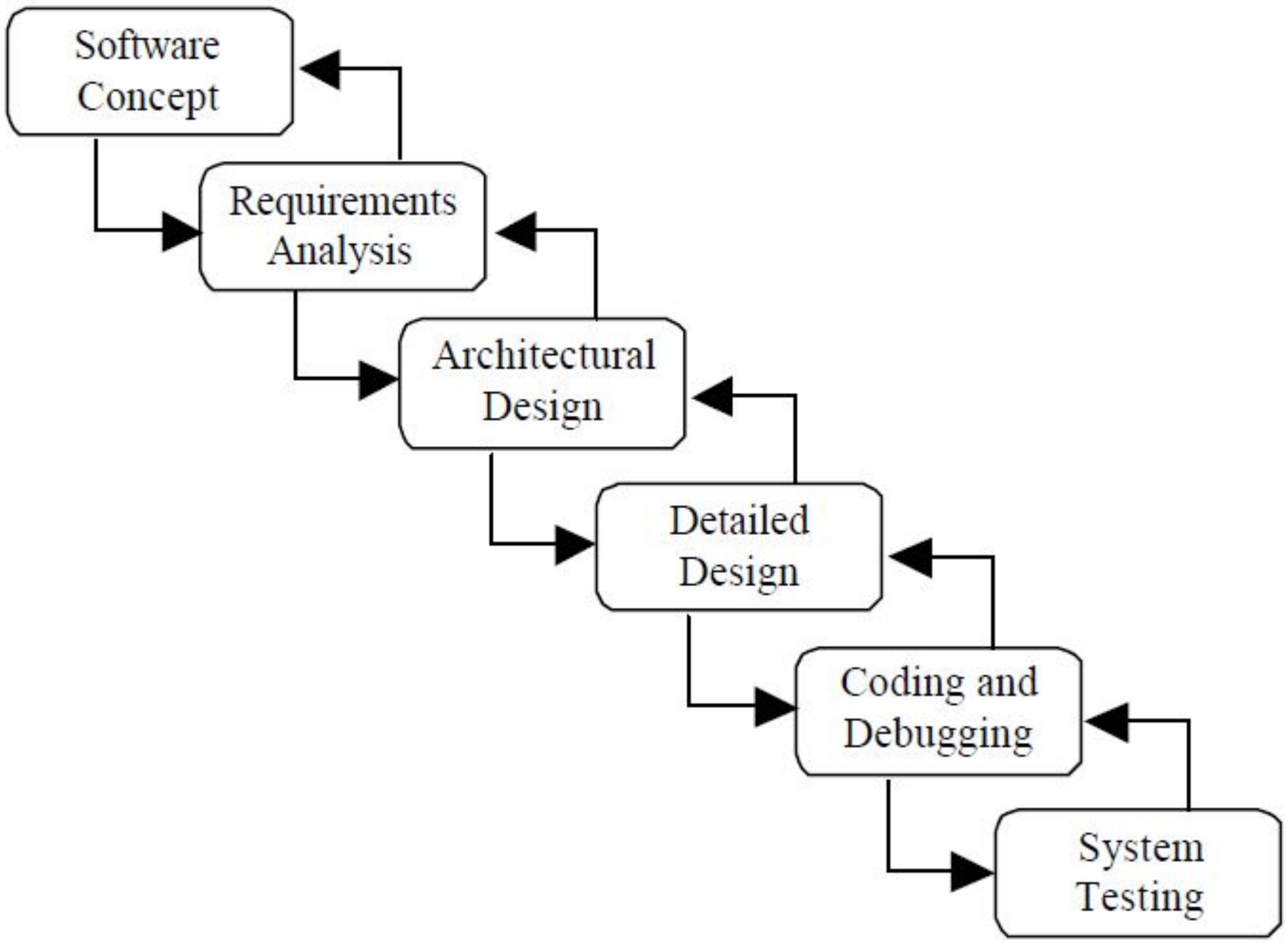
Quick Survey

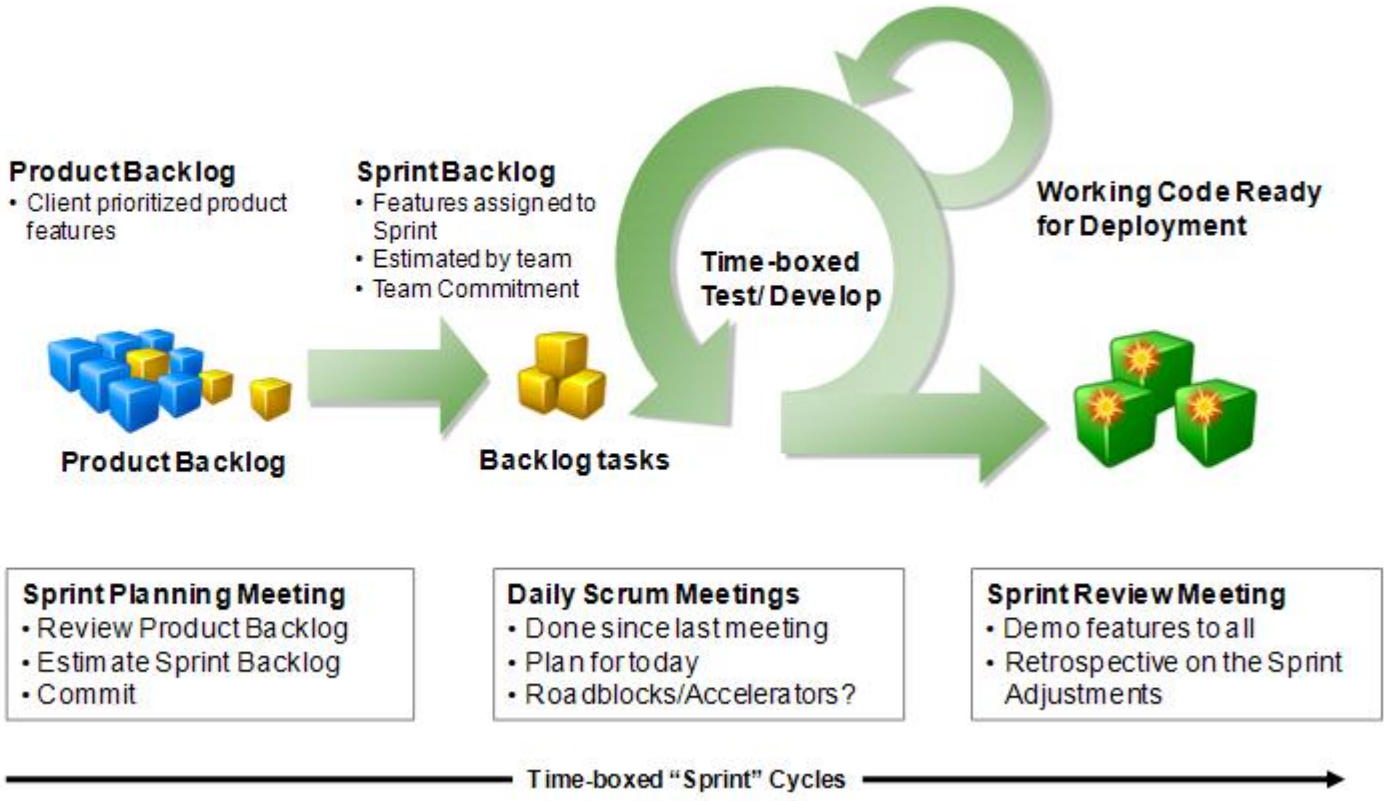
How many of us

- are in Agile Development?
- use TDD (Test Driven Development)?
- work on Java platform?
- work on .Net platform?

Waterfall


Software Concept
Requirements Analysis
Architectural Design
Detailed Design
Coding and Debugging
System Testing

Agile


Product Backlog
• Client prioritized product features
Sprint Backlog
• Features assigned to Sprint
• Estimated by team
• Team Commitment
Time-boxed Test/Develop
Working Code Ready for Deployment
Product Backlog
Backlog tasks
Sprint Planning Meeting
• Review Product Backlog
• Estimate Sprint Backlog
• Commit
Daily Scrum Meetings
• Done since last meeting
• Plan for today
• Roadblocks/Accelerators?
Sprint Review Meeting
• Demo features to all
• Retrospective on the Sprint Adjustments
Time-boxed "Sprint" Cycles

What has Changed?

