IOException  
 {  
 seek(length());  
 }  
  
 **public void** seekSet() **throws** IOException  
 {  
 seek(0);  
 }  
 **public void** setLength(**long** newLength) **throws** IOException  
 {  
 **m\_raf**.setLength(newLength);  
 }  
  
 **public int** skipBytes(**int** n) **throws** IOException  
 {  
 **return m\_raf**.skipBytes(n);  
 }  
  
 **public void** newLine() **throws** IOException  
 {  
 **m\_bw**.newLine();  
 }  
  
 **public void** write(**byte**[] b, **int** off, **int** len) **throws** IOException  
 {  
 **m\_raf**.write(b, off, len);  
 }  
  
 **public void** write(**byte**[] b) **throws** IOException  
 {  
 **m\_raf**.write(b);  
 }  
  
 **public void** write(**int** b) **throws** IOException  
 {  
 **m\_raf**.write(b);  
 }  
   
 **public void** write(String s) **throws** IOException  
 {   
 **m\_bw**.write(s);  
 **m\_bw**.flush();   
 }  
  
 **public void** writeBoolean(**boolean** v) **throws** IOException  
 {  
 **m\_raf**.writeBoolean(v);  
 }  
  
 **public void** writeByte(**int** v) **throws** IOException  
 {  
 **m\_raf**.writeByte(v);  
 }  
  
 **public void** writeBytes(String s) **throws** IOException  
 {  
 **m\_raf**.writeBytes(s);  
 }  
  
 **public void** writeChar(**int** v) **throws** IOException  
 {  
 **m\_raf**.writeChar(v);  
 }  
  
 **public void** writeChars(String s) **throws** IOException  
 {  
 **m\_raf**.writeChars(s);  
 }  
  
 **public void** writeDouble(**double** v) **throws** IOException  
 {  
 **m\_raf**.writeDouble(v);  
 }  
  
 **public void** writeFloat(**float** v) **throws** IOException  
 {  
 **m\_raf**.writeFloat(v);  
 }  
  
 **public void** writeInt(**int** v) **throws** IOException  
 {  
 **m\_raf**.writeInt(v);  
 }  
   
 **public void** writeLine(String s) **throws** IOException   
 {   
 **m\_bw**.write(s);  
 **m\_bw**.newLine();  
 **m\_bw**.flush();  
 }  
  
 **public void** writeLong(**long** v) **throws** IOException  
 {  
 **m\_raf**.writeLong(v);  
 }  
   
 @SuppressWarnings(**"resource"**)  
 **public void** writeObject(Object o) **throws** IOException  
 {   
 **new** ObjectOutputStream(**new** FileOutputStream(**m\_raf**.getFD())).writeObject(o);  
 }  
  
 **public void** writeShort(**int** v) **throws** IOException  
 {  
 **m\_raf**.writeShort(v);  
 }  
  
 **public void** writeUTF(String str) **throws** IOException  
 {  
 **m\_raf**.writeUTF(str);  
 }  
  
 **public void** writeFixedUTF(String str, **int** size) **throws** IOException  
 {  
 **if** (str.length() > size)  
 **throw new** IOException();  
  
 write(Arrays.*copyOf*(str.getBytes(), size));  
 }  
  
 @Override  
 **public void** close() **throws** IOException  
 {  
 **m\_raf**.close();   
 }  
  
 @Override  
 **public** String toString()  
 {  
 **return m\_raf**.toString();  
 }  
}

**package** org.csystem.io;  
  
**public enum** FileAccess {***READ***, ***READWRITE***}

Bu sınıf genel olarak dosyaya ilişkin yukarıdaki işlemlerin tamamını yapabilmektedir. FileStream sınıfı text veya binary formatlarda da işlem yapabilmektedir:

**package** org.csystem.app;  
  
**import** org.csystem.io.FileAccess;  
**import** org.csystem.io.FileStream;  
**import** org.csystem.util.Console;  
  
**import** java.io.EOFException;  
  
**class** App {  
 **public static void** main(String []args)  
 {  
 **try** (FileStream fs = **new** FileStream(**"test.dat"**, FileAccess.***READWRITE***)) {  
 fs.seekEnd();  
 Console.*writeLine*(**"File Pointer:%d"**, fs.getFilePointer());  
 **for** (**int** i = 0; i < 10; ++i)  
 fs.writeInt(i);  
  
