**1. 구현사항**

LevelDBBlockDB의 다음 메소드들을 구현했습니다:

* Block getBlock(BlockKey key)
* void setblock(BlockKey key, Block block)
* void removeBlock(BlockKey key)

그 외 위 메소드들의 구현을 위해 Block 클래스와 BlockKey 클래스에 각각 toBytes함수와 fromBytes 함수를 구현했습니다.

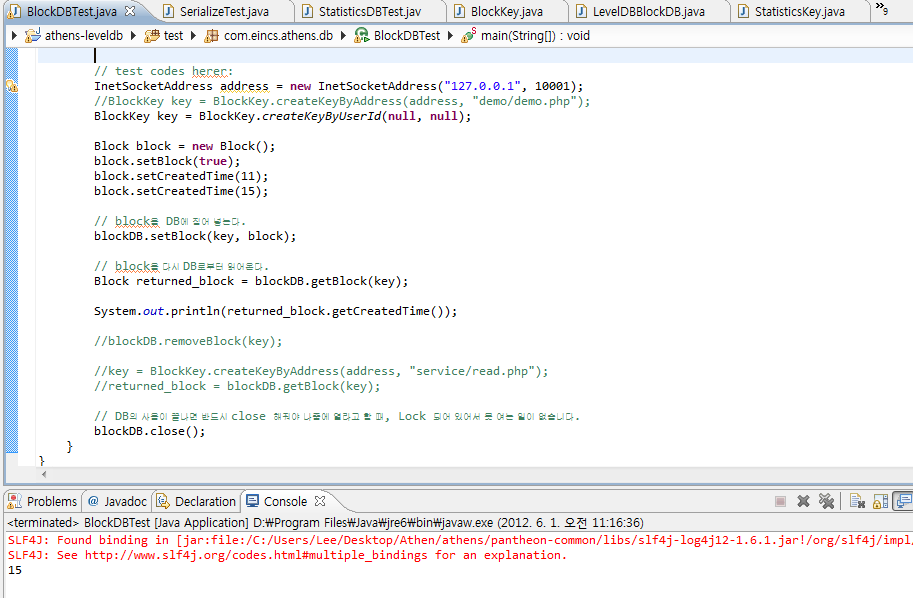
**2. 테스트 사항**

다음의 항목들에 대해 테스트 해 보았습니다:

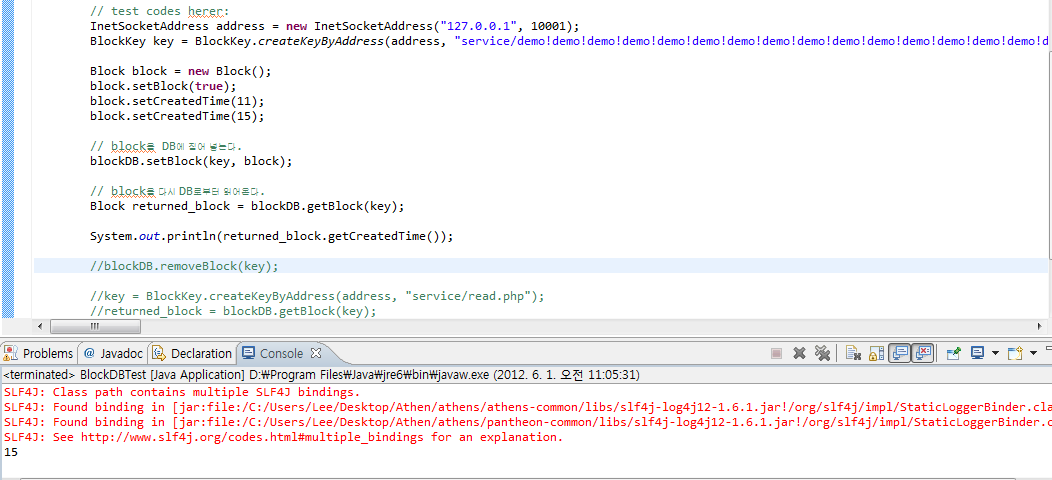
* 너무 짧거나, 긴 key값에 대한 대응
* 이미 입력된 key값 다시 set 시도
* 멀티 스레딩 환경에서 연속적인 삽입/읽기/삭제

**2.1. 너무 짧거나 깉 key값에 대한 대응**

짧은 key 값에서 어떻게 작동할지 알기 위해, key값을 결정할 인자들에 null값을 줘 보았습니다.

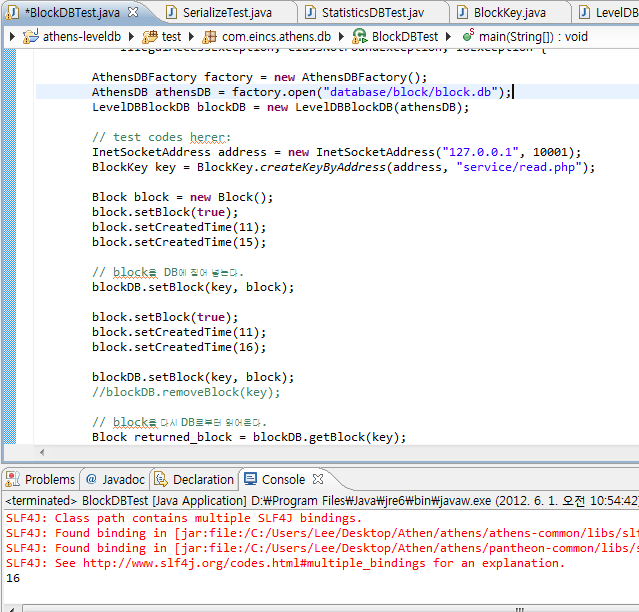


매우 긴 key값 (4000바이트 이상)도 줘 보았습니다.



위의 두 경우 모두 잘 작동합니다. key값의 길이는 거의 신경을 쓰지 않아도 될 것 같습니다.

**2.2. 이미 삽입된 key값으로 다시 입력**



그냥 value값이 자동으로 replace됩니다.

**2.3. 멀티 스레딩 환경에서 연속적인 삽입/읽기/삭제**

저희 시스템이 blockDB는 데이터들이 동시에 추가/삭제 되면서 열심히 읽히기도 해야 되기 때문에, 멀티 스레딩 환경일 것이라고 가정하고, Concurrency test 를 해봤습니다 (제가 잘 이해한 건진 모르겠습니다…).

당연히 동기화를 구현해야 할 줄 알았으나, 몇 번의 삽질 후, 구글에서 검색해보니, OS에서 제공하는 lock을 사용하기 때문에, 적어도 한 프로세스에 의한 멀티스레딩 환경에선 따로 동기화를 구현해 줄 필요가 없다고 합니다. 아, 만약에 iterator나 batch같은 자료구조를 공유한다면 거기에는 synchronize를 해줘야 하지만, 저희는 그런 거 안쓰고 DB에 직접 접근하는 방식이기에…. 필요 없을 거라고 생각했습니다 (관련 내용은 <http://leveldb.googlecode.com/svn/trunk/doc/index.html> 에서 확인 가능합니다). 아 만약에 멀티 프로세스 환경에서 작동하려면 따로 동기화가 필요합니다. 이 기능이 필요하다면 구현하겠습니다.

테스트는 각각 쓰기/읽기/삭제 를 랜덤한 시간간격으로 60번씩 수행하는 3개의 쓰레드를 동시 돌리는 방식으로 진행하였습니다. 결과는 잘 작동합니다 ☺