- No Big Design Upfront
- Clear separation between Architecture and Design
- Less focus on Paper/Documentation
- Focus on Today's requirements
- More prepared for change
- More focus on Code

No Big Design Up Front

- No separate Design phase
- Everyday focus on Design
  - No Design Rot
- Design Evolves
- Decisions made Just In Time

Design Architecture Demarcated

Architectural Decisions are hard to change

- Which framework to use for front end?
- Do we want to use ORM?

Design Decisions today are easy to change

- What should be done in a super class?
- Should a class be abstract?
- Should you have a separate method for a functionality?
- Should you create a new class?

Less focus on Documentation

Detailed Design Documents

- Are usually outdated by the time first development cycle is complete?
  - When rubber meets the road, there will be changes
- Are not updated when code changes

Architectural Documentation is important.

Less focus doesn't mean NO documentation.

Focus on Today’s Requirements

Requirements Change. PERIOD.

Aim to meet today’s requirements with Clean Code.

Complex Design only when Simple Design does not solve the problem.

- Start simple and evolve to use Design Patterns

Change is Expected

Things Change. PERIOD.

Better Prepared for Change

- Good automated test bed
- Simple Design
- Better Refactoring tools
- Better Build tools - Maven

More Focus On Code

"Code" is given utmost importance

Design discussions are done over code

Designer's are expected to Code

Developer's are expected to Design

- And this is where the importance of making Design simple is important. Even starting developers are expected to be able to Design.

Basic Knowledge for Designer

Focus on Principles

4 Principles of Simple Design

SOLID Principles

Test Driven Development

- Refactoring

Standards vs Principles

Clarity of Code

↓

Complexity of a method should less than equal to 10

Length of the method < 35

A class should not be greater than 500 lines

There should not be Magic Numbers

Problems if we follow only Standards

Standards change.

- Numbers like 35 (Method Length), 10 (Complexity), 7 (lines of duplication) change

Tools Improve every day

- 3 Types of Duplication – Now sonar finds Type 2 duplications as well.
- JavaScript rules for complexity.

Most important Maintainability rules are not checked by the tools

- How good the variable/method names are?
- Is decomposition properly done?

Standards without understanding principles causes problems

- Situations like public static int ZERO = 0;
- doThis() calls doThis1() and doThis2()