 Console.*writeLine*(**"File Pointer:%d"**, fs.getFilePointer());  
  
 fs.seekSet();  
  
 Console.*writeLine*(**"File Pointer:%d"**, fs.getFilePointer());  
  
 **try** {  
 **for** (;;) {  
 **int** val = fs.readInt();  
 Console.*writeLine*(val);  
 }  
 }  
 **catch** (EOFException ex) {  
  
 }  
 Console.*writeLine*(**"File Pointer:%d"**, fs.getFilePointer());  
 }  
 **catch** (Exception ex) {  
 Console.Error.*writeLine*(ex.getMessage());  
 }  
 }  
}

**Sınıf Çalışması:** Dosya formatı aşağıdaki gibi olan ve içerisinde öğrencinin adı soyadı, vize ve final notlarının bulunduğu bilgilerden, notu en yüksek olan öğrenci, notu en düşük olan öğrenci bilgileri ile sınıfın ortalamasını results.txt isimli bir dosyaya yazan programı yazınız.

Programda FileStream sınıfı kullanılacaktır

**Dosya Formatı:**

Yusuf Bıçakçıoğlu 90 85

Furkan Tüzemen 100 70

Ediz Mert Demir 80 45

Batuhan Beçik 50 40

Ozan Cıngır 20 10

**Çözüm:**

**package** org.csystem.samples.concatfilesapp;  
  
**public class** Student {  
 **private** String **m\_name**;  
 **private int m\_midtermGrade**, **m\_finalGrade**;  
  
 **public** Student(String name, **int** midtermGrade, **int** finalGrade)  
 {  
 **m\_name** = name;  
 **m\_midtermGrade** = midtermGrade;  
 **m\_finalGrade** = finalGrade;  
 }  
  
 **public** String getName()  
 {  
 **return m\_name**;  
 }  
  
 **public void** setName(String name)  
 {  
 **m\_name** = name;  
 }  
  
 **public int** getMidtermGrade()  
 {  
 **return m\_midtermGrade**;  
 }  
  
 **public void** setMidtermGrade(**int** midtermGrade)  
 {  
 **m\_midtermGrade** = midtermGrade;  
 }  
  
 **public int** getFinalGrade()  
 {  
 **return m\_finalGrade**;  
 }  
  
 **public void** setFinalGrade(**int** finalGrade)  
 {  
 **m\_finalGrade** = finalGrade;  
 }  
  
 **public double** getResultGrade()  
 {  
 **return** 0.4 \* **m\_midtermGrade** + 0.6 \* **m\_finalGrade**;  
 }  
  
 **public** String toString()  
 {  
 **double** resultGrade = getResultGrade();  
  
 **return** String.*format*(**"%s%d %d %.2f %c"**, **m\_name**, **m\_midtermGrade**, **m\_finalGrade**, resultGrade, resultGrade >= 50 ? **'S'** : **'F'**);  
 }  
}

**package** org.csystem.samples.concatfilesapp;  
  
**import** org.csystem.io.FileAccess;  
**import** org.csystem.io.FileStream;  
**import** org.csystem.util.Console;  
**import** org.csystem.util.Util;  
  
**import** java.io.\*;  
  
**public class** FindGrades {  
 **private static** Student *ms\_maxStudent*, *ms\_minStudent*;  
 **private static double** *ms\_average*;  
 **private static int** *ms\_count*;  
  
 **static** {  
 *ms\_maxStudent* = **new** Student(**""**, -1, -1);  
 *ms\_minStudent* = **new** Student(**""**, 101, 101);  
 }  
  
 **private** FindGrades() {}  
  
 **private static void** writeFile(FileStream fsWrite, String [] personInfoStr, **int** midtermGrade, **int** finalGrade) **throws** IOException  
 {  
 ++*ms\_count*;  
 **int** len = personInfoStr.**length**;  
  
 **double** resultGrade = 0.4 \* midtermGrade + 0.6 \* finalGrade;  
  
 StringBuilder sb = **new** StringBuilder();  
  
 **for** (**int** i = 0; i < len - 2; ++i)  
 sb.append(personInfoStr[i]).append(**" "**);  
  
 Student s = **new** Student(sb.toString(), midtermGrade, finalGrade);  
  
