Message Queue design notes

The basic idea is coming from the post, https://engineering.canva.com/2015/03/25/hermeticity/, and I made some improvement. Using system properties `canva.queue.dir` or environment properties `canva_queue_dir` or `$home\canva_queue_data` as data dir.

basic data structure

For each message queue maintain a deque "double ended queue", the element of the queue is a `Record`, each record storages a message. The define of `Record` looks likes this:

```
1 public class Record {
2     //the body of one message
3     private String messageBody;
4     //the time point when this message is visible to the client
5     private Long visibleFrom;
6     //receipt handle used to delete a message
7     private String receiptHandle;
8 }
```

After pushing a new message to the queue, the record's `visibleFrom` is the current time which means this message is visible to the client at this moment.

```
1 1500816565777::this is a message
```

if the message's visiblity time out is 10 second,after pulling the message, the record becomes like this:

```
1 1500816575777:361f6029-ffe8-4a32-b177-afc70545189f:this is a message
```

the details of each operation are listed below:

operation	implementation	efficiency
push	append a new record to tail of the deque	O(1)
pull	scan from the head of the queue, find first record whose `visibleFrom` is less than current time, before returning, update the record's `visibleFrom` to current time plus the visible time out of the queue	O(n)
delete	scan from the head of the queue, delete the first record whose receipt is matched to the input	O(n)

some high lights

1. making program much more robust by using FileLock instead of using `mkdir()` function

As mentioned in the artical listed below,

> .lock is used to establish across-process mutex lock,by leveraging the atomicity of mkdir in a POSIX file system.All read and write operations on the queue are performed in critical sections scoped by possession of that lock.

This way works perfectly if program runs normally, however, in really world, crash happens all the time. If using a directory as lock, the lock could not be released if the program crashes. The better way to implement to a lock between processes is the `FileLock` introduced since JDK1.4. Read this for info about `FileLock`.

2. test the visibility timeout but don't relay on on physical time or thread sleeping.

In both of memory and file queue, the system's time is retrieved through `clock.now()` instead of calling `System.currentTimeMillis()` directory. When doing unit test, I inject an different implentation of `Clock` called `TimeMachine` which could move the "time" forward and back. Its implementaion is quite simple:

```
public class TimeMachine extends Clock {

    long currentTime = 0L;

    public Long now() {
        return currentTime;
    }

    /**
     * move the 'time' forward
     */
    public TimeMachine moveForward(long duration, TimeUnit unit) {
        currentTime += unit.toMillis(duration);
        return this;
    }
    ...

}
```

3. test time and code coverage

test time on my MacBook Air

```
Running com.example.FileQueueTest
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.841 sec
Running com.example.InMemoryQueueTest
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.018 sec
```

code coverage

80% classes, 82% lines covered in package 'com.example'

Element	Class, %	Method, %	Line, %
SqsQueueService	0% (0/1)	0% (0/7)	0% (0/33)
InMemoryQueueService	100% (1/1)	100% (7/7)	97% (45/46)
FileQueueService	100% (1/1)	100% (23/23)	92% (163/176)
Clock	100% (1/1)	100% (1/1)	100% (2/2)
model	100% (1/1)	100% (9/9)	100% (15/15)