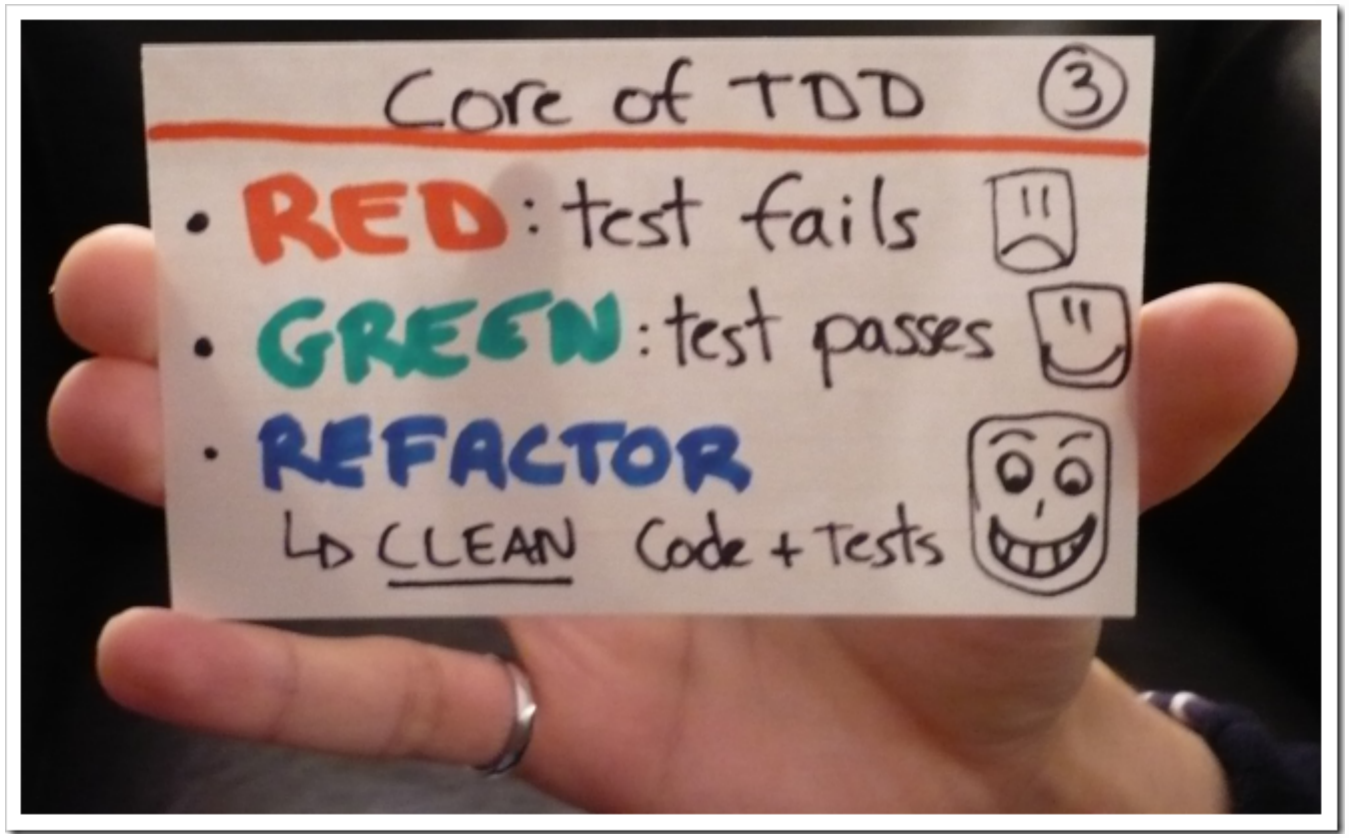
4 Principles of Simple Design

- Runs all tests
- Contains no duplication
- Express intent of programmers
- Minimizes number of classes and methods

SOLID Principles

S	SRP	-	Single responsibility principle
O	OCP	-	Open/closed principle
L	LSP	-	Liskov substitution principle
I	ISP	-	Interface segregation principle
D	DIP	-	Dependency inversion principle