 **if** (s.getResultGrade() > *ms\_maxStudent*.getResultGrade())  
 *ms\_maxStudent* = s;  
  
 **if** (s.getResultGrade() < *ms\_minStudent*.getResultGrade())  
 *ms\_minStudent* = s;  
  
 *ms\_average* += s.getResultGrade();  
  
 fsWrite.writeLine(s.toString());  
 }  
  
 **private static void** calculateResults(FileStream fsRead, FileStream fsWrite) **throws** IOException  
 {  
 String line;  
  
 **while** ((line = fsRead.readLine()) != **null**) {  
 String [] personInfoStr = line.split(**"[ \t]+"**);  
 **int** len = personInfoStr.**length**;  
  
 **int** midtermGrade = Integer.*parseInt*(personInfoStr[len - 2]);  
 **int** finalGrade = Integer.*parseInt*(personInfoStr[len - 1]);  
  
 *writeFile*(fsWrite, personInfoStr, midtermGrade, finalGrade);  
 }  
  
 *ms\_average* /= *ms\_count*;  
  
 fsWrite.writeLine(String.*format*(**"Max Student:%s"**, *ms\_maxStudent*));  
 fsWrite.writeLine(String.*format*(**"Min Student:%s"**, *ms\_minStudent*));  
 fsWrite.writeLine(**"Average:"** + *ms\_average*);  
 }  
  
 **public static void** run(String [] args)  
 {  
 args = Util.*getCommandLines*(args, **"Dosyayı giriniz:"**);  
 Util.*controlForlengthEquals*(args, 1, **"Kullanım şekli:java -jar FindGrades.jar gradesfile"**);  
  
 **try** (FileStream fsRead = **new** FileStream(args[0], FileAccess.***READ***);  
 FileStream fsWrite = **new** FileStream(**"results.txt"**, FileAccess.***READWRITE***)) {  
 *calculateResults*(fsRead, fsWrite);  
 }  
 **catch** (FileNotFoundException ex) {  
 Console.Error.*writeLine*(**"Dosya bulunamadı"**);  
 }  
 **catch** (Throwable ex) {  
 **new** File(**"results.txt"**).delete();  
 Console.Error.*writeLine*(**"Dosya formatı bozuk:"**);  
 }  
 }  
}  
  
  
**package** org.csystem.samples.concatfilesapp;  
  
**public class** App {  
 **public static void** main(String[] args)  
 {  
 FindGrades.*run*(args);  
 }  
}

**Kodlarda Kullanılan Yardımcı Sınıflar**

Döküman içerisindeki kodlarda kullanılan sınıflar:

**Console Sınıfı**

**package** org.csystem.util;  
  
**import** java.util.Scanner;  
  
**public final class** Console {  
 **private static final** Scanner ***ms\_kb***;  
  
 **static** {  
 ***ms\_kb*** = **new** Scanner(System.***in***);  
 }  
 **private** Console() {}  
  
 **public static class** Error {  
 **private** Error() {}  
 **public static void** write(**boolean** b)  
 {  
 *write*(**"%b"**, b);  
 }  
  
