File Writing

Java Basics kategorisinin altıncı topic'i (File I/O'nun yazma yarısı): `Files.writeString()` / `Files.write()`, `StandardOpenOption.APPEND` ile sona ekleme, `BufferedWriter` ile yazma, `Files.copy()` ile dosya kopyalama, `Files.createDirectories()`, bir dizin ağacını silmenin standart deseni, ve CSV dosyası yazma.

Başlangıç 20 dk

Son güncelleme: 31 Ekim 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/file-writing

---

File Writing

Java Temelleri kategorisinin yedinci konusu File I/O'nun YAZMA yarısı -- "File Reading" dersinin doğrudan devamı. O derste dosyaları okumanın iki yolunu (`java.nio.file.Files` ve klasik `java.io.BufferedReader`) gördük; burada aynı iki API ailesiyle dosyalara YAZMAYI, dosyaları KOPYALAMAYI, ve dizinleri YÖNETMEYİ ele alıyoruz.

File Writing Nedir?

Dosyaya yazmak, "File Reading" dersindeki okuma API'lerinin ayna görüntüsüdür: `Files.writeString()` / `Files.write()` (modern, `java.nio.file`) ve `BufferedWriter` + `FileWriter` (klasik, `java.io` , try-with-resources ile). Ek olarak bu ders, tek bir dosyaya yazmanın ötesine geçerek dosya KOPYALAMA (`Files.copy()`) ve dizin OLUŞTURMA/SİLME (`Files.createDirectories()` , `Files.walk()`) konularını da kapsıyor.

Neden Var?

Bir programın sonuçlarını KALICI hale getirmesi -- bir rapor üretmek, bir log dosyası tutmak, bir yapılandırmayı kaydetmek, bir kullanıcı yüklemesini diske yazmak -- File Writing gerektirir. Modern `Files.writeString()` / `Files.write()` metotları, eski `java.io` 'nun gerektirdiği "akış aç → yaz → akışı kapat" ritüelini TEK BİR statik metot çağrısına indirger; ama `BufferedWriter` gibi klasik sınıflar, özellikle çok sayıda küçük yazma işlemini VERİMLİ bir şekilde biriktirmek gerektiğinde hâlâ tercih edilir.

Tarihçe

"File Reading" dersindeki tarihçenin AYNISI burada da geçerlidir: `java.io` (`FileWriter`/`BufferedWriter`) Java 1.0'dan (1996) beri var, `java.nio.file` (`Path`/`Files`) Java 7 (2011) ile geldi. `Files.writeString()` Java 11 (2018) ile eklendi -- ondan önce, tek bir `String`'i bir dosyaya yazmak için `Files.write(path, content.getBytes())` gibi bir dönüştürme adımı gerekiyordu.

Files.writeString(): Oluşturma ve Üzerine Yazma

`Files.writeString(path, içerik)`, en basit yazma yoludur -- dosya yoksa OLUŞTURUR, varsa TAMAMEN ÜZERİNE YAZAR (üzerine yazma, biriktirme değil). Aynı dosyaya art arda iki kez çağırmak, yalnızca İKİNCİ çağrının içeriğini bırakır.

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class WriteStringExample {
    public static void main(String[] args) throws IOException {
        Path path = Path.of("demo-write.txt");

        // Files.writeString() is the simplest way to write text to a file --
        // it CREATES the file if it doesn't exist, or OVERWRITES it entirely
        // if it does. There is no accumulation here: calling it twice in a row
        // leaves only the SECOND call's content.
        Files.writeString(path, "Hello, File!");
        System.out.println("After first write: " + Files.readString(path));

        Files.writeString(path, "This REPLACES the previous content entirely.");
        System.out.println("After second write (overwritten): " + Files.readString(path));

