Αντικειμενοστραφής Προγραμματισμός

Β' εξάμηνο – Τμήμα Πληροφορικής – Ιόνιο Πανεπιστήμιο

Ενότητα 9

Java I/O & Streams

Δημήτρης Ρίγγας

Μηχανικός Η/Υ & Πληροφορικής, MSc, PhD

Ε.Δι.Π. Τμήματος Πληροφορικής – Ιόνιο Παν/μιο

riggas@ionio.gr

Java I/O & Streams

Πώς διαβάζουμε, γράφουμε και ανταλλάσσουμε δεδομένα —
χωρίς να τα χάνουμε κάθε φορά που τελειώνει το πρόγραμμα.

Γιατί χρειαζόμαστε αρχεία;

Τα δεδομένα στη μνήμη (μεταβλητές, αντικείμενα) είναι **εφήμερα** — εξαφανίζονται μόλις τελειώσει η εκτέλεση.

Μνήμη RAM (μεταβλητές)

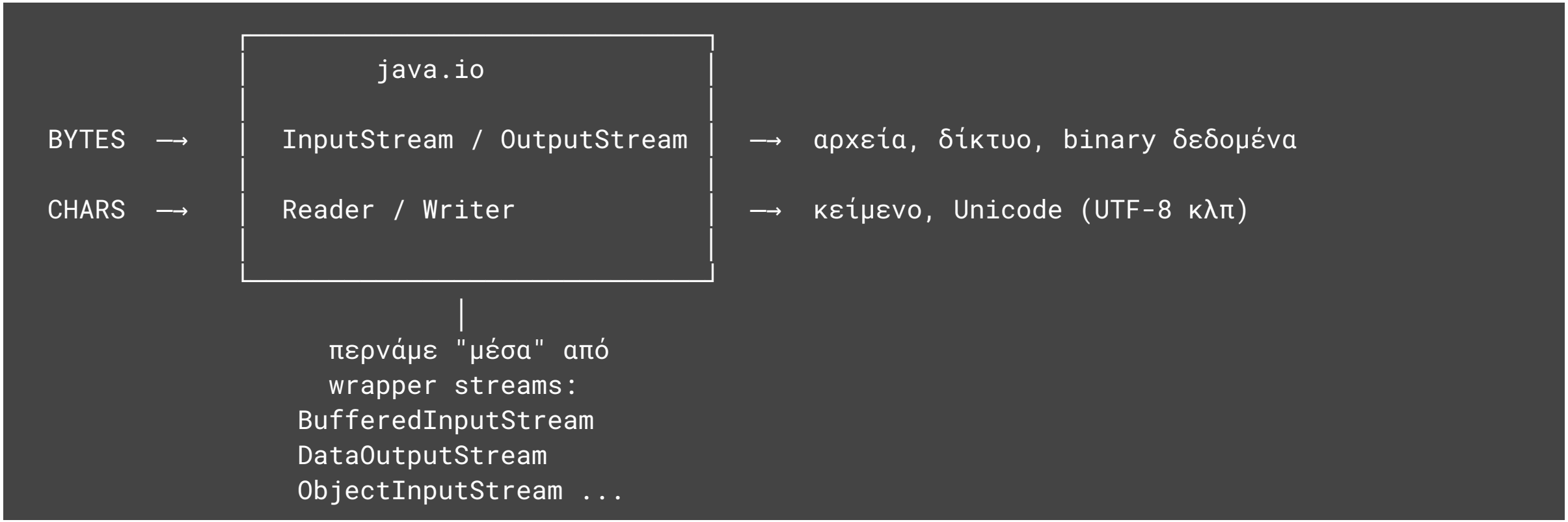
- Πολύ γρήγορη
- Χάνεται στο κλείσιμο
- Δεν μοιράζεται εύκολα

Αρχεία / Δίσκος

- Πιο αργά
- Παραμένουν
- Μοιράζονται μεταξύ προγραμμάτων

Η Java βλέπει κάθε πηγή ή προορισμό δεδομένων ως **ρεύμα (stream)**:
√ αρχεία,
√ δίκτυο,
√ keyboard
— ίδια αντιμετώπιση.

Ο χάρτης των Streams



√ Η ιδέα: **stack από wrappers** — κάθε επίπεδο προσθέτει δυνατότητα.

Byte Streams — η βάση

Διαβάζουν/γράφουν **ακατέργαστα bytes** (8 bits). Χρήσιμα για binary αρχεία.