 **public static void** write(**int** val)  
 {  
 *write*(**"%d"**, val);  
 }  
  
 **public static void** write(**long** val)  
 {  
 *write*(**"%d"**, val);  
 }  
  
 **public static void** write(**char** ch)  
 {  
 *write*(**"%c"**, ch);  
 }  
  
 **public static void** write(**double** val)  
 {  
 *write*(**"%f"**, val);  
 }  
  
 **public static void** write(**float** val)  
 {  
 *write*(**"%d"**, val);  
 }  
  
 **public static void** write(String s)  
 {  
 *write*(**"%s"**, s);  
 }  
  
 **public static void** write(**char** [] c)  
 {  
 **for** (**char** ch : c)  
 *write*(ch);  
 }  
  
 **public static void** write(Object obj)  
 {  
 *write*(**"%s"**, obj);  
 }  
  
 **public static void** write(String fmt, Object...objects)  
 {  
 System.***err***.printf(fmt, objects);  
 }  
  
 **public static void** writeLine()  
 {  
 *write*(**"\n"**);  
 }  
  
 **public static void** writeLine(**boolean** b)  
 {  
 *writeLine*(**"%b"**, b);  
 }  
  
 **public static void** writeLine(**int** val)  
 {  
 *writeLine*(**"%d"**, val);  
 }  
  
 **public static void** writeLine(**long** val)  
 {  
 *writeLine*(**"%d"**, val);  
 }  
  
 **public static void** writeLine(**char** ch)  
 {  
 *writeLine*(**"%c"**, ch);  
 }  
  
 **public static void** writeLine(**double** val)  
 {  
 *writeLine*(**"%f"**, val);  
 }  
  
 **public static void** writeLine(**float** val)  
 {  
 *writeLine*(**"%d"**, val);  
 }  
  
 **public static void** writeLine(String s)  
 {  
 *writeLine*(**"%s"**, s);  
 }  
  
 **public static void** writeLine(**char** [] c)  
 {  
 **for** (**char** ch : c)  
 *writeLine*(ch);  
 }  
  
 **public static void** writeLine(Object obj)  
 {  
 *writeLine*(**"%s"**, obj);  
 }  
  
 **public static void** writeLine(String fmt, Object...objects)  
 {  
 *write*(fmt + **"\n"**, objects);  
 }  
 }  
 **public static double** readDouble()  
 {  
 **return** *readDouble*(**""**);  
 }  
  
 **public static double** readDouble(String msg)  
 {  
 **return** *readDouble*(msg, **""**);  
 }  
  
 **public static double** readDouble(String msg, String errMsg)  
 {  
  
 **for** (;;) {  
 **try** {  
 System.***out***.print(msg);  
  
 **return** Double.*parseDouble*(***ms\_kb***.nextLine());  
 }  
 **catch** (NumberFormatException ex) {  
 System.***out***.print(errMsg);  
 }  
 }  
 }  
  
 **public static double** readDoubleLine(String msg)  
 {  
 **return** *readDouble*(msg + **"\n"**, **""**);  
 }  
  
 **public static double** readDoubleLine(String msg, String errMsg)  
 {  
 **return** *readDouble*(msg + **"\n"**, errMsg + **"\n"**);  
 }  
  
 **public static int** readInt()  
 {  
 **return** *readInt*(**""**);  
 }  
  
 **public static int** readInt(String msg)  
 {  
 **return** *readInt*(msg, **""**);  
 }  
  
 **public static int** readInt(String msg, String errMsg)  
 {  
 **for** (;;) {  
 **try** {  
 System.***out***.print(msg);  
  
 **return** Integer.*parseInt*(***ms\_kb***.nextLine());  
 }  
 **catch** (NumberFormatException ex) {  
 System.***out***.print(errMsg);  
 }  
 }  
 }  
  
 **public static int** readIntLine(String msg)  
 {  
 **return** *readInt*(msg + **"\n"**, **""**);  
 }  
  
 **public static int** readIntLine(String msg, String errMsg)  
 {  
 **return** *readInt*(msg + **"\n"**, errMsg + **"\n"**);  
 }  
  
 **public static long** readLong()  
 {  
 **return** *readLong*(**""**);  
 }  
  
 **public static long** readLong(String msg)  
 {  
 **return** *readLong*(msg, **""**);  
 }  
  
 **public static long** readLong(String msg, String errMsg)  
 {  
 **for** (;;) {  
 **try** {  
 System.***out***.print(msg);  
  
 **return** Long.*parseLong*(***ms\_kb***.nextLine());  
 }  
 **catch** (NumberFormatException ex) {  
 System.***out***.print(errMsg);  
 }  
 }  
 }  
  
 **public static long** readLongLine(String msg)  
 {  
 **return** *readLong*(msg + **"\n"**, **""**);  
 }  
  
 **public static long** readLongLine(String msg, String errMsg)  
 {  
 **return** *readLong*(msg + **"\n"**, errMsg + **"\n"**);  
 }  
  
