	Lehrveranstaltung	**Databases and Information Systems 2020**		
	Aufgabenzettel	**4**		
	STiNE-Gruppe 14	**Simon Weidmann, Aram Yesildeniz**		
	Ausgabe	**19. Mai 2020**	Abgabe	**09. Juni 2020**

Report

a)

There is a singleton PersistenceManager. He holds atomic Integers for LSN and TID values, so that there are no duplicates for those.

Then he also holds two hashtables. One is for the buffer, mapping pageIDs to buffer pages. BufferPage is a data type that holds TID, pageID, LSN and data (string). The second one is for the transactions (this one could easily be implemented with simple lists as well), mapping TIDs to booleans, saying true if the transaction is committed, falso otherwise.

On beginTransaction(), a new TID is polled and a new entry is inserted into the transaction hashtable.

On commit(TID), a new LSN ist polled, an EOT is logged (new log entry) and the transaction hashtable entry for that TID is set to true.

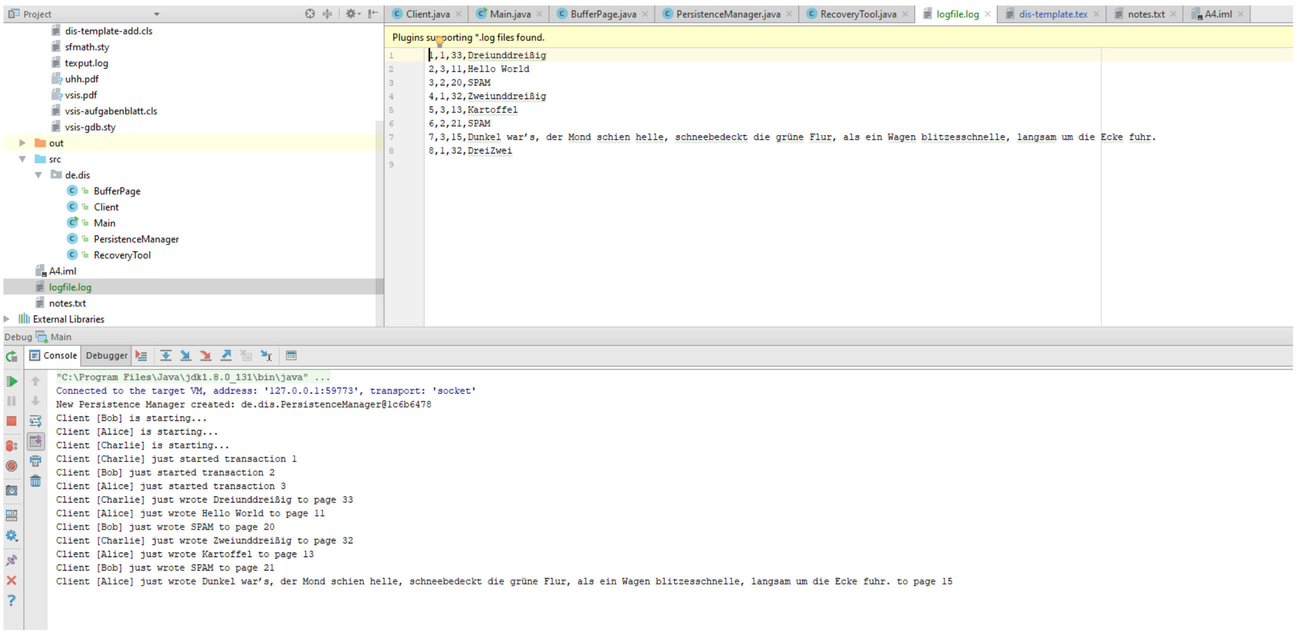
On write(TID, pageID, data), a new LSN is polled, a new BufferPage is created that holds all information (see above) and a corresponding log entry is written. If the buffer now exceed the threshhold, committed transactions will be persisted. This is simply done by iterating over the transaction hashtable to find out which transactions are commited. Then we iterate over the buffer and persist the committed transactions.

There is one RecoveryTool instantiated by the PersistenceManager. It iterates over the log, chooses winner TAs by finding the EOT strings in the log. Then it iterates over the log again and for the winner TAs checks if the log entry has a higher LSN than the actual persisted data page. If not, the logged data will be persisted.