        Files.deleteIfExists(path);
    }
}
```

Beklenen Çıktı

```
After first write: Hello, File!
After second write (overwritten): This REPLACES the previous content entirely.
```

Dosyaya Ekleme (Append)

Varsayılan davranış (üzerine yazma) yerine mevcut içeriğin SONUNA eklemek için `StandardOpenOption.APPEND` geçilir. Kritik bir nokta: `APPEND` TEK BAŞINA, dosyanın ZATEN VAR olduğunu varsayar -- henüz var olmayan bir dosyaya yalnızca `APPEND` ile yazmaya çalışmak `NoSuchFileException` fırlatır; dosyayı hem oluşturmak hem de eklemek istiyorsanız `StandardOpenOption.CREATE` İLE BİRLİKTE geçilmelidir.

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardOpenOption;

public class AppendToFileExample {
    public static void main(String[] args) throws IOException {
        Path path = Path.of("demo-append.txt");
        Files.writeString(path, "First line");

        // By default Files.writeString() OVERWRITES -- passing
        // StandardOpenOption.APPEND changes that behavior to add onto the
        // END of the existing content instead of replacing it.
        Files.writeString(path, "\nSecond line", StandardOpenOption.APPEND);
        Files.writeString(path, "\nThird line", StandardOpenOption.APPEND);

        System.out.println("Final content after two appends:");
        System.out.println(Files.readString(path));

        // Trying to append to a file that doesn't exist yet fails, UNLESS you
        // also pass StandardOpenOption.CREATE -- APPEND alone assumes the file
        // is already there.
        Path newPath = Path.of("demo-append-new.txt");
        Files.writeString(newPath, "Created via CREATE + APPEND",
                StandardOpenOption.CREATE, StandardOpenOption.APPEND);
        System.out.println("New file via CREATE + APPEND: " + Files.readString(newPath));

        Files.deleteIfExists(path);
        Files.deleteIfExists(newPath);
    }
}
```

Beklenen Çıktı

```
Final content after two appends:
First line
Second line
Third line
New file via CREATE + APPEND: Created via CREATE + APPEND
```

`Files.writeString(path, içerik, StandardOpenOption.APPEND)`'i henüz var olmayan bir dosya için çağırmak `NoSuchFileException` fırlatır -- `APPEND`, `CREATE`'i OTOMATİK OLARAK içermez. İkisini birlikte geçmek (`StandardOpenOption.CREATE, StandardOpenOption.APPEND`), dosya var olsun ya da olmasın güvenle çalışır.

Bir Listeyi Satır Satır Yazmak

`Files.write(path, list)`, bir `List<String>` alır ve her elemanı KENDİ SATIRINA yazar -- satır sonu karakterlerini elle eklemekle (`String.join("\n", list)`) uğraşmaya gerek kalmaz.

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;

public class WriteLinesExample {
    public static void main(String[] args) throws IOException {
        Path path = Path.of("demo-lines.txt");

        // Files.write(path, list) takes a List<String> and writes each
        // element as its own line -- the line-separator handling is done for
        // you, unlike manually joining with "\n".
        List<String> fruits = List.of("Apple", "Banana", "Cherry");
        Files.write(path, fruits);

        System.out.println("File content:");
        System.out.println(Files.readString(path));

        System.out.println("Read back as a List:");
        Files.readAllLines(path).forEach(line -> System.out.println("  " + line));

        Files.deleteIfExists(path);
    }
}
```

Beklenen Çıktı

```
File content:
Apple
Banana
Cherry

Read back as a List:
  Apple
  Banana
  Cherry
```

BufferedWriter ile Yazma

`BufferedWriter`, `BufferedReader`'ın (bkz. "File Reading" dersi) yazma karşılığıdır -- bir `FileWriter`'ı sarar, yazmaları içeride tamponlar. `write()` KENDİ BAŞINA bir satır sonu EKLEMEZ -- bunun için açıkça `newLine()` çağrılmalıdır (platforma doğru satır sonunu kullanır: Linux/macOS'ta `\n`, Windows'ta `\r\n`).

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class BufferedWriterExample {
    public static void main(String[] args) throws IOException {
        Path path = Path.of("demo-buffered-write.txt");

        // BufferedWriter is the classic java.io counterpart to
        // BufferedReader -- it wraps a FileWriter and buffers writes
        // internally. write() does NOT add a line separator by itself;
        // newLine() does that explicitly (and uses the platform's correct
        // separator, "\n" on Linux/macOS, "\r\n" on Windows).
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(path.toFile()))) {
            writer.write("Line 1");
            writer.newLine();
            writer.write("Line 2");
            writer.newLine();
            writer.write("Line 3");
            // No trailing newLine() here -- the file ends right after "Line 3".
        }

        System.out.println("File content:");
        System.out.println(Files.readString(path));

        Files.deleteIfExists(path);
    }
}
```

Beklenen Çıktı

```
File content:
Line 1
Line 2
Line 3
```

> Son satırdan sonra `newLine()` çağırıp çağırmamak SİZİN kararınızdır -- yukarıdaki örnekte üçüncü satırdan sonra `newLine()` ÇAĞRILMADI, bu yüzden dosya `"Line 3"` 'ten hemen sonra, satır sonu OLMADAN biter.