  
 **public static** String read(String msg)  
 {  
 System.***out***.print(msg);  
  
 **return *ms\_kb***.nextLine();  
 }  
  
 **public static** String readLine()  
 {  
 **return** *read*(**""**);  
 }  
  
 **public static** String readLine(String msg)  
 {  
 **return** *read*(msg + **"\n"**);  
 }  
  
 **public static void** write(**boolean** b)  
 {  
 *write*(**"%b"**, b);  
 }  
  
 **public static void** write(**int** val)  
 {  
 *write*(**"%d"**, val);  
 }  
  
 **public static void** write(**long** val)  
 {  
 *write*(**"%d"**, val);  
 }  
  
 **public static void** write(**char** ch)  
 {  
 *write*(**"%c"**, ch);  
 }  
  
 **public static void** write(**double** val)  
 {  
 *write*(**"%f"**, val);  
 }  
  
 **public static void** write(**float** val)  
 {  
 *write*(**"%d"**, val);  
 }  
  
 **public static void** write(String s)  
 {  
 *write*(**"%s"**, s);  
 }  
  
 **public static void** write(**char** [] c)  
 {  
 **for** (**char** ch : c)  
 *write*(ch);  
 }  
  
 **public static void** write(Object obj)  
 {  
 *write*(**"%s"**, obj);  
 }  
  
 **public static void** write(String fmt, Object...objects)  
 {  
 System.***out***.printf(fmt, objects);  
 }  
  
 **public static void** writeLine()  
 {  
 *write*(**"\n"**);  
 }  
  
 **public static void** writeLine(**boolean** b)  
 {  
 *writeLine*(**"%b"**, b);  
 }  
  
 **public static void** writeLine(**int** val)  
 {  
 *writeLine*(**"%d"**, val);  
 }  
  
 **public static void** writeLine(**long** val)  
 {  
 *writeLine*(**"%d"**, val);  
 }  
  
 **public static void** writeLine(**char** ch)  
 {  
 *writeLine*(**"%c"**, ch);  
 }  
  
 **public static void** writeLine(**double** val)  
 {  
 *writeLine*(**"%f"**, val);  
 }  
  
 **public static void** writeLine(**float** val)  
 {  
 *writeLine*(**"%d"**, val);  
 }  
  
 **public static void** writeLine(String s)  
 {  
 *writeLine*(**"%s"**, s);  
 }  
  
 **public static void** writeLine(**char** [] c)  
 {  
 **for** (**char** ch : c)  
 *writeLine*(ch);  
 }  
  
 **public static void** writeLine(Object obj)  
 {  
 *writeLine*(**"%s"**, obj);  
 }  
  
 **public static void** writeLine(String fmt, Object...objects)  
 {  
 *write*(fmt + **"\n"**, objects);  
 }  
}

**ArrayUtil Sınıfı**

**package** org.csystem.util;  
  
**import** java.math.BigDecimal;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.Random;  
  
**public final class** ArrayUtil {  
 **private** ArrayUtil() {}  
  
 **public static int** [][] addMatrices(**int** [][] a, **int** [][] b)  
 {  
 **if** (!*isMatrix*(a) || !*isMatrix*(b) || a.**length** != b.**length** || a[0].**length** != b[0].**length**)  
 **throw new** UnsupportedOperationException(**"Invalid matrices"**);  
  
 **int** row = a.**length**;  
 **int** col = a[0].**length**;  
 **int** [][] c = **new int**[row][col];  
  
 **for** (**int** i = 0; i < row; ++i)  
 **for** (**int** j = 0; j < col; ++j)  
 c[i][j] = a[i][j] + b[i][j];  
  
 **return** c;  
 }  
  
 **public static double** average(**int** [] a)  
 {  
 **return** (**double**)*sum*(a) / a.**length**;  
 }  
  
 **public static void** copy(**int** [] src, **int** [] dest)  
 {  
 *copy*(src, dest, src.**length**);  
 }  
  
 **public static void** copy(**int** [] src, **int** [] dest, **int** size)  
 {  
 **for** (**int** i = 0; i < size; ++i)  
 dest[i] = src[i];  
 }  
  
 **public static void** display(**int**...a)  
 {  
 *display*(1, a);  
 }  
  
 **public static void** display(**int** n, **int** [] a)  
 {  
 String fmt = String.*format*(**"%%0%dd "**, n);  
  
 **for** (**int** val : a)  
 System.***out***.printf(fmt, val);  
  
 System.***out***.println();  
 }  
  
 **public static void** display(**int** []... a)  
 {  
 *display*(1, a);  
 }  
  
 **public static void** display(**int** n, **int** []... a)  
 {  
 **for** (**int** [] array : a)  
 *display*(n, array);  
 }  
  
  
 **public static void** display(**byte**...bytes)  
 {  
 **for** (**byte** b : bytes)  
 System.***out***.printf(**"%d "**, b);  
  
