**Project Planning – distributed db**

Structure:

The project is built using three main classes – Database, DbSynchronizer, DbFileHandler.

Also contain an Enum for the db running options – OperationSettings.

There will also be two unit-tests testing the database in its two options of operation – threads, processes.

Class docs:

Database class:

Guidelines:

* The db will be stored in a dictionary
* The db will only be accessed by the Database class
* The Database class will only handle the db directly, no file or sync handling.

Functions:

set\_value:

Args:

* key: type – any, the key where the data needs to be stored
* value: type – any, the value to store in the db.

Return value:

Type – bool

Insertion successful or not.

Additional info:

* If a key does not exist then create a new key in the db.

get\_value:

Args:

* key: the key where the value is stored

Return value:

Type – Any

The stored value if nonexistent None

delete\_value:

Args:

* key: the key to delete

Return value:

Type – Any

The stored value if nonexistent None

DbFileHandler:

Guidelines:

* Uses pickle to load and save the data into and from the file

Functions:

get\_data:

Args:

None

Return value:

Type – dict

What was written in the db file

Additional info:

* Use the pickle load function to load the information from the file
* Make sure that file is not empty before using the pickle function.

write\_data:

Args:

None

Return value:

None

Additional info:

* No need to receive the dict because it is already stored in the instance of the Database class.

Additional info:

* data must include the entire database not only the changes – pickle overrides everything that was in the file before.
* Use the pickle dump method to save the data into the file.

DbSynchronizer:

Guidelines:

* While reading up to 10 readers allowed
* Only one writer is allowed, and no readers are allowed while writing
* Writers get priority to the db.
* Reading and writing to the file will be kept to a minimum.
* Reading of the file will only happen once at the start of the program the result will be saved in a variable available for all readers.
* File will be accessed only when writing, after writing the readers will **not** read the db from the file again – **the changes to the db will be stored in the global variable from the start**.
* **The global db will be saved in a** **backup temp var before writing** in case of an unsuccessful writing so that the global variable will not be different from the file.

Functions:

read\_from\_db:

Args:

* key: type – Any, the key to get the value from

Return value:

Type – Any

The value from the key, if key is nonexistent None

Additional info:

* When someone is reading you must check if there are **less than 10 readers** and check if there is a **writer waiting**
* If a writer is waiting the reader will not get reading permission

write\_to\_db:

Args:

* key: type – Any, the key to write the info to.
* Value: type – Any, the value to add to the key

Return value:

Type – bool

If writing was successful return true else return false

Additional info:

* The writer must wait for there to be **no readers** and **no other writers** before him.

\_\_init\_\_:

Args:

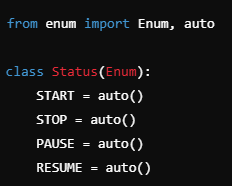
* operation\_method: type – OperationSettings, setting whether to work on threads or processes.

Return value:

None

Additional info:

* In the constructor it will set a **private** member called **operation\_method**. This member will help the method to define the correct locks.

OperationSettings:

Guidelines:

* This class will be an Enum holding the operation methods of the db
* It will hold two settings THREADS, PROCESSES.