	Lehrveranstaltung	**Databases and Information Systems 2020**		
	Aufgabenzettel	**4**		
	STiNE-Gruppe 14	**Simon Weidmann, Aram Yesildeniz**		
	Ausgabe	**19. Mai 2020**	Abgabe	**09. Juni 2020**

b)

i)

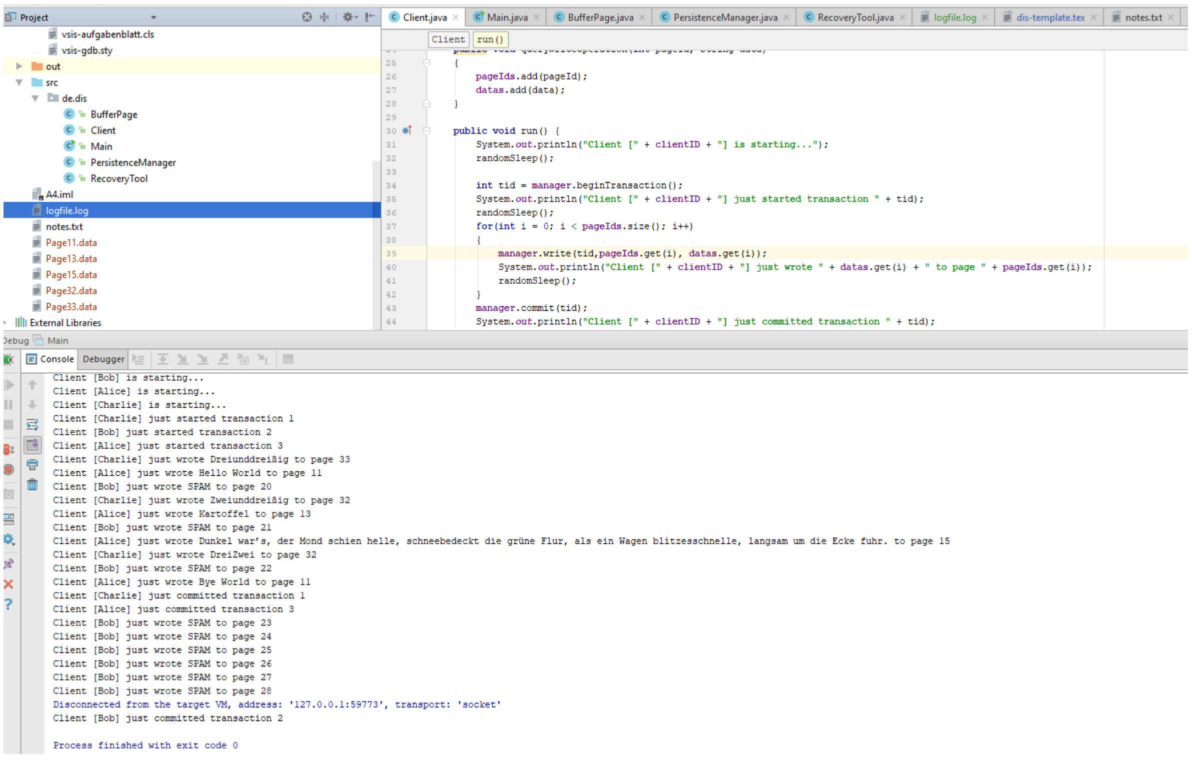


```
1,1,33,Dreiunddreißig
2,3,11,Hello World
3,2,20,SPAM
4,1,32,Zweiunddreißig
5,3,13,Kartoffel
6,2,21,SPAM
7,3,15,Dunkel war’s, der Mond schien helle, schneebedeckt die grüne Flur, als ein Wagen blitzesschnelle, langsam um die Ecke fuhr.
8,1,32,DreiZwei
```

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
Connected to the target VM, address: '127.0.0.1:59773', transport: 'socket'
New Persistence Manager created: de.dis.PersistenceManager@1c6b6478
Client [Bob] is starting...
Client [Alice] is starting...
Client [Charlie] is starting...
Client [Charlie] just started transaction 1
Client [Bob] just started transaction 2
Client [Alice] just started transaction 3
Client [Charlie] just wrote Dreiunddreißig to page 33
Client [Alice] just wrote Hello World to page 11
Client [Bob] just wrote SPAM to page 20
Client [Charlie] just wrote Zweiunddreißig to page 32
Client [Alice] just wrote Kartoffel to page 13
Client [Bob] just wrote SPAM to page 21
Client [Alice] just wrote Dunkel war’s, der Mond schien helle, schneebedeckt die grüne Flur, als ein Wagen blitzesschnelle, langsam um die Ecke fuhr. to page 15
```

As you can see, there are several log entries but no files written (on the left, should be in same directory as the logfile.log)

Lehrveranstaltung	**Databases and Information Systems 2020**		
Aufgabenzettel	**4**		
STiNE-Gruppe 14	**Simon Weidmann, Aram Yesildeniz**		
Ausgabe	**19. Mai 2020**	Abgabe	**09. Juni 2020**



