**Què és la serialització?**

* És el procés de **guardar un objecte Java sencer** (amb tots els seus atributs) en un fitxer.
* Després pots **recuperar-lo** i tindrà el mateix estat que quan el vas escriure.
* Exemple: tens una classe Alumne amb nom i nota. Amb serialització, pots escriure directament un Alumne al fitxer, i després llegir-lo sense haver de parsejar text o binari manualment.

**1. Preparar una classe Alumne**

Perquè un objecte sigui “serialitzable”, la classe ha d’implementar la interfície Serializable.

package Paquet1;

import java.io.Serializable;

public class Alumne implements Serializable {

private static final long serialVersionUID = 1L;

private String nom;

private double nota;

public Alumne(String nom, double nota) {

this.nom = nom;

this.nota = nota;

}

@Override

public String toString() {

return "Alumne{" +

"nom='" + nom + '\'' +

", nota=" + nota +

'}';

}

}

**2. Escriure i llegir objectes serialitzats**

Ara fem una classe SerialitzacioExemple.java per provar-ho:

package Paquet1;

import java.io.\*;

public class SerialitzacioExemple {

public static void main(String[] args) {

File fitxer = new File("alumnes.dat");

// --- Escriure objectes ---

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fitxer))) {

Alumne a1 = new Alumne("Anna", 8.5);

Alumne a2 = new Alumne("Bernat", 7.2);

oos.writeObject(a1);

oos.writeObject(a2);

System.out.println("S'han escrit alumnes a: " + fitxer.getAbsolutePath());

} catch (IOException e) {

e.printStackTrace();

}

// --- Llegir objectes ---

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fitxer))) {

while (true) {

try {

Alumne alumne = (Alumne) ois.readObject();

System.out.println("Llegit: " + alumne);

} catch (EOFException eof) {

// Quan arribem al final del fitxer, sortim del bucle

break;

}

}

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

**Què fa aquest codi ??**

1. **Escriu** dos objectes Alumne dins alumnes.dat.
2. **Llegeix** el fitxer i recupera els objectes.
3. Els imprimeix amb el toString().

Sortida esperada:

Texto

El contenido generado por IA puede ser incorrecto.

**Detall important a tenir en compte.**

Si vols **afegir** més objectes a un fitxer serialitzat ja existent, no pots usar directament new ObjectOutputStream(new FileOutputStream(fitxer, true)) perquè Java reescriu la capçalera del fitxer i dóna error.

La solució és crear una subclasse d’ObjectOutputStream que no torni a escriure la capçalera quan fem *append*. (Això ho veurem en el següent pas).

**Següent Pas**:

**Substituir** la classe Alumne.java :

* Ves al fitxer Alumne.java que ja tens.
* Esborra el contingut antic.
* Copia i enganxa la versió nova

package Paquet1;

import java.io.Serializable;

public class Alumne implements Serializable {

private static final long serialVersionUID = 1L;

private String nom;

private double nota;

public Alumne(String nom, double nota) {

this.nom = nom;

this.nota = nota;

}

public String getNom() {

return nom;

}

public double getNota() {

return nota;

}

@Override

public String toString() {

return "Alumne{" +

"nom='" + nom + '\'' +

", nota=" + nota +

'}';

}

}

* Desa (Ctrl+S).
* Torna a executar SerialitzacioExemple.java.

Si tot és correcte, hauríes de veure a l’Output alguna cosa com:

**Texto

El contenido generado por IA puede ser incorrecto.**

**Anem a veure amb calma què hem fet quan vam ampliar la classe Alumne i per què.**

* **La primera versió (la “mínima”)**

**Quan vam fer el primer exemple de serialització, la classe Alumne era molt senzilla:**

**public class Alumne implements Serializable {**

**public String nom;**

**public double nota;**

**public Alumne(String nom, double nota) {**

**this.nom = nom;**

**this.nota = nota;**

**}**

**public String toString() {**

**return nom + " (" + nota + ")";**

**}**

**}**

Això ja funcionava perquè:

* implements Serializable permet guardar i recuperar objectes.
* Tenia un constructor per crear alumnes.
* Tenia un toString() per imprimir-los.