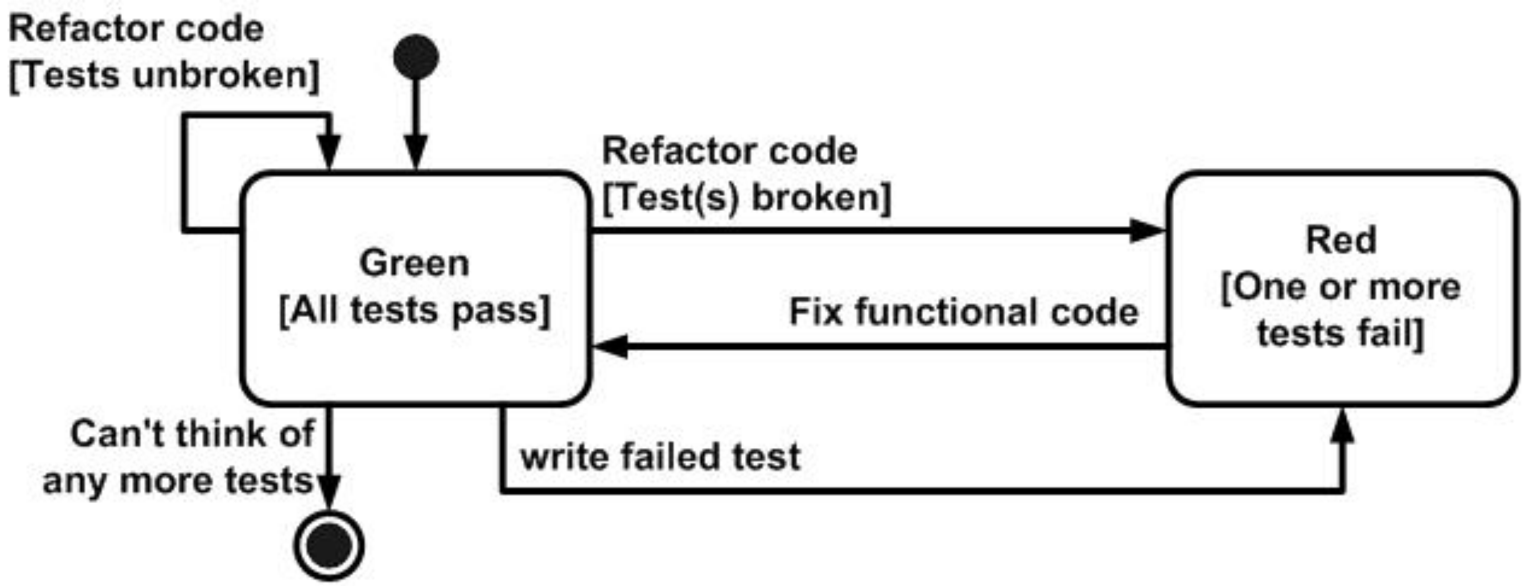
TDD



Core of TDD ③

- RED: test fails
- GREEN: test passes
- REFACTOR
  ↳ CLEAN Code + Tests

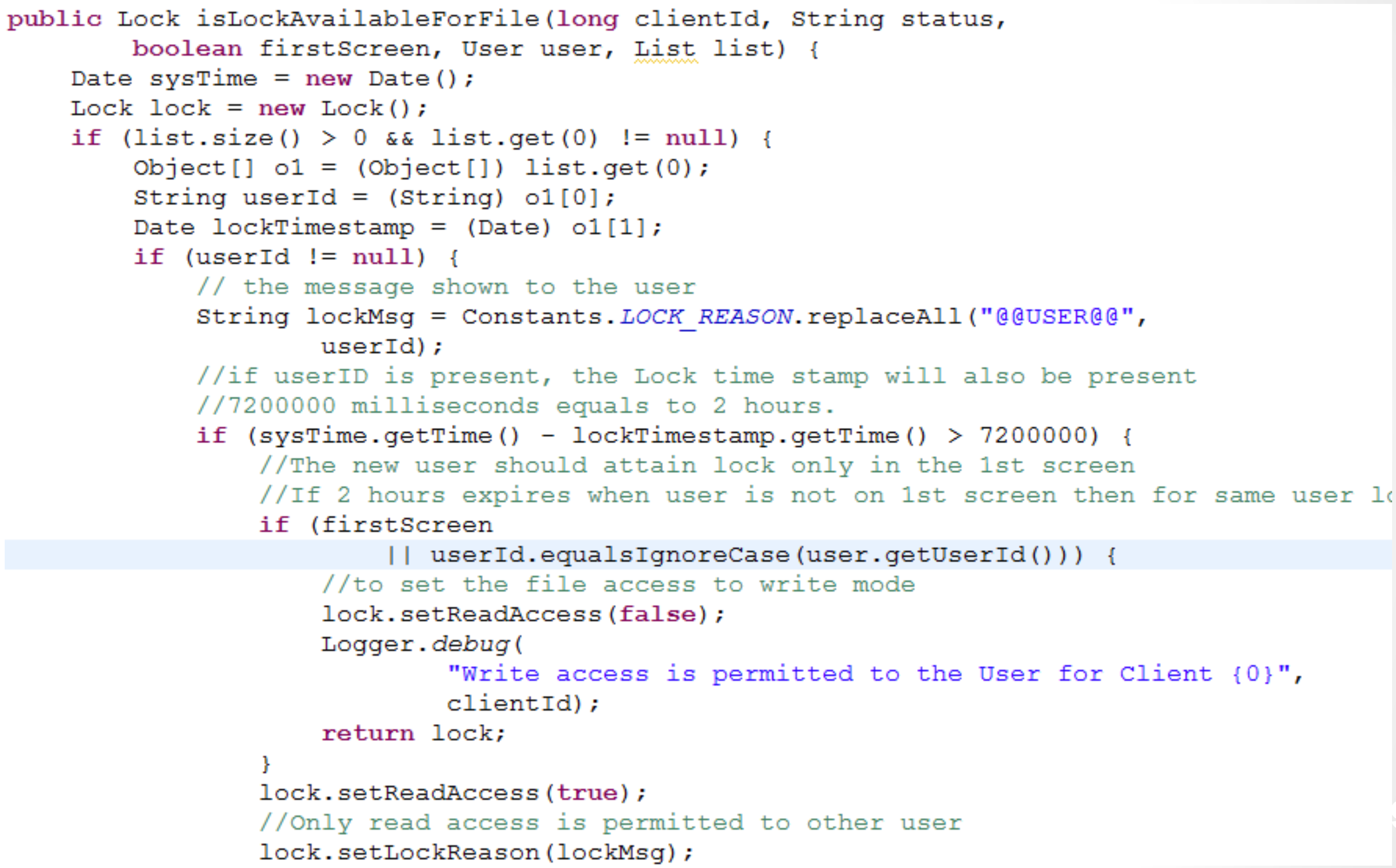
TDD


Refactor code
[Tests unbroken]
Refactor code
[Test(s) broken]
Green
[All tests pass]
Red
[One or more tests fail]
Fix functional code