```java
    {
        pageIds.add(pageId);
        datas.add(data);
    }

    public void run() {
        System.out.println("Client [" + clientID + "] is starting...");
        randomSleep();

        int tid = manager.beginTransaction();
        System.out.println("Client [" + clientID + "] just started transaction " + tid);
        randomSleep();
        for(int i = 0; i < pageIds.size(); i++)
        {
            manager.write(tid,pageIds.get(i), datas.get(i));
            System.out.println("Client [" + clientID + "] just wrote " + datas.get(i) + " to page " + pageIds.get(i));
            randomSleep();
        }
        manager.commit(tid);
        System.out.println("Client [" + clientID + "] just committed transaction " + tid);
```

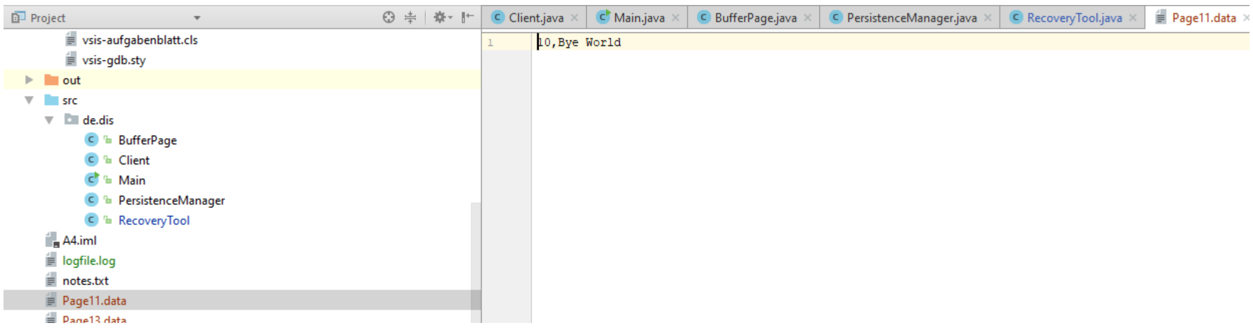
```
Client [Bob] is starting...
Client [Alice] is starting...
Client [Charlie] is starting...
Client [Charlie] just started transaction 1
Client [Bob] just started transaction 2
Client [Alice] just started transaction 3
Client [Charlie] just wrote Dreiunddreißig to page 33
Client [Alice] just wrote Hello World to page 11
Client [Bob] just wrote SPAM to page 20
Client [Charlie] just wrote Zweiunddreißig to page 32
Client [Alice] just wrote Kartoffel to page 13
Client [Bob] just wrote SPAM to page 21
Client [Alice] just wrote Dunkel war's, der Mond schien helle, schneebedeckt die grüne Flur, als ein Wagen blitzesschnelle, langsam um die Ecke fuhr. to page 15
Client [Charlie] just wrote DreiZwei to page 32
Client [Bob] just wrote SPAM to page 22
Client [Alice] just wrote Bye World to page 11
Client [Charlie] just committed transaction 1
Client [Alice] just committed transaction 3
Client [Bob] just wrote SPAM to page 23
Client [Bob] just wrote SPAM to page 24
Client [Bob] just wrote SPAM to page 25
Client [Bob] just wrote SPAM to page 26
Client [Bob] just wrote SPAM to page 27
Client [Bob] just wrote SPAM to page 28
Disconnected from the target VM, address: '127.0.0.1:59773', transport: 'socket'
Client [Bob] just committed transaction 2

Process finished with exit code 0
```

This is what it looks like after all the commits. Some of the pages are still not persisted as described in the excercise. You could easily implement this to persist all before closing the application.

c)

i)

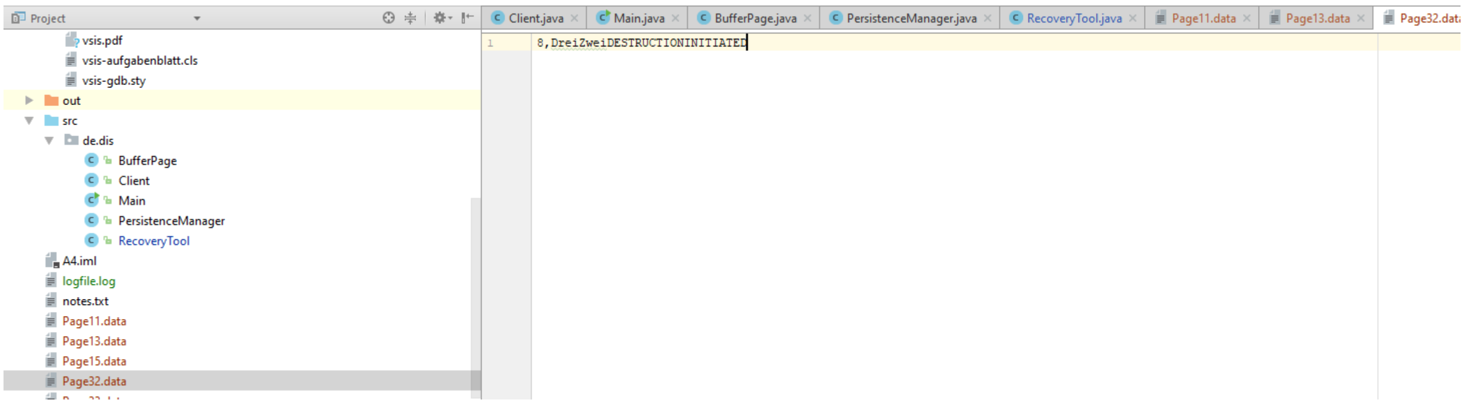
We delete the page 11:



