2009 Paper 5 Question 6

Concurrent Systems and Applications

(*a*) The following method is intended to return unique integer values to callers:

```
volatile int x = 0;
int getNext() {
  x = x + 1;
  return x;
}
```

(*i*) Two threads call `getNext` concurrently on the same object. Explain how both threads can receive the result `1`. [1 mark]

(*ii*) Explain the semantics of the `synchronized` keyword in Java, and illustrate this by correcting `getNext` (you may ignore the possibility of integer overflow). [6 marks]

(*iii*) Explain the meaning of the `volatile` modifier. Explain whether or not you need to use it with your new implementation of `getNext`. [2 marks]

(*b*) The following method is intended to implement a *barrier* for synchronization between four threads. The first three threads to call the `barrier` method are meant to block. These threads are all unblocked when the fourth call is made.

```
int barrierCount = 0;
void synchronized barrier() throws InterruptedException {
  barrierCount ++;
  if (barrierCount < 4) {
    wait();
  } else {
    notifyAll();
} }
```

(*i*) A programmer finds that some threads return early, although there have been fewer than four calls to `barrier`. How can this happen? [2 marks]

(*ii*) Rewrite `barrier` so that threads wait correctly. [2 marks]

(*iii*) Explain whether or not it would be correct to use `notify` in place of `notifyAll` in your solution. [2 marks]

(*iv*) If a thread is *interrupted* while waiting within `barrier` then the call to `wait` will fail with `InterruptedException`. Rewrite `barrier` so that, if one thread is interrupted when using a barrier, then any future (or concurrent) calls to that barrier will also fail with `InterruptedException`. [5 marks]