Dosya Kopyalama ve Dizin Yönetimi

`Files.createDirectories()`, bir dizini VE yoldaki tüm eksik üst dizinleri oluşturur (`mkdir -p` gibi) -- dizin zaten varsa hata VERMEZ. `Files.copy()` bir dosyanın içeriğini tek çağrıda kopyalar -- ama VARSAYILAN olarak hedef zaten varsa `FileAlreadyExistsException` fırlatır; `StandardCopyOption.REPLACE_EXISTING` bunun yerine üzerine yazmayı sağlar. Bir dizin ağacını TAMAMEN silmek için önce onu GEZMEK (`Files.walk()`) ve EN DERİNDEKİ (dosyalar, sonra alt dizinler) girişlerden başlayarak silmek gerekir -- boş olmayan bir dizin doğrudan silinemez.

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.util.Comparator;

public class CopyAndDirectoryExample {
    public static void main(String[] args) throws IOException {
        // Files.createDirectories() creates a directory AND any missing
        // parent directories along the way (like "mkdir -p") -- it does
        // NOT fail if the directory already exists.
        Path baseDir = Files.createDirectories(Path.of("demo-output"));
        System.out.println("Created directory: " + baseDir.getFileName());

        Path original = baseDir.resolve("original.txt");
        Files.writeString(original, "Content to be copied.");

        Path copy = baseDir.resolve("copy.txt");
        // Files.copy() copies a file's content in one call -- but by default
        // it throws FileAlreadyExistsException if the destination already
        // exists. StandardCopyOption.REPLACE_EXISTING makes it overwrite
        // instead.
        Files.copy(original, copy, StandardCopyOption.REPLACE_EXISTING);
        System.out.println("Copied content: " + Files.readString(copy));

        // Copying again to the SAME destination without REPLACE_EXISTING
        // would throw -- demonstrating why the option matters.
        try {
            Files.copy(original, copy); // no REPLACE_EXISTING this time
            System.out.println("unreachable");
        } catch (java.nio.file.FileAlreadyExistsException e) {
            System.out.println("Caught FileAlreadyExistsException without
REPLACE_EXISTING");
        }

        // Cleaning up an entire directory tree requires walking it and
        // deleting from the DEEPEST entries first -- you can't delete a
        // non-empty directory. Files.walk() + Comparator.reverseOrder()
        // (files/subdirectories before their parent) is the standard pattern
        // for this.
        try (var paths = Files.walk(baseDir)) {
            paths.sorted(Comparator.reverseOrder())
                    .forEach(p -> {
                        try {
                            Files.delete(p);
                        } catch (IOException e) {
                            System.out.println("Failed to delete: " + p);
                        }
                    });
        }
        System.out.println("Directory exists after cleanup? " + Files.exists(baseDir));
    }
}
```

> `Files.walk(dizin).sorted(Comparator.reverseOrder()).forEach(Files::delete)` deseni, bir dizin ağacını silmenin STANDART yoludur -- `reverseOrder()` KRİTİKTİR, çünkü bir dizin, İÇİNDEKİ dosyalar/alt dizinler silinmeden silinemez. `Comparator.reverseOrder()` olmadan (doğal, sığdan derine sıralamayla) silme denemesi, boş olmayan dizinler için istisna fırlatır.

CSV Dosyası Yazmak

Yapılandırılmış bir metin dosyası (CSV gibi) yazmak, string birleştirme temellerini (bkz. "String" dersi) File I/O ile birleştirir: `String.join(",", dizi)` her satırı oluşturur, tüm satırlar bir `StringBuilder`'da toplanır, ve TEK bir `Files.writeString()` çağrısıyla diske yazılır -- her satır için ayrı bir yazma çağrısı yapmaktan daha VERİMLİDİR.

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class WriteCsvExample {
    public static void main(String[] args) throws IOException {
        Path path = Path.of("demo-data.csv");

        String[] headers = {"Name", "Age", "City"};
        String[][] rows = {
                {"Alice", "30", "London"},
                {"Bob", "25", "Paris"},
                {"Charlie", "35", "Tokyo"}
        };

        writeCsv(path, headers, rows);

        System.out.println("CSV file content:");
        System.out.println(Files.readString(path));

        Files.deleteIfExists(path);
    }

    private static void writeCsv(Path path, String[] headers, String[][] rows) throws
IOException {
        // String.join(",", array) turns an array into a single comma-joined
        // line -- the same helper used for the header row and every data row.
        StringBuilder csv = new StringBuilder();
        csv.append(String.join(",", headers)).append('\n');
        for (String[] row : rows) {
            csv.append(String.join(",", row)).append('\n');
        }

        // Writing the fully-built content in ONE Files.writeString() call is
        // more efficient than writing line by line -- fewer individual I/O
        // operations against the filesystem.
        Files.writeString(path, csv.toString());
    }
}
```