```
10,Bye World
```

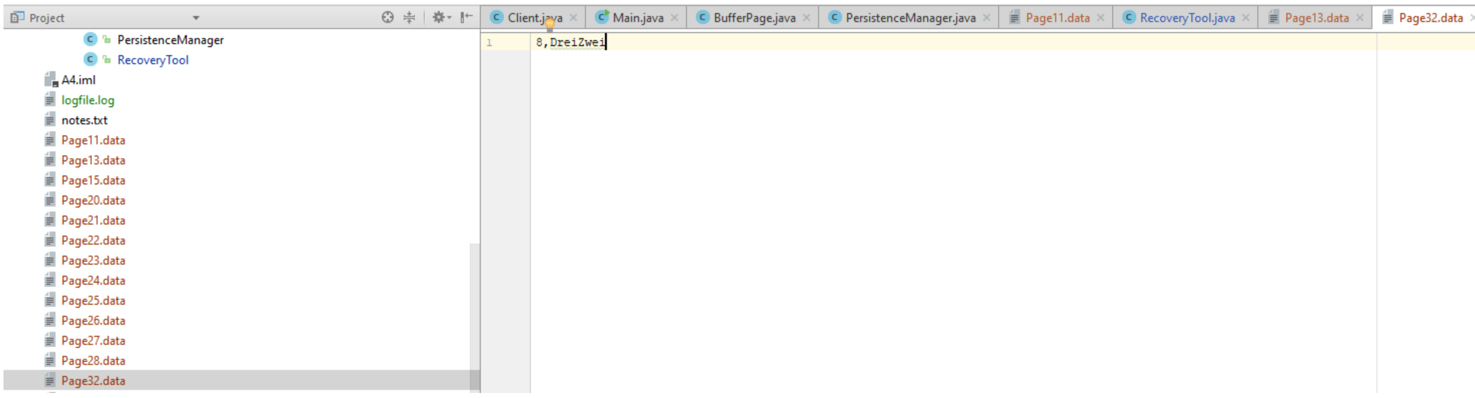
We modify page 32 by adding some stuff in Capslock:

	Lehrveranstaltung	**Databases and Information Systems 2020**		
	Aufgabenzettel	**4**		
	STiNE-Gruppe 14	**Simon Weidmann, Aram Yesildeniz**		
	Ausgabe	**19. Mai 2020**	Abgabe	**09. Juni 2020**



```
8,DreiZweiDESTRUCTIONINITIATED
```

After the recovery, page 11 is restored and the page 32 modification is reverted (note that some other pages were recovered, those that have not been persisted yet after the last run. That was just because Client Alices (pages in the 20s) was the last to commit its transaction and no further write operation was performed afterwards):



```
8,DreiZwei
```

The exact behavior on the restoration depends on if you actively restore from log entries with LSN that are $\geq$ than the persisted one or only greater than them. For both cases, an argument can be made In our case, it does the $\geq$. The lecture says to do it the other way. it would not do a restore. It might do a full consistency check as we will describe below in c)iii), which is very easy to implement (if wanted).

ii)

We have already seen in the last excercise, that Alice's transaction is not persisted after her commit, as the persisting is only triggered by write() operations. So her pages are not even existing yet when the program closes the first time (see in c)i)). As her transaction is fully logged, it is recovered at the next recovery step (see above as well.)

iii)

Nothing happens. Therefore we omit the screenshot. There is no mechanism that would find in error in the data when seeing a higher LSN. One could implement a consistency check in all the data pages, checking if their data is actually the one that it should be according to the log file. The algorithm would go through all pages, check their LSN and look back into the log file if the data is logged as it is persisted. We were not sure if this was intended by looking into the lecture slides. The implementation would be a simple for loop over the log winners and looking into the individual pages.