***La versió ampliada:***

La versió nova és més “professional” i “robusta”:

**public class Alumne implements Serializable {**

**private static final long serialVersionUID = 1L;**

**private String nom;**

**private double nota;**

**public Alumne(String nom, double nota) {**

**this.nom = nom;**

**this.nota = nota;**

**}**

**public String getNom() { return nom; }**

**public double getNota() { return nota; }**

**@Override**

**public String toString() {**

**return "Alumne{" +**

**"nom='" + nom + '\'' +**

**", nota=" + nota +**

**'}';**

**}**

**}**

**Canvis principals respecte a la clase antiga:**

1. **Atributs privats (private)**

* **Abans eren públics (public) → qualsevol part del programa podia canviar-los directament.**
* **Ara són privats → només la mateixa classe els pot modificar.**
* **És una bona pràctica de programació orientada a objectes: encapsulació.**

1. **Getters (getNom(), getNota())**
   * **Com que els atributs són privats, si un altre codi vol consultar-los ha de fer-ho amb aquests mètodes.**
   * **Exemple: a.getNom() en lloc de a.nom.**
2. **serialVersionUID**

* **És un número que Java fa servir per saber si la versió de la classe ha canviat.**
* **Amb private static final long serialVersionUID = 1L; assegurem compatibilitat entre versions de la classe quan serialitzem i llegim objectes.**

1. **toString() més detallat**
   * **En lloc de només "Anna (8.5)", ara imprimeix en format Alumne{nom='Anna', nota=8.5}.**
   * **És només una millora estètica.**

**Per què l’hem ampliat?**

* **Per acostumar-nos a bones pràctiques (atributs privats + getters).**
* **Per fer la classe més fàcil de mantenir i segura.**
* **Perquè després, quan fem gestors més complexos (serialització, menús, bases de dades), tindrem un model d’objecte ben fet.**

**El canvi principal és d’estil i bones pràctiques, no de funcionalitat.**

**El següent pas després de tenir la classe Alumne ben definida és el Gestor d’alumnes amb serialització, que funciona com el que ja vam fer amb CSV, però guardant objectes sencers dins un fitxer .dat.**

**Recordatori:**

* **Alumne.java → ja el tens, amb atributs privats, getters i Serializable.**
* **Per fer append (afegir alumnes al mateix fitxer sense corrompre’l) necessitem una classe especial: AppendableObjectOutputStream.**

