File Handling

Overview

In this lab, you will add file-handling capabilities to an existing application, to read/write company and employee data to a file.

Source folders

Student project: `StudentFileHandling`
Solution project: `SolutionFileHandling`

Roadmap

There are 3 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Reading/writing data to a text file
2. Serializing objects
3. Reading/writing data to a binary file

Familiarization with code base

Open the student project, and expand the `student.filehandling` package. This package contains the "application logic" classes for the system.

Take a quick look at the `Company` and `Employee` classes. There are no great surprises; these classes are essentially the same as in an earlier lab. Now take a look at the `Main` class; the `main()` method is similar to previous labs, but we've added some new methods:

- `loadCompany()` – This method is complete. It reads company/employee data from a text file, and returns a `Company` object (with its associated `Employee` objects). The method delegates the file-reading task to another class named `CompanyPersister`, which you'll implement in this exercise.
- `saveCompany()` – This method is complete. It writes a `Company` object (and its associated `Employee` objects) to a text file. The method delegates the file-writing task to the `CompanyPersister` class.
- `deserializeCompany()` – This method is incomplete. You will implement this method in Exercise 2, to deserialize a `Company` object (and its associated `Employee` objects) from a Java serialization file.
- `serializeCompany()` – This method is incomplete. You will implement this method in Exercise 2, to serialize a `Company` object (and its associated `Employee` objects) to a Java serialization file.

Now expand the `student.filehandling.persistence` package. This package contains the "persistence logic" classes/interfaces for the system:

- `PersistableToTextFile<T>` – This is a generic interface. It specifies 2 methods, relating to reading and writing an object (of some type `T`) to a text file. (Remember, `BufferedReader` and `BufferedWriter` are used for text I/O in Java).
- `EmployeePersister` – Implements `PersistableToTextFile<Employee>`, to read/write an `Employee` to a text file. You will implement this class in this exercise.
- `CompanyPersister` – Implements `PersistableToTextFile<Company>`, to read/write a `Company` to a text file. You will implement this class in this exercise.

Exercise 1: Reading/writing data to a text file

Time to write some code!

- In `EmployeePersister`, write the `readFromTextFile()` and `writeToTextFile()` methods as per the comments in the code, to read/write an `Employee` to a text file.
- In `CompanyPersister`, write the `readFromTextFile()` and `writeToTextFile()` methods as per the comments in the code, to read/write a `Company` to a text file.

Back in `main()`, uncomment the statements to load/save data at application start-up and shut-down. Run the application, and verify that it successfully loads and saves data to a text file.

Exercise 2: Serialization

Enhance the `Employee` and `Company` classes, so that they are serializable. Then add code in `Main.java`, to implement the `deserializeCompany()` and `serializeCompany()` methods.

In `main()`, uncomment the statements to deserialize/serialize data at application start-up and shut-down. Run the application, and verify that it successfully deserializes and serializes data to a Java serialization file.

Exercise 3 (If time permits): Reading/writing data to a binary file

In the `student.filehandling.persistence` package, define a new interface named `PersistableToBinFile<T>`. Define two methods in the interface:

- `readFromBinFile()` – Takes a `BufferedInputStream` object as a parameter, representing a binary input stream. Returns a `T`, i.e. an object read in from the stream.
- `writeToBinFile()` – Takes two parameters: a `BufferedOutputStream` representing a binary output stream, and a `T` representing the object to write out.

Enhance the `CompanyPersister` and `EmployeePersister` classes, so that they implement your new interface. In other words, add binary file I/O support to these two classes.

Finally, enhance `Main.java` so that it allows the user to load/save binary data (hint: write methods similar to `loadCompany()` and `saveCompany()`, to set up `BufferedInputStream` and `BufferedOutputStream` objects for binary I/O). Call these new methods from `main()` at application start-up and shut-down.