Java 7+ — try-with-resources

```
// Κλείνει αυτόματα – χωρίς finally!
try (var in  = new FileInputStream("src.bin");
     var out = new FileOutputStream("dst.bin")) {

  byte[] buf = new byte[8192];
  int n;
  while ((n = in.read(buf)) != -1) {
    out.write(buf, 0, n);
  }
} // in και out κλείνουν εδώ
```

Java 9+ — transferTo()

```
// Μία γραμμή – ίδιο αποτέλεσμα!
try (var in  = new FileInputStream("src.bin");
     var out = new FileOutputStream("dst.bin")) {

  in.transferTo(out);

}
```

√ Το `transferTo()` χρησιμοποιεί internally buffer — εξίσου αποδοτικό, πολύ πιο ευανάγνωστο.

Character Streams — κείμενο & Unicode

Για αρχεία κειμένου. Χειρίζονται **Unicode** σωστά (UTF-8, κλπ).

```
// FileReader / FileWriter: για απλή ανάγνωση χαρακτήρα-χαρακτήρα
try (var in  = new FileReader("input.txt",  StandardCharsets.UTF_8);
     var out = new FileWriter("output.txt", StandardCharsets.UTF_8)) {

    int ch;
    while ((ch = in.read()) != -1) {
        out.write(Character.toUpperCase(ch));
    }
}
```

✓ Πάντα ορίζουμε **encoding** ρητά (π.χ. `StandardCharsets.UTF_8`) — αλλιώς εξαρτάται από το λειτουργικό σύστημα.. *ελληνικά!;;*

Buffered I/O — κάθε *κλήση* έχει σημασία

Χωρίς buffer: κάθε `read()` / `write()` → **κλήση στο λειτουργικό σύστημα** → αργό.
Με buffer: λίγες κλήσεις, μεγάλα κομμάτια → **πολύ πιο γρήγορο**.

Αρχείο 16 MB

CopyBytes: 52.3 sec

Buffered: 0.27 sec

```
// Απλά "τυλίγει" το προηγούμενο stream:
try (
  var in  = new BufferedInputStream(
              new FileInputStream("a.bin"));
  var out = new BufferedOutputStream(
              new FileOutputStream("b.bin"))
) {
    in.transferTo(out); // Java 9+
}
```

√ Η μέθοδος `flush()` αναγκάζει τον buffer να γράψει τώρα — χρήσιμο σε network streams ή logging.

Ανάγνωση γραμμών με BufferedReader

```
// ❌ Pre Java 11 – verbose
var sb = new StringBuilder();
try (var br = new BufferedReader(new FileReader("note.txt", StandardCharsets.UTF_8))) {
    String line;
    while ((line = br.readLine()) != null) sb.append(line).append('\n');
}
String content = sb.toString();
```

Java 11+ — writeString

```
Path path = Path.of("note.txt");

Files.writeString(
    path,
    "Γεια σου, Java 11!\n",
    StandardCharsets.UTF_8
);
// Δημιουργεί ή αντικαθιστά το αρχείο
// Κλείνει αυτόματα – χωρίς try!
```

Java 11+ — readString

```
Path path = Path.of("note.txt");

String content = Files.readString(
    path,
    StandardCharsets.UTF_8
);

System.out.println(content);
// → "Γεια σου, Java 11!"
```

✓ Μόνο για αρχεία που χωράνε στη μνήμη (config, JSON).
✓ Για μεγάλα αρχεία `Files.lines()` ή `BufferedReader` — διαβάζουν streaming.

Data Streams — binary εγγραφές

Αποθηκεύουν primitive types (`int`, `double`, `boolean`...) σε **binary μορφή**.
Γρήγορα, μικρά σε μέγεθος — αλλά **μη αναγνώσιμα** από άνθρωπο.

```
// Εγγραφή
try (var out = new DataOutputStream(
        new BufferedOutputStream(new FileOutputStream("stocks.dat")))) {
    out.writeUTF("AAPL");
    out.writeDouble(178.50);
    out.writeInt(100);
}

// Ανάγνωση – ίδια σειρά, αλλιώς corruption!
try (var in = new DataInputStream(
        new BufferedInputStream(new FileInputStream("stocks.dat")))) {
    try {
        while (true) {
            System.out.printf("%s: %.2f x %d%n",
                in.readUTF(), in.readDouble(), in.readInt());
        }
    } catch (EOFException e) { /* φυσιολογικό τέλος αρχείου */ }
}
```