**Classe auxiliar: AppendableObjectOutputStream.java**

**package Paquet1;**

**import java.io.IOException;**

**import java.io.ObjectOutputStream;**

**import java.io.OutputStream;**

**public class AppendableObjectOutputStream extends ObjectOutputStream {**

**public AppendableObjectOutputStream(OutputStream out) throws IOException {**

**super(out);**

**}**

**@Override**

**protected void writeStreamHeader() throws IOException {**

**// No escrivim la capçalera quan fem append**

**reset();**

**}**

**}**

**Classe principal: GestorAlumnesSerialitzacio.java**

**package Paquet1;**

**import java.io.\*;**

**import java.util.Scanner;**

**public class GestorAlumnesSerialitzacio {**

**private static final File FITXER = new File("alumnes.dat");**

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

**Scanner sc = new Scanner(System.in);**

**int opcio;**

**do {**

**System.out.println("\n--- GESTOR D'ALUMNES (SERIALITZACIÓ) ---");**

**System.out.println("1. Afegir alumne");**

**System.out.println("2. Llistar alumnes");**

**System.out.println("0. Sortir");**

**System.out.print("Opció: ");**

**opcio = Integer.parseInt(sc.nextLine());**

**switch (opcio) {**

**case 1 -> afegirAlumne(sc);**

**case 2 -> llistarAlumnes();**

**case 0 -> System.out.println("Sortint...");**

**default -> System.out.println("Opció no vàlida");**

**}**

**} while (opcio != 0);**

**sc.close();**

**}**

**private static void afegirAlumne(Scanner sc) {**

**System.out.print("Nom: ");**

**String nom = sc.nextLine();**

**System.out.print("Nota: ");**

**double nota = Double.parseDouble(sc.nextLine());**

**try {**

**boolean existeix = FITXER.exists();**

**ObjectOutputStream oos;**

**if (existeix) {**

**// Append sense reescriure capçalera**

**oos = new AppendableObjectOutputStream(new FileOutputStream(FITXER, true));**

**} else {**

**// Primer cop: capçalera normal**

**oos = new ObjectOutputStream(new FileOutputStream(FITXER));**

**}**

**oos.writeObject(new Alumne(nom, nota));**

**oos.close();**

**System.out.println("Alumne afegit correctament!");**

**} catch (IOException e) {**

**System.err.println("Error escrivint: " + e.getMessage());**

**}**

**}**

**private static void llistarAlumnes() {**

**if (!FITXER.exists()) {**

**System.out.println("Encara no hi ha alumnes.");**

**return;**

**}**

**try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FITXER))) {**

**System.out.println("\n--- LLISTA D'ALUMNES ---");**

**while (true) {**

**try {**

**Alumne a = (Alumne) ois.readObject();**

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

**} catch (EOFException eof) {**

**break; // Final del fitxer**

**}**

**}**

**} catch (IOException | ClassNotFoundException e) {**

**System.err.println("Error llegint: " + e.getMessage());**

**}**

**}**

**}**

**Com provar-ho**

1. **Executa GestorAlumnesSerialitzacio.java amb Run File.**
2. **Al menú:**

**Opció 1 → escriu un alumne (Nom i Nota).**

**Opció 2 → mostra tots els alumnes guardats.**

**Opció 0 → sortir.**

1. **Pots executar el programa tantes vegades com vulguis → els alumnes es van guardant a alumnes.dat.**

**Exemple d’ús**

**Texto

El contenido generado por IA puede ser incorrecto.**

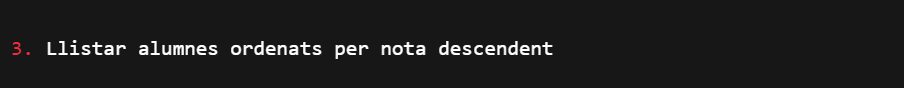
**Exercicis proposats — Gestor d’alumnes amb serialització (versió avançada)**

**Objectius:**

* **Treballar lectura i escriptura seqüencial d’objectes.**
* **Practicar llistes en memòria per ordenar, filtrar i modificar dades.**
* **Veure casos reals: actualització, eliminació, backup i integració amb CSV.**
* **Fer-los pensar en com funciona la persistència real en aplicacions.**

**Exercici 1 — Llistar ordenats per nota descendent**

**Afegeix una opció al menú:**

****

**Ha de llegir tots els alumnes i mostrar només els que tenen nota ≥ 5.**

**Exercici 2 — Millor i pitjor alumne**

**Nova opció:**

****

**Mostra només l’alumne amb la nota més alta i l’alumne amb la nota més baixa.**

**Exercici 3 — Estadístiques**

**Nova opció:**

****

**Calcula i mostra:**

* **Mitjana de totes les notes**
* **Nota màxima**
* **Nota mínima**

**Exercici 4 — Actualitzar nota**

**Nova opció:**

****

**Demana el nom d’un alumne i una nova nota.**

* **Carrega tots els alumnes a memòria.**
* **Actualitza la nota de l’alumne.**
* **Reescriu el fitxer .dat amb els nous valors.**

**Exercici 5 — Eliminar per suspès**

**Nova opció:**

****

**Llegeix tots els alumnes, elimina els que tenen nota < 5, i torna a escriure el fitxer amb la resta.**

**Exercici 6 — Importar des de CSV**

**Nova opció:**

****

**Llegeix un fitxer alumnes.csv amb format Nom,Cognom,Edat,Nota, crea objectes Alumne i els afegeix al fitxer de serialització.**

**Exercici 7 (avançat) — Backup i restauració**

**Nova opció:**

****

**Copia el fitxer alumnes.dat a backup\_alumnes.dat.**

**Exercici 8 Opció adicional:**

****

**Substitueix alumnes.dat pel fitxer de backup.**