 System.***out***.println();  
 }  
  
 **public static void** display(String [] str)  
 {  
 **for** (String s : str)  
 System.***out***.println(s);  
 }  
  
 **public static void** drawHistogram(**int** [] hist, **int** nMax, **char** ch)  
 {  
 **int** maxCount = *max*(hist);  
  
 **for** (**int** val : hist) {  
 **int** nChars = (**int**)Math.*round*((**double**)val \* nMax / maxCount);  
  
 **while** (nChars-- > 0)  
 System.***out***.print(ch);  
  
 System.***out***.println();  
 }  
 }  
  
 **public static boolean** equalsMatrices(**int** [][] a, **int** [][] b)  
 {  
 **if** (a.**length** != b.**length** || !*isMatrix*(a) || !*isMatrix*(b))  
 **return false**;  
  
 **int** row = a.**length**;  
  
 **for** (**int** i = 0; i < row; ++i)  
 **if** (!Arrays.*equals*(a[i], b[i]))  
 **return false**;  
  
 **return true**;  
 }  
  
 **public static int** [][] getMultipliedByScalar(**int** [][] a, **int** val)  
 {  
 **int** [][] m = **new int** [a.**length**][a[0].**length**];  
  
 **for** (**int** i = 0; i < m.**length**; ++i)  
 **for** (**int** j = 0; j < m[0].**length**; ++j)  
 m[i][j] = -a[i][j];  
  
 **return** m;  
 }  
  
 **public static int** [] getHistogram(**int** [] a, **int** n) *//[0, n]* {  
 **int** [] histogram = **new int**[n + 1];  
  
 **for** (**int** val : a)  
 ++histogram[val];  
  
 **return** histogram;  
 }  
  
 **public static byte** [] getRandomArray(**int** n, **byte** min, **byte** max)  
 {  
 **return** *getRandomArray*(**null**, n, min, max);  
 }  
  
 **public static byte** [] getRandomArray(Random r, **int** n, **byte** min, **byte** max)  
 {  
 **if** (n <= 0 || min > max)  
 **throw new** IllegalArgumentException(**"Illegal argument"**);  
  
 **if** (r == **null**)  
 r = **new** Random();  
  
 **byte** [] result = **new byte**[n];  
  
 **for** (**int** i = 0; i < n; ++i)  
 result[i] = (**byte**)(r.nextInt(max - min) + min);  
  
 **return** result;  
 }  
  
 **public static int** [] getRandomArray(**int** n, **int** min, **int** max) *//[min, max)* {  
 **return** *getRandomArray*(**new** Random(), n, min, max);  
 }  
  
 **public static int** [] getRandomArray(Random r, **int** n, **int** min, **int** max) *//[min, max)* {  
 **if** (n <= 0 || min > max)  
 **throw new** IllegalArgumentException(**"Illegal argument"**);  
  
 **if** (r == **null**)  
 r = **new** Random();  
  
 **int** [] result = **new int**[n];  
  
 **for** (**int** i = 0; i < n; ++i)  
 result[i] = r.nextInt(max - min) + min;  
  
 **return** result;  
 }  
  
  
 **public static double** [] getRandomArray(**int** n, **double** min, **double** max) *//[min, max)* {  
 **return** *getRandomArray*(**new** Random(), n, min, max);  
 }  
  
 **public static double** [] getRandomArray(Random r, **int** n, **double** min, **double** max) *//[min, max)* {  
 **if** (n <= 0 || min > max)  
 **throw new** IllegalArgumentException(**"Illegal argument"**);  
  
 **if** (r == **null**)  
 r = **new** Random();  
  
 **double** [] result = **new double**[n];  
  
 **for** (**int** i = 0; i < n; ++i)  
 result[i] = r.nextDouble() \* (max - min) + min;  
  
 **return** result;  
 }  
  
 **public static int** [][] getRandomMatrix(**int** m, **int** n, **int** min, **int** max) *//[min, max)* {  
 **return** *getRandomMatrix*(**new** Random(), m, n, min, max);  
 }  
  
 **public static int** [][] getRandomMatrix(Random r, **int** m, **int** n, **int** min, **int** max) *//[min, max)* {  
 **if** (r == **null**)  
 r = **new** Random();  
  
 **int** [][] a = **new int**[m][];  
  
 **for** (**int** i = 0; i < m; ++i)  
 a[i] = *getRandomArray*(r, n, min, max);  
  
 **return** a;  
 }  
  
 **public static int** [][] getRandomSquareMatrix(**int** n, **int** min, **int** max) *//[min, max)* {  
 **return** *getRandomSquareMatrix*(**new** Random(), n, min, max);  
 }  
  
