Маляр Иван Владиславович

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3**

**НА ТЕМУ: «Mini MapDB»**

***Ссылка***

<https://github.com/malyariv/labs/tree/master/lab3>

***Реализованная функциональность:***

* классами (DBSet<T>, DBList<T>, DBCacheList<T>, DBQueue<T> и DBMap<K,V>) были реализованы стандартные интерфейсы Collection и Map, где элементы хранятся в файлах на жестком диске, а не в оперативной памяти. Для этого используется сериализация, то есть возможна работа только с объектами классов с интерфейсом Serializable;
* каждый объект записывается в файл типа число.ser (начиная с 0), а в файл с расширением .txt записывается (размер ранее записанных объектов+размер объекта). В классе DBCacheList объекты записываются блоками (массивами), что экономит время на записи, последовательном чтении объектов и их удалении, однако снижает скорость чтения объектов из разных блоков.
* количество объектов в файле (writesize) не может превышать 50\_000 элементов, но может быть меньше. Предполагается, что размер файла не будет превышать 250 Мб, поэтому при добавлении первого элемента рассчитывается место, сколько он занимает на диске, откуда определяется число элементов в файле. Предполагается, что размер файла не превысит Integer.MAX\_VALUE байт.
* количество объектов в блоках (blocksize) для DBCacheList также определяется при добавлении первого элемента, при этом это число является делителем количества объектов в файле (writesize). Например, 500.
* наряду с хранением на диске, в оперативной памяти для каждого объекта хранятся его хеш-код для быстрого поиска, а также либо логическая переменная, либо переменная типа int. Она нужна для «гибкого удаления» объектов и реализации операции set(T t) и add(int index, T t). Гибкое удаление заключается в том, что объект получает флаг false или индекс -1, и уже не индексируется программой. Реальное же удаление объектов с жесткого диска происходит только, если в целом файле нет актуальных объектов, или путем слияния двух файлов, если количество актуальных объектов в них меньше writesize.
* есть возможность сохранить состояние коллекции с помощью метода save() и возобновить работу с ней при следующем запуске;
* есть возможность вызвать метод recover(String folder, int maxFileIndex), позволяющей восстановить коллекцию из файлов в папке folder из (maxFileIndex+1) файлов. При этом могут быть восстановлены элементы, которые ранее были удалены в программе;
* у каждого класса есть конструктор, в котором надо указать папку, куда будет происходить запись объектов, а также флаг на возобновление работы с прошлыми данными. У DBSet() также есть конструктор, принимающий папку и Set для ускорения заполнения.
* У каждого класса имеется логгер для протоколирования. В частности, он записывает writesize и blocksize, ошибки, и размер коллекции при операции слияния файлов или удаления.
* для каждого класса имеется две обертки для работы в многопоточном режиме: synchronized и locked. Вторая использует ReadWriteLock, то есть возможно много читателей и только один писатель. Итератор получил WriteLock, так как у него реализован метод remove();

***Описание UML-схемы и реализованных классов:***

***Основные классы***

* The classes DBSet<T>, DBList<T>, DBCacheList<T>, DBQueue<T> and DBMap<K,V> implement methods of corresponding interfaces. Each class store objects on disk and their hashcode in inner object of HashContainer class.
* The class *AHashContainer* is an abstract class describing an array of object hashcode values. This array is a reflection of the file containing real serialized objects.
* The class *BooleanHashContainer* extends *AHashContainer* adding a boolean array coupled with the hashcode array. A boolean value indicates the status of object: presented or to be removed. The class implements abstract methods of the parent class as add(int h), remove(int index), …
* The class *IndexedHashContainer* extends *AHashContainer* adding an int array coupled with the hashcode array. An int value indicates the index of object in collection. Value of -1 is a default value or indicates that object is removed. The class implements abstract methods of the parent class as add(int h), remove(int index), …
* The class *ADataNode* is an abstract class providing a structure to store data on HDD and basic operations of data processing as add(T t), remove(Object o), etc. Every object will be written on disk but its hashcode will be stored in a HashContainer object. This class also implements some methods as iterator(), toArray(), containerMerge(), recover(), etc.
* The class *SetDataNode* extends *ADataNode*. It uses *BooleanHashContainer* to work with objects.
* The class *ListDataNode* extends *ADataNode*. It uses *IndexedHashContainer* to work with objects.
* The class *CacheListDataNode* extends *ListDataNode*. It uses inner cache to store objects. Cache will be written into file when it is full.

***Вспомогательные классы:***

* The class *AState* is an abstract class with a single method add(T t) which will be implemented by inheritors.
* The class *InitialState* extends *AState*. It describes the operations during first object was put into collection as serialization check, to calculate the number of objects in one file, etc.
* The class *ListState* extends *AState*. Every object will be written in a file and its hashcode will be put into HashcodeContainer object.
* The class *SetState* extends *AState*. It describes the operations for sets. In particular, for every object will be invoked method contains() before adding. Then object will be written in a file and its hashcode will be put into HashcodeContainer object.
* The class *FileUtils* implements methods to work with a file system and to read/write objects on disk using serialization.
* The class *FileUtilsExt* extends the class *FileUtils*. It implements methods to read/write an array of objects on disk using serialization. It also contains cache to increase reading objects from one block.
* The class *ConfigClass* is a container to store runtime parameters of DBSet<T>, DBList<T>, DBCacheList<T>, DBQueue<T> and DBMap<K,V>.

сама UML-диаграмма в файле UML.pdf