Philipps University of Marburg
Department of Mathematics and Computer Science
Database Research Group

Winter semester 2024/2025
October 31, 2024

Prof. Dr. Bernhard Seeger
Amir El-Shaikh, M.Sc.
Marius Kuhrt, M.Sc.

Exercises for the lecture
Implementation of Database Systems

Submission: November 1, 2024,
no later than 8 p.m.
via the ILIAS platform

Exercise 1

Submission Requirements:

1. For implementations, you **must** follow coding conventions. You can read the java coding conventions here: Java Code Conventions.
2. Submissions failing to meet standardized coding conventions or don't compile will be **rejected**.

Task 1.1: Manuel Serialization (4+1+2+3) (10 Points)

You should become familiar with serializing records using the java programming language in this task. You can find the binary file `stocks.bin` of fictitious stock exchange prices in the ILIAS that contains tuples of form:

`(ID, Name, Timestamp, Market Value)`

ID is an Integer, *Market Value* is to be interpreted as a floating point value and *timestamp* as an integer. *Name* contains the respective company's name and can be of any length. This is encoded in the file as an integer n (number of bytes in string *Name*) before the actual *Name* string. The file thus has entries in the following format:

ID	*n*	*Name*	*Timestamp*	*Market Value*
`Integer`	`Integer`	`utf-8 String`	`Integer`	`Floating Point`
8 Bytes	2 Bytes	n Bytes	8 Bytes	8 Bytes

a) Define a constructor to create an instance from a given `ByteBuffer` instead of using parameter values for the class fields. Then, add the method `ByteBuffer getBytes()` to transform the record to the binary representation accordingly.

b) In class **Stocks**, define a constructor `Stocks(String path)` to access the binary file at the given `path` using a `RandomAccessFile`.

c) In class **Stocks**, implement the method `StockEntry get(int i)` to parse the record number `i` from the binary file.

d) Implement the `Iterable` interface to the class **Stocks**, to support an `iterator` that returns all records from the binary file from beginning to end without repeatedly calling `get()`.

Task 1.2: Serialization in Fixed-Size Data Containers (5+10) **(15 Points)**

Containers (`Container<Key, Value>`) are a collection of similar objects (in this task of **fixed size**) that can be accessed using unique keys, regardless of the underlying storage medium. The keys are managed by the container and passed on to the caller. Moreover, keys are of type `Long` and are generated starting from 0 in ascending order.
In this task you should simulate a container in memory and a second time implement the container via a data-file, i.e., on disk.

a) Complete the class `MapContainer<Value>`, which implements a container based on the java interface `java.util.Map<Key,Value>` to simulate an **in-memory** container.

b) Complete the class `SimpleFileContainer<Value>`, which implements a **file-based** container.
Consider the following points:

- Main memory structures cannot usually be stored 1:1 in files but must be serialized appropriately (converted to bytes). The `FixedSizeSerializer<Value>` passed in the constructor should be used for this purpose.
- Manage **two** separate files. One for the contents of the container and one for the associated metadata. In the context of a container, metadata refers to information about the contents of the container, such as its object size.
- When deleting, note that you cannot simply cut out parts of a file; otherwise, the keys will no longer be valid. Instead, mark the object as deleted, e.g., by adding a byte before the actual data of an entry or by a bit on the key itself.