Object Streams & Serialization

Αποθηκεύουν ολόκληρα **αντικείμενα** — η Java κάνει τη μετατροπή μόνη της.

```
// Η κλάση πρέπει να implement-άρει Serializable:
public class Stock implements Serializable {
    @Serial  // ← Java 14+: annotation για ασφάλεια
    private static final long serialVersionUID = 1L;
    private String symbol;
    private double price;
    // ...
}
```

```
// Εγγραφή αντικειμένου
try (var out = new ObjectOutputStream(new FileOutputStream("stock.obj"))) {
    out.writeObject(new Stock("AAPL", 178.50));
}

// Ανάγνωση — απαιτεί cast
try (var in = new ObjectInputStream(new FileInputStream("stock.obj"))) {
    Stock s = (Stock) in.readObject();
}
```

try-with-resources

Πρόβλημα: αν ξεχαστεί ένα `close()` ή πεταχτεί exception → resource leak.

```
// ❌ Παλιά, επίπονη προσέγγιση (pre-Java 7):
FileInputStream in = null;
try {
    in = new FileInputStream("file.txt");
    // ...
} finally {
    if (in != null) in.close();  // πρέπει να γίνει για κάθε resource!
}
```

```
// ✅ Java 7+ – αυτόματο κλείσιμο:
try (var in  = new FileInputStream("source.txt");
     var out = new FileWriter("dest.txt")) {
    // Κλείνουν αντίστροφα: out → in
}
// Δουλεύει για οτιδήποτε implements AutoCloseable
```

√ Κλείνουν **αντίστροφα** από τη σειρά δήλωσης — το τελευταίο ανοιχτό κλείνει πρώτο.

java.nio.file — σύγχρονο file API

Εισήχθη στη **Java 7** και εξελίχθηκε σημαντικά έως **Java 11+**.
Αντικαθιστά το παλιό `java.io.File` με πολύ πιο εκφραστικό API.

```
// Java 11+: Path.of()
Path file = Path.of("data", "stocks.json");   // cross-platform paths!
// αντί για: Path file = Path.of("data/stocks.json");

// Metadata
System.out.println(Files.size(file));
System.out.println(Files.isReadable(file));
System.out.println(Files.getLastModifiedTime(file));

// Απλή ανάγνωση / εγγραφή (Java 11+)
String content = Files.readString(file, StandardCharsets.UTF_8);
Files.writeString(file, "Hello!", StandardCharsets.UTF_8);

// Αντιγραφή / Μετακίνηση
Files.copy(file, Path.of("backup.json"), StandardCopyOption.REPLACE_EXISTING);
Files.move(file, Path.of("archive/stocks.json"));
```

Directories & File Listing

```java
// Λίστα περιεχομένων φακέλου:
try (var entries = Files.newDirectoryStream(Path.of("."))) {
    for (Path entry : entries) {
        System.out.println(entry.getFileName());
    }
}

// ✨ Java 8+: Files.list() επιστρέφει Stream<Path>
try (var files = Files.list(Path.of("."))) {
    files
        .filter(p -> p.toString().endsWith(".java"))
        .sorted()
        .forEach(System.out::println);
}

// Recursive walk (Java 8+):
try (var walk = Files.walk(Path.of("src"), 3)) {
    walk.filter(Files::isRegularFile)
        .forEach(System.out::println);
}
```

Scanner & Formatting

Scanner: ανάλυση (parsing) structured κειμένου token-by-token.

```java
// Διαβάζει από αρχείο και ξεχωρίζει αριθμούς από κείμενο:
try (var sc = new Scanner(Path.of("data.txt"), StandardCharsets.UTF_8)) {
    while (sc.hasNext()) {
        if (sc.hasNextDouble()) {
            System.out.println("Αριθμός: " + sc.nextDouble());
        } else {
            System.out.println("Κείμενο: " + sc.next());
        }
    }
}
```

```java
// Formatting: printf-style έξοδος
System.out.printf("%-10s %8.2f%n", "AAPL", 178.50);

// Ή σε αρχείο:
try (var pw = new PrintWriter(new FileWriter("report.txt"))) {
    pw.printf("Σύνολο: %d εγγραφές%n", count);
}
```

JSON — το standard ανταλλαγής δεδομένων

JSON (JavaScript Object Notation): ανθρωποαναγνώσιμο, γλωσσαγνωστικό, παντού.

```
{
  "symbol": "AAPL",
  "name": "Apple Inc.",
  "price": 178.50,
  "tags": ["tech", "nasdaq"],
  "active": true
}
```

Γιατί JSON αντί για Java Serialization;

	Java Serialization	JSON
Ανθρωποαναγνώσιμο	❌	✅
Cross-language	❌	✅
Stable μεταξύ εκδόσεων	⚠️	✅
Security	⚠️ Risky	✅

Jackson — το de facto standard

Το **Jackson** είναι η πιο διαδεδομένη βιβλιοθήκη JSON στο Java ecosystem (Spring Boot, Quarkus, Jakarta EE — όλα το χρησιμοποιούν).

```xml
<!-- Maven dependency -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.17.0</version>
</dependency>
```

```java
public class Stock {
    private String symbol;
    private String name;
    private double price;
    // constructors, getters, setters...
    // ή χρησιμοποίησε record! (βλ. επόμενη διαφάνεια)
}
```

