ScopedLocking

Always use scoped lockers to lock mutexes and the like. Don't do this:

```
lock.acquire();
do_stuff(); // DANGER: lock never released if exception thrown here.
lock.release();
```

Instead use a "scoped locker". This is simply a class that does the "acquire" in its constructor and the "release" in its destructor:

```
 Locker locker(lock);
 do_stuff();
```

Not only does this save a bit of typing, it guarantees that the lock will be released even if an exception is thrown, because C++ guarantees to call destructors of all local variables on exit from a scope. This also protects you forgetting to release the lock at every exit point from a function with multiple exit points - the compiler takes care of it for you. This technique applies more generally to any situation where you have to acquire and release resources. s `td::auto_ptr` is a similar tool for memory management.