This document contains homework tasks for Working with files in Java. NOTE: Don't forget using try-with-resources for all home tasks.

**Task 1 - Serialization**

**Cost**: 1 point.

Add the ability to serialize / deserialize for the hierarchy of objects from another homework. However, all numerical fields should not be serialized. Provide a test code in which you create several instances of various classes of this hierarchy and successfully complete the serialization/deserialization operation.

**Task 2 - DiskAnalyzer**

**Cost**: 1 point.

Write the DiskAnalyzer utility command line, which accepts a path to the input (for example, C: ) and a function number, correctly processes incorrect paths or function numbers. The utility outputs the results of its work to a file. The program should work for the C: drive of your working machine.

Required functionality:

1. Search for the file name with the maximum number of letters ‘s’ in the name, display the path to it.
2. Print Top-5 largest files (by size in bytes).
3. The average file size in the specified directory or any its subdirectory.
4. The number of files and folders, divided by the first letters of the alphabet (for example, 100,000 files and 200 folders begin with the letter A).

**Task 3 - FastFileMover**

**Cost**: 1 point.

Write several versions of the FastFileMover utility, which moves a file from one directory to another directory. It takes both file paths as a command line parameters. All exceptions must be handled correctly.

Required functionality:

1. (5 points) It uses FileStreams
2. (5 points) It uses FileStreams with buffer 100 kb
3. (5 points) It uses FileChannel
4. (5 points) It uses NIO 2 File API

After that prepare a performance report based on next requirements.

Measure the time for copying, run on several reference files of different sizes (1 Kb, 100 Kb, 10 Mb, 1 GB). On each file, run 1000 times, get the average time.

**Task 4 - Play the Game with DBUnit**

**Cost**: 1 point.

Add five DBUnit tests to appropriate project (pet project) or another. Prepare test datasets if it is required.

**Task 5 (Optional) - Stored Procedure**

**Cost**: 1 point.

1. Take the existing (or write from zero) JDBC solution with a few CRUD operations and more complex SQL (for simple report generation) and migrate it to stored procedures. **\***
2. Write SQL script to create those stored procedures with Java style parameters and specific external names. **\*\***
3. Write a test which drops all stored procedures and creates a few of them via Java code.
4. Also, provide the script to print out all stored procedure in your database and dropping them for test purposes, for example.
5. Check that the application works properly, all test are green and so on.
6. Compare the performance of two solutions; explain to your mentor the benefits or disadvantages of storage procedure usage for the taken application.

**\*** 3-5 tables with CRUD operations and two complex SELECTs can be enough. **\*\*** Use MySQL or PostgreSQL or Oracle.

**Task 6 (Optional) - Pre-defined Stored Procedure**

**Cost**: 1 points.

If your database **\*** has pre-defined storage procedure, use a few of them to print out interesting information or maintain something in the database.

**\*** Use MySQL or PostgreSQL or Oracle.

**Task 7 - File Sharing**

**Cost**: 30 points.

Implement the next use cases of File Share application:

1. Save file to the database.
2. Retrieve file from the database.
3. Optional: file expiration.

Large files should be supported (size up to 200 MB).

Acceptance criteria:

1. File Share database schema is developed:
   * DB schema diagram is provided (5 points);
   * stored procedures for saving and retrieving files from DB are created (5 points).
2. DAO on JDBC is implemented:
   * DAO methods that are not used in proposed use cases can throw UnsupportedOperationException (2 points);
   * CallableStatement is used to call DB stored procedures (3 points);
   * large binary files are retrievable from DB (5 points).

Think about **pros** and **cons** of stored procedures usage comparing to SQL statement stored in Java code. Describe what difficulties you’ve faced when working with large binary files. Make demo via console interface or via special main method.

**Task 8 - JabaORM**

**Cost**: 30 points.

Write your own annotation-based or XML-based JabaORM that parses specific class and generates SQL-queries for CRUD (or SCRUD) operations.

Your mini-ORM should have one entry point, which supports CRUD operations for parsed class passed as a parameter in

* .save
* .load
* .update
* .delete

methods.

Implement all actions via RowSet if it is possible.

**Bonus**: inheritance support will be a plus (+5 points).

**Penalties**

* -5 points for the missed subtask;
* -5 points if table populating algorithm is trivial and ugly (using simple counters without the fantasy flight);
* -3 points if mentee doesn't use try-with-resources if it's possible;
* -2 points if connection parameters are not stored in property file;
* -5 points for each task if mentee ignores Java Code Conventions.