 **public static int** [][] getRandomSquareMatrix(Random r, **int** n, **int** min, **int** max) *//[min, max)* {  
 **return** *getRandomMatrix*(r, n, n, min, max);  
 }  
  
  
 **public static double** [][] getRandomMatrix(**int** m, **int** n, **double** min, **double** max) *//[min, max)* {  
 **return** *getRandomMatrix*(**new** Random(), m, n, min, max);  
 }  
  
 **public static double** [][] getRandomMatrix(Random r, **int** m, **int** n, **double** min, **double** max) *//[min, max)* {  
 **if** (r == **null**)  
 r = **new** Random();  
  
 **double** [][] a = **new double**[m][];  
  
 **for** (**int** i = 0; i < m; ++i)  
 a[i] = *getRandomArray*(r, n, min, max);  
  
 **return** a;  
 }  
  
 **public static int** [][] getTranspose(**int** [][] a)  
 {  
 **int** [][] t = **new int**[a[0].**length**][a.**length**];  
  
 **for** (**int** i = 0; i < a.**length**; ++i)  
 **for** (**int** j = 0; j < a[i].**length**; ++j)  
 t[j][i] = a[i][j];  
  
 **return** t;  
 }  
  
 **public static boolean** isMatrix(**int** [][] a)  
 {  
 **for** (**int** i = 1; i < a.**length**; ++i)  
 **if** (a[i].**length** != a[0].**length**)  
 **return false**;  
  
 **return true**;  
 }  
  
 **public static boolean** isSquareMatrix(**int** [][] a)  
 {  
 **return** *isMatrix*(a) && a[0].**length** == a.**length**;  
 }  
  
  
 **public static** String join(ArrayList list, String delim)  
 {  
 String result = **""**;  
  
 **for** (Object o : list)  
 result += o + delim;  
  
 **return** result.isEmpty() ? result : result.substring(0, result.length() - delim.length());  
 }  
  
 **public static** String join(ArrayList list, **char** delim)  
 {  
 **return** *join*(list, delim + **""**);  
 }  
  
 **public static** String join(String [] str, String delim)  
 {  
 String result = **""**;  
  
 **for** (String s : str)  
 result += s + delim;  
  
 **return** result.isEmpty() ? result : result.substring(0, result.length() - delim.length());  
 }  
  
 **public static** String join(String [] s, **char** delim)  
 {  
 **return** *join*(s, delim + **""**);  
 }  
  
 **public static int** max(**int** [] a)  
 {  
 **int** result = a[0];  
  
 **for** (**int** i = 0; i < a.**length**; ++i)  
 **if** (result < a[i])  
 result = a[i];  
  
 **return** result;  
 }  
  
 **public static** BigDecimal max(BigDecimal [] bigDecimals)  
 {  
 BigDecimal result = bigDecimals[0];  
  
 **for** (**int** i = 1; i < bigDecimals.**length**; ++i)  
 result = result.max(bigDecimals[i]);  
  
 **return** result;  
 }  
  
 **public static int** [] merge(**int** [] a, **int**...b)  
 {  
 **int** [] c = **new int**[a.**length** + b.**length**];  
  
 **int** index = 0;  
 **for** (**int** i = 0; i < a.**length**; ++i)  
 c[index++] = a[i];  
  
 **for** (**int** i = 0; i < b.**length**; ++i)  
 c[index++] = b[i];  
  
 **return** c;  
 }  
  
 **public static int** [] merge(**int** []...a)  
 {  
 *//****TODO:* throw new** UnsupportedOperationException();  
 }  
  
 **public static int** min(**int** [] a)  
 {  
 **int** result = a[0];  
  
 **for** (**int** i = 1; i < a.**length**; ++i)  
 **if** (result > a[i])  
 result = a[i];  
  
 **return** result;  
 }  
  
 **public static** BigDecimal min(BigDecimal [] bigDecimals)  
 {  
 BigDecimal result = bigDecimals[0];  
  
 **for** (**int** i = 1; i < bigDecimals.**length**; ++i)  
 result = result.min(bigDecimals[i]);  
  
 **return** result;  
 }  
  
 **public static int**[][] multiplyByScalar(**int** [][] a, **int** val)  
 {  
 **for** (**int** [] array : a)  
 *multiplyByScalar*(array, val);  
  
 **return** a;  
 }  
  
 **public static int**[] multiplyByScalar(**int** [] a, **int** val)  
 {  
 **for** (**int** i = 0; i < a.**length**; ++i)  
 a[i] \*= val;  
  