Jackson: Serialization & Deserialization

```
ObjectMapper mapper = new ObjectMapper();

// Αντικείμενο → JSON string
Stock s = new Stock("AAPL", "Apple", 178.50);
String json = mapper.writeValueAsString(s);
// → {"symbol":"AAPL","name":"Apple","price":178.5}

// JSON string → Αντικείμενο
Stock s2 = mapper.readValue(json, Stock.class);

// Αρχείο → Αντικείμενο
Stock s3 = mapper.readValue(Path.of("stock.json").toFile(), Stock.class);

// List → JSON αρχείο
List<Stock> stocks = List.of(new Stock("AAPL","Apple",178.5), ...);
mapper.writerWithDefaultPrettyPrinter()
      .writeValue(new File("stocks.json"), stocks);

// JSON αρχείο → List
List<Stock> loaded = mapper.readValue(
    new File("stocks.json"),
    new TypeReference<List<Stock>>() {});
```

Java Records & JSON Java 16+

Records (Java 16): immutable data classes με ελάχιστο boilerplate.
Δουλεύουν τέλεια με Jackson!

```
// Αντί για κλάση με getters/setters/constructor:
public record Stock(String symbol, String name, double price) {}
```

```
ObjectMapper mapper = new ObjectMapper();

// Serialization
String json = mapper.writeValueAsString(new Stock("AAPL", "Apple", 178.50));

// Deserialization
Stock s = mapper.readValue(json, Stock.class);
```

Διαλειτουργικότητα με Python

Η ομορφιά του JSON: γράφεις από Java, διαβάζεις από Python (ή οτιδήποτε άλλο).

```python
import json

# Διαβάζει το JSON που έγραψε η Java
with open("stocks.json") as f:
    stocks = json.load(f)

for stock in stocks:
    print(f"{stock['name']} ({stock['symbol']}): ${stock['price']:.2f}")
```

```
Apple (AAPL): $178.50
Alphabet (GOOGL): $141.30
...
```

→ Αυτό είναι **interoperability** — ένας τυπικός τρόπος επικοινωνίας ανεξάρτητα από γλώσσα.

Text Blocks & JSON Java 15+

Text Blocks (Java 15+): πολυγραμμικά strings χωρίς escape hell.
Ιδανικό για embedded JSON / SQL / HTML στον κώδικα.

```
// ❌ Πριν (Java 14 και παλιότερα):
String json = "{\n  \"symbol\": \"AAPL\",\n  \"price\": 178.50\n}";

// ✅ Java 15+:
String json = """
        {
          "symbol": "AAPL",
          "price": 178.50
        }
        """;

// Μπορείς να κάνεις κατευθείαν parse:
Stock s = mapper.readValue(json, Stock.class);
```

😭 Ας τα βάλουμε σε κουτάκια: τι χρησιμοποιούμε & πότε;

Ανάγκη	Χρήση
Αντιγραφή binary αρχείου	`FileInputStream` + `transferTo()`
Ανάγνωση κειμένου γραμμή-γραμμή	`BufferedReader` / `Files.lines()`
Απλή ανάγνωση/εγγραφή κειμένου	`Files.readString()` / `writeString()` ✨
Αποθήκευση primitive types	`DataOutputStream`
Long-term αποθήκευση αντικειμένων	JSON (Jackson)
JVM-to-JVM μεταφορά (short-term)	`ObjectOutputStream`
Δουλειά με paths/directories	`java.nio.file.Path` + `Files`
Parsing structured κειμένου	`Scanner`

Βασικές αρχές

1. Κλείνουμε τους πόρους

```
try (var r = new FileReader("f.txt")) { ... }  // αυτόματο κλείσιμο
```

2. Buffering για απόδοση

```
new BufferedReader(new FileReader(...))  // ή Files.lines()
```

3. Encoding ρητά

```
new FileReader(file, StandardCharsets.UTF_8)  // ποτέ χωρίς charset!
```

4. Προτιμούμε modern API

- `Path.of()` αντί `Paths.get()` *(Java 11+)*
- `Files.readString()` / `writeString()` *(Java 11+)*
- `InputStream.transferTo()` *(Java 9+)*
- Records για data classes *(Java 16+)*

📚 Πηγές & Περαιτέρω Μελέτη

- **Oracle Java Tutorials — Essential I/O**
  https://docs.oracle.com/javase/tutorial/essential/io/
- **Jackson Documentation**
  https://github.com/FasterXML/jackson-databind
- **Java NIO.2 Guide** (Baeldung)
  https://www.baeldung.com/java-nio-2-file-api
- **Java 17 API Docs** — `java.io`, `java.nio.file`, `java.util.Scanner`
  https://docs.oracle.com/en/java/api/java.base/
- **JEP 395** — Records (Java 16)
  https://openjdk.org/jeps/395