Employee Data Serializer

File Handling and Serialization in Java - Project Documentation

Project Overview

This Java application demonstrates **file handling** and **object serialization** by reading employee data from a text file, serializing it to a binary file, deserializing it back to objects, and displaying the results. The project showcases core Java I/O operations essential for enterprise applications and technical interviews.

Objective

Create a complete workflow that:

1. Reads structured employee data from CSV-format text file
2. Converts text data into Java objects
3. Serializes objects to binary format for persistent storage
4. Deserializes binary data back to live objects
5. Displays results in formatted console output

Key Technologies Demonstrated

- **Java File I/O**: BufferedReader, FileReader
- **Object Serialization**: ObjectOutputStream, ObjectInputStream
- **Serializable Interface**: Object persistence mechanism
- **Exception Handling**: Production-ready error management
- **Collections Framework**: ArrayList<Employee>
- **Date Handling**: java.util.Date

Core Components

1. Employee.java (Data Model)

Purpose: Serializable POJO representing employee entity

Key Features:

- Implements Serializable interface with serialVersionUID
- Private fields with public getters/setters
- toString() method for formatted display
- Default constructor for deserialization support

Fields:

Field	Type	Description
name	String	Employee full name
id	int	Unique employee ID
salary	double	Annual salary
hireDate	Date	Date of hire

2. MainApp.java (Application Logic)

Purpose: Orchestrates complete file handling and serialization workflow

Key Methods:

Method	Responsibility	Exception Handling
readEmployeesFromTextFile()	CSV parsing from text file	FileNotFoundException, IOException, NumberFormatException
serializeEmployees()	Object → Binary conversion	IOException
deserializeEmployees()	Binary → Object conversion	ClassNotFoundException, IOException

Method	Responsibility	Exception Handling
displayEmployees()	Formatted console output	None

Error Handling Strategy

- **FileNotFound**: User-friendly message with solution
- **Parse Errors**: Skip invalid lines, continue processing
- **Serialization Errors**: Detailed IOException messages
- **ClassNotFound**: Handles class version mismatches
- **Resource Management**: try-with-resources ensures streams clos

Technical Specifications

- **Java Version**: 8+ (uses try-with-resources)
- **Memory**: Minimal (serialization streams objects directly)
- **Performance**: O(n) read/write time complexity
- **Thread Safety**: Single-threaded console application
- **File Format**: Platform-independent binary serialization

Use Cases

1. **Data Migration**: Text → Binary format conversion
2. **Application State Persistence**: Save/load application data
3. **Cache Implementation**: Object caching to disk
4. **Interview Demonstration**: File I/O + Serialization concepts
5. **Backup System**: Serializable object backup/restore

Limitations & Future Enhancements

- **Date Format**: Simple Date constructor (extend with SimpleDateFormat)
- **File Locking**: No concurrent access protection
- **Validation**: Basic field validation only
- **Compression**: Raw serialization (add GZIP compression)

- **Encryption**: Plain binary (add cryptography)

Learning Outcomes

- Master Java NIO file operations
- Understand object serialization lifecycle
- Implement robust exception handling
- Apply collections framework effectively
- Design production-ready console applications