 **return** a;  
 }  
  
 **public static int** [] resize(**int** [] a, **int** size)  
 {  
 **if** (size <= a.**length**)  
 **return** a;  
  
 **return** Arrays.*copyOf*(a, size);  
 }  
  
 **public static int**[] reverse(**int** [] a)  
 {  
 **int** halfLen = a.**length** / 2;  
 **int** len = a.**length**;  
  
 **for** (**int** i = 0; i < halfLen; ++i) {  
 **int** temp = a[i];  
  
 a[i] = a[len - 1 - i];  
 a[len - 1 - i] = temp;  
 }  
  
 **return** a;  
 }  
  
 **public static char**[] reverse(**char** [] a)  
 {  
 **int** halfLen = a.**length** / 2;  
 **int** len = a.**length**;  
  
 **for** (**int** i = 0; i < halfLen; ++i) {  
 **char** temp = a[i];  
  
 a[i] = a[len - 1 - i];  
 a[len - 1 - i] = temp;  
 }  
  
 **return** a;  
 }  
 **public static int** [][] subMatrices(**int** [][] a, **int** [][] b)  
 {  
 **return** *addMatrices*(a, *getMultipliedByScalar*(b, -1));  
 }  
  
 **public static int** sum(**int**...a)  
 {  
 **return** *sum*(0, a);  
 }  
  
 **public static int** sum(**int** init, **int**...a)  
 {  
 **int** result = init;  
  
 **for** (**int** x : a)  
 result += x;  
  
 **return** result;  
 }  
  
 **public static int** sum(**int** [][] a)  
 {  
 **int** total = 0;  
  
 **for** (**int** [] array : a)  
 total += *sum*(array);  
  
 **return** total;  
 }  
  
 **public static int** [] sumsOfArrays(**int** []...a)  
 {  
 **int** [] sums = **new int**[a.**length**];  
  
 **for** (**int** i = 0; i < a.**length**; ++i)  
 sums[i] = *sum*(a[i]);  
  
 **return** sums;  
 }  
  
 **public static int** sumDiagonal(**int** [][] a)  
 {  
 **int** total = 0;  
  
 **for** (**int** i = 0; i < a.**length**; ++i)  
 total += a[i][i];  
  
 **return** total;  
 }  
}

**Util Sınıfı**

**package** org.csystem.util;  
  
**import** java.io.EOFException;  
**import** java.io.FileInputStream;  
**import** java.io.IOException;  
**import** java.io.RandomAccessFile;  
  
**public final class** Util {  
 **private** Util() {}  
  
 **public static boolean** checkSum(String path) **throws** IOException  
 {  
 **byte** sum = 0;  
  
 **try** (FileInputStream fis = **new** FileInputStream(path)) {  
 **int** val;  
 **while** ((val = fis.read()) != -1)  
 sum += (**byte**)val;  
 }  
 **catch** (IOException ex) {  
 **throw** ex;  
 }  
 **return** sum == 0;  
 }  
 **public static void** controlForlengthEquals(String [] args, **int** val, String msg)  
 {  
 **if** (args.**length** != val) {  
 Console.Error.*writeLine*(msg);  
 System.*exit*(-1);  
 }  
 }  
  
 **public static void** controlForlengthGreater(String [] args, **int** val, String msg)  
 {  
 **if** (args.**length** <= val) {  
 Console.Error.*writeLine*(msg);  
 System.*exit*(-1);  
 }  
 }  
  
 **public static void** controlForlengthGreaterOrEqual(String [] args, **int** val, String msg)  
 {  
 **if** (args.**length** < val) {  
 Console.Error.*writeLine*(msg);  
 System.*exit*(-1);  
 }  
 }  
  
 **public static byte** getCheckSumVal(RandomAccessFile raf) **throws** IOException  
 {  
 raf.seek(1);  
  
 **byte** sum = 0;  
  
 **try** {  
 **for** (;;) {  
 sum += raf.readByte();  
 }  
 }  
 **catch** (EOFException ex) {  
  
 }  
 **return** sum;  
 }  
  
 **public static** String [] getCommandLines(String [] args, String msg)  
 {  
 **if** (args.**length** == 0)  
 **return** Console.*read*(msg).split(**"[ \t\r]+"**);  
  
 **return** args;  
 }  
}