Can't think of any more tests
write failed test

Refactoring

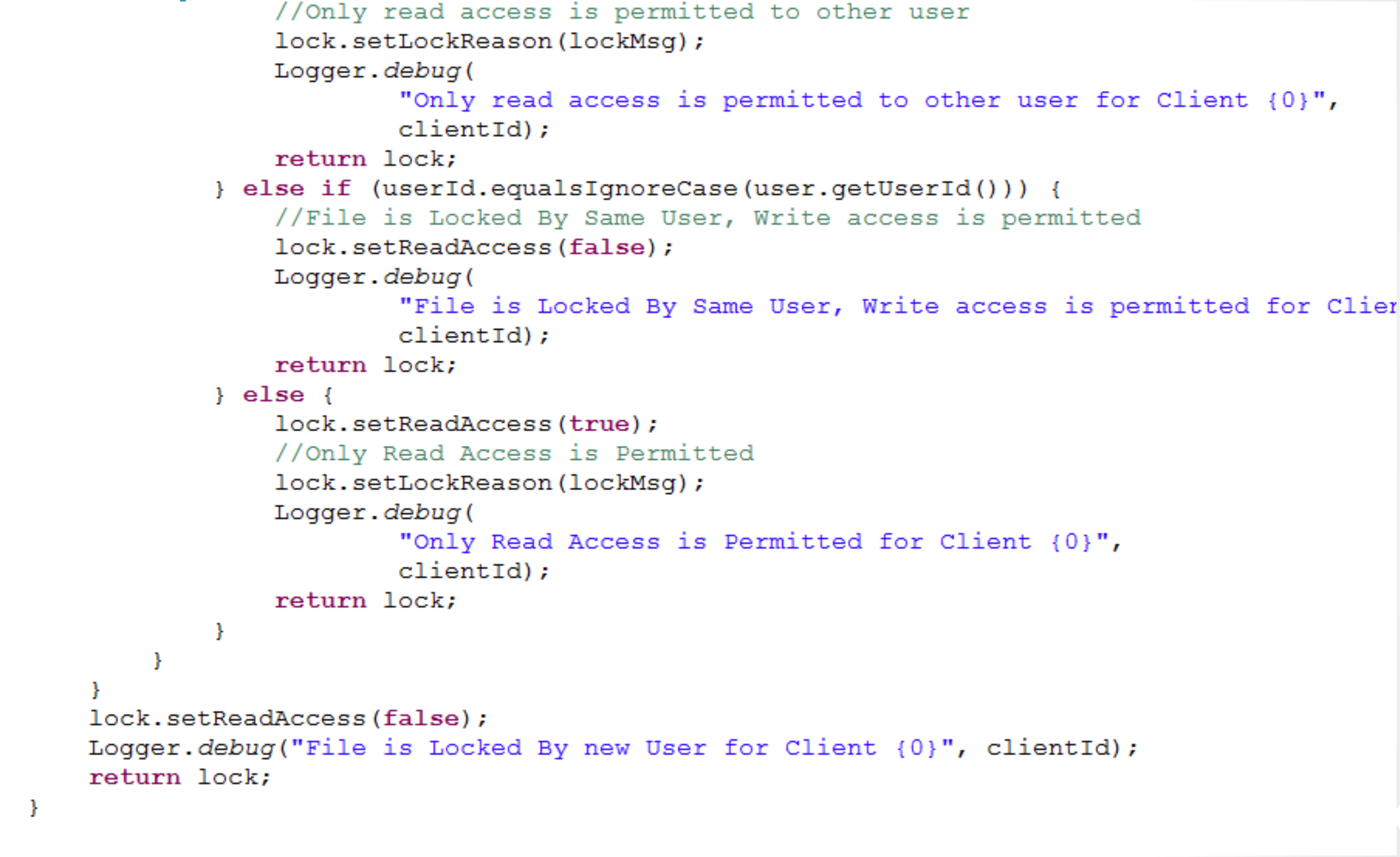
- Altering Structure of Code without affecting “Behaviour”
- Toughest part of Refactoring is the order or sequencing of Refactoring steps
- Continuous Refactoring – aided by Tests – leads to “Clean Code”