Beklenen Çıktı

```
CSV file content:
Name,Age,City
Alice,30,London
Bob,25,Paris
Charlie,35,Tokyo
```

Best Practices

- **Tek seferlik, basit yazmalar için `Files.writeString()` / `Files.write()` kullanın** -- kısa ve okunabilir; çok sayıda küçük yazma biriktiriyorsanız `BufferedWriter` tercih edin.
- **`APPEND` kullanırken dosyanın var olup olmadığından emin değilseniz `StandardOpenOption.CREATE`'i de birlikte geçin** -- yalnızca `APPEND`, dosya yoksa `NoSuchFileException` fırlatır.
- **`Files.copy()` çağırırken hedefin üzerine yazılmasını istiyorsanız her zaman `StandardCopyOption.REPLACE_EXISTING` ekleyin** -- aksi halde hedef zaten varsa istisna alırsınız.
- **Bir dizin ağacını silerken `Files.walk().sorted(Comparator.reverseOrder())` desenini kullanın** -- doğal sırayla silmeye çalışmak boş olmayan dizinlerde başarısız olur.

Yaygın Hatalar

- **`StandardOpenOption.APPEND`'i henüz var olmayan bir dosyada kullanıp `NoSuchFileException` almak.** `CREATE`'i de birlikte geçmek gerekir.
- **`Files.copy()`'yi `REPLACE_EXISTING` olmadan çağırıp hedef zaten varken `FileAlreadyExistsException` almak.** Üzerine yazma isteniyorsa bu seçenek eklenmeli.
- **Bir dizin ağacını doğal sırayla (sığdan derine) silmeye çalışıp boş olmayan dizin hatası almak.** `Comparator.reverseOrder()` ile derinden sığa doğru silinmeli.
- **`BufferedWriter.write()`'ın otomatik satır sonu ekleyeceğini varsaymak.** `write()` yalnızca metni yazar -- satır sonu için ayrıca `newLine()` çağrılmalı.

Özet, Cheat Sheet ve Terimler Sözlüğü

Dosyaya yazmak için de iki API vardır: modern `Files.writeString()` / `Files.write()` (tek satırlık, `StandardOpenOption` ile davranışı özelleştirilebilir) ve klasik `BufferedWriter` + `FileWriter` (satır satır yazma, `newLine()` ile açık satır sonu kontrolü). `Files.copy()` dosya kopyalar (`REPLACE_EXISTING` gerekebilir), `Files.createDirectories()` dizin oluşturur, ve bir dizin ağacını silmek `Files.walk()` + ters sıralama gerektirir.

Hızlı referans:

```
Files.writeString(path, "content");                                  // oluştur/üzerine yaz
Files.writeString(path, "\nmore", StandardOpenOption.APPEND);          // sona ekle (dosya
VAR olmalı)
Files.writeString(path, "content",
        StandardOpenOption.CREATE, StandardOpenOption.APPEND);          // yoksa oluştur +
ekle
Files.write(path, listOfStrings);                                        // List<String> -
> satır satır

try (BufferedWriter w = new BufferedWriter(new FileWriter(path.toFile()))) {  // klasik
yazma
    w.write("line");
    w.newLine();                                                                  // AÇIKÇA
satır sonu
}

Files.createDirectories(dirPath);                                              // dizin (+
üst dizinler)
Files.copy(source, dest, StandardCopyOption.REPLACE_EXISTING);                 // kopyala,
üzerine yaz

Files.walk(dirPath)
    .sorted(Comparator.reverseOrder())                                          // derinden
sığa
    .forEach(p -> { try { Files.delete(p); } catch (IOException e) {} });         // ağacı
sil
```

Terimler Sözlüğü

BufferedWriter — Bir yazma hedefini (örn. `FileWriter`) saran, yazmaları tamponlayan klasik `java.io` sınıfı.

StandardOpenOption — `Files.writeString()` / `Files.write()`'ın davranışını (üzerine yazma, ekleme, oluşturma) özelleştiren enum (`APPEND`, `CREATE` vb.).

StandardCopyOption — `Files.copy()`'nin davranışını özelleştiren enum (örn. `REPLACE_EXISTING`).

Files.walk() — Bir dizin ağacındaki TÜM dosya/alt dizinleri gezen, bir `Stream<Path>` döndüren metot.

FileAlreadyExistsException — `Files.copy()` gibi bir metot, üzerine yazma izni verilmeden ZATEN VAR olan bir hedefe yazmaya çalıştığında fırlatılan istisna.