Example Code 1



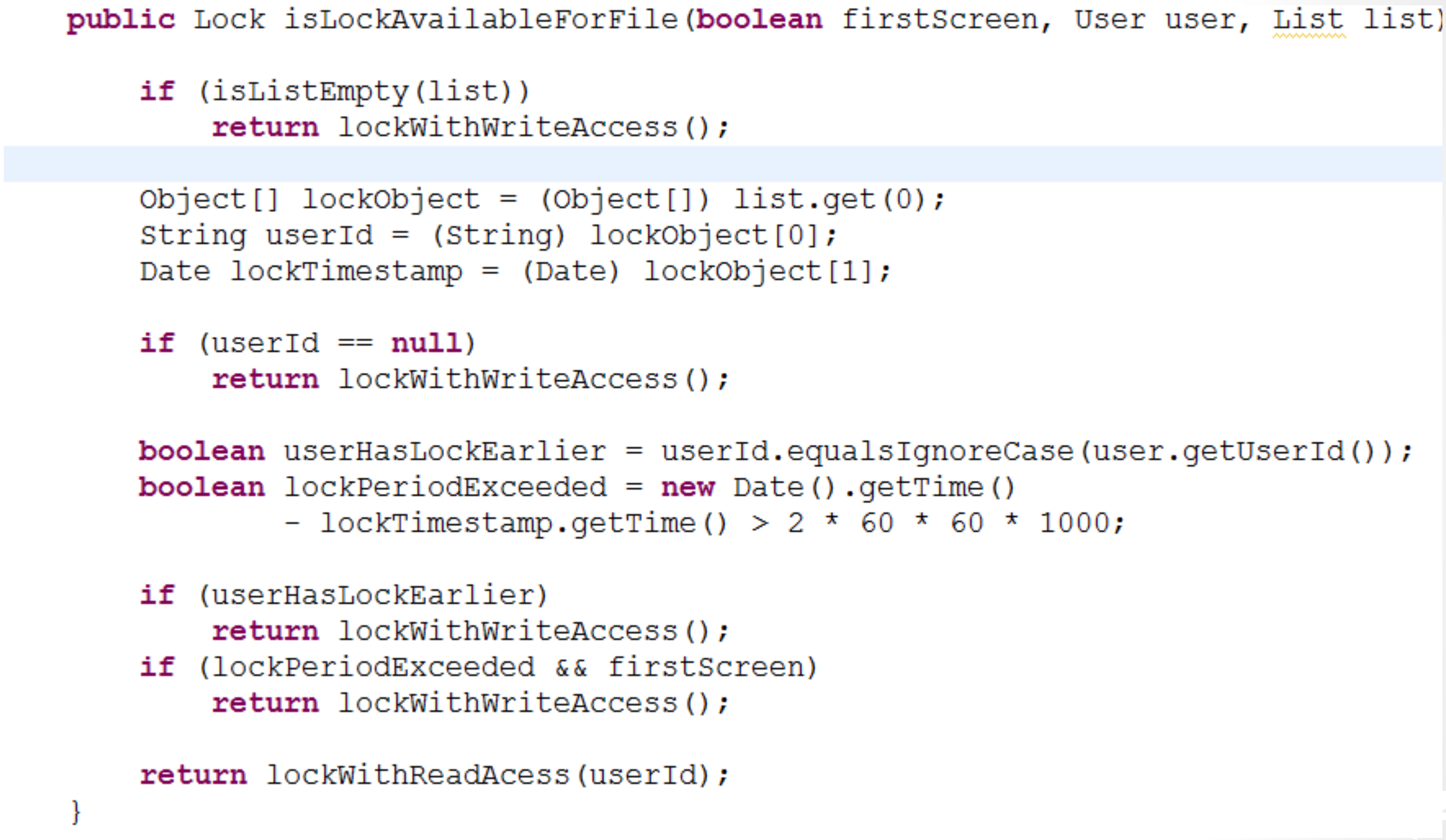
```java
public Lock isLockAvailableForFile(long clientId, String status,
        boolean firstScreen, User user, List list) {
    Date sysTime = new Date();
    Lock lock = new Lock();
    if (list.size() > 0 && list.get(0) != null) {
        Object[] o1 = (Object[]) list.get(0);
        String userId = (String) o1[0];
        Date lockTimestamp = (Date) o1[1];
        if (userId != null) {
            // the message shown to the user
            String lockMsg = Constants.LOCK_REASON.replaceAll("@@USER@@",
                    userId);
            //if userID is present, the Lock time stamp will also be present
            //7200000 milliseconds equals to 2 hours.
            if (sysTime.getTime() - lockTimestamp.getTime() > 7200000) {
                //The new user should attain lock only in the 1st screen
                //If 2 hours expires when user is not on 1st screen then for same user lo
                if (firstScreen
                        || userId.equalsIgnoreCase(user.getUserId())) {
                    //to set the file access to write mode
                    lock.setReadAccess(false);
                    Logger.debug(
                            "Write access is permitted to the User for Client {0}",
                            clientId);
                    return lock;
                }
                lock.setReadAccess(true);
                //Only read access is permitted to other user
                lock.setLockReason(lockMsg);
```

Example Code 1



```
                //Only read access is permitted to other user
                lock.setLockReason(lockMsg);
                Logger.debug(
                        "Only read access is permitted to other user for Client {0}",
                        clientId);
                return lock;
            } else if (userId.equalsIgnoreCase(user.getUserId())) {
                //File is Locked By Same User, Write access is permitted
                lock.setReadAccess(false);
                Logger.debug(
                        "File is Locked By Same User, Write access is permitted for Clie
                        clientId);
                return lock;
            } else {
                lock.setReadAccess(true);
                //Only Read Access is Permitted
                lock.setLockReason(lockMsg);
                Logger.debug(
                        "Only Read Access is Permitted for Client {0}",
                        clientId);
                return lock;
            }
        }
    }
    lock.setReadAccess(false);
    Logger.debug("File is Locked By new User for Client {0}", clientId);
    return lock;
}
```

Example Code 2



```
public Lock isLockAvailableForFile(boolean firstScreen, User user, List list)

    if (isListEmpty(list))
        return lockWithWriteAccess();

    Object[] lockObject = (Object[]) list.get(0);
    String userId = (String) lockObject[0];
    Date lockTimestamp = (Date) lockObject[1];

    if (userId == null)
        return lockWithWriteAccess();

    boolean userHasLockEarlier = userId.equalsIgnoreCase(user.getUserId());
    boolean lockPeriodExceeded = new Date().getTime()
            - lockTimestamp.getTime() > 2 * 60 * 60 * 1000;

    if (userHasLockEarlier)
        return lockWithWriteAccess();
    if (lockPeriodExceeded && firstScreen)
        return lockWithWriteAccess();

    return lockWithReadAcess(userId);
}
```

Thank you