# Test-driven Development

Intensive Workshop

## About Me

Blah blah blah blah blah etc

jason.gorman@codemanship.com

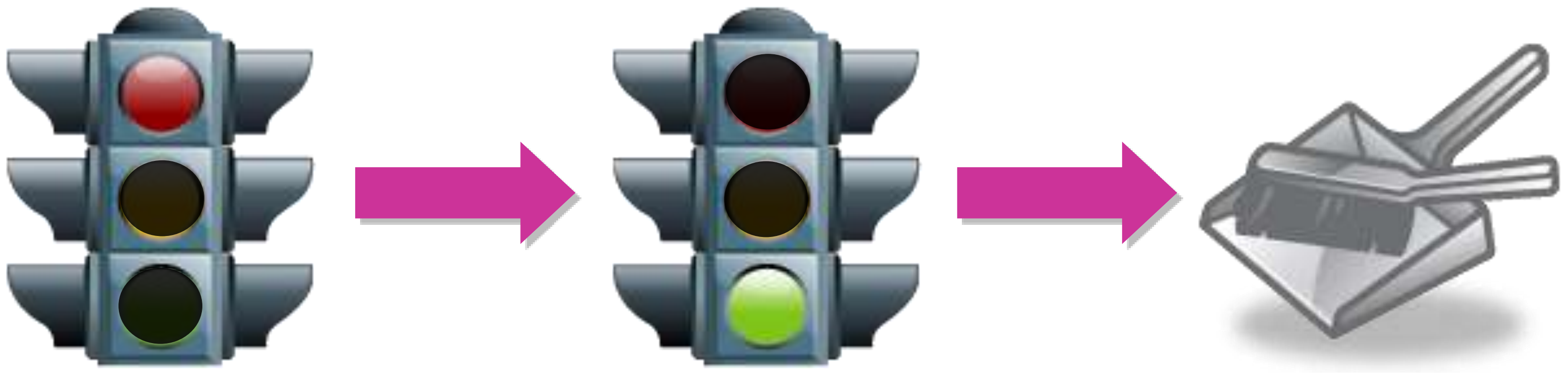
## What To Expect This Morning

- Introduction to TDD
- TDD Basics
  - Write a failing test, write the simplest code to pass the test, refactor to remove duplication
- Example 1- Bank Transfer
- More TDD Basics
  - Write the assertion first and work backwards, see the test fail, write meaningful tests, tests should test one thing, triangulate
- Example 2 – Fibonacci Sequence Generator
- Another 5 TDD Basics
  - Don’t refactor on a failing test, Keep your test and model code separate, isolate your tests, organise tests to reflect model code, maintain your tests
- Example 3 – FizzBuzz

## What To Expect This Afternoon

- Test Doubles – Mocks, Stubs & Fakes
- Example 4 – Using Stubs: JG Holidays Ltd
- End-to-end Test-driven Development
- Example 5 – Community Video Library
- Mastering TDD – TDD Practice Regimes

# Introduction To TDD



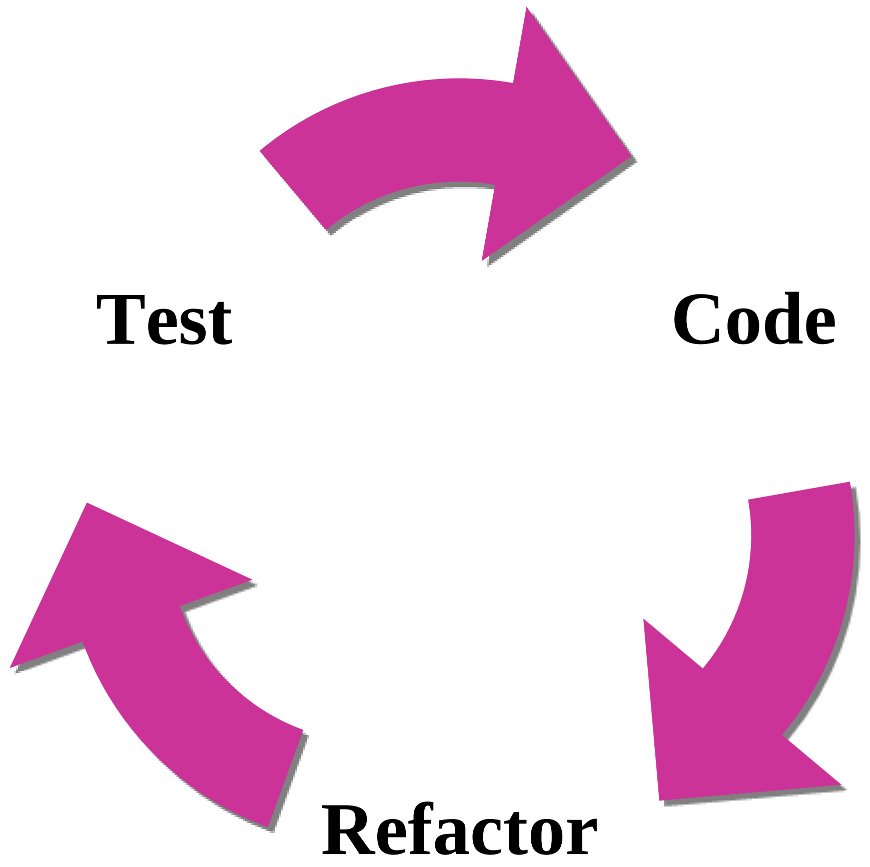
write a *failing* test

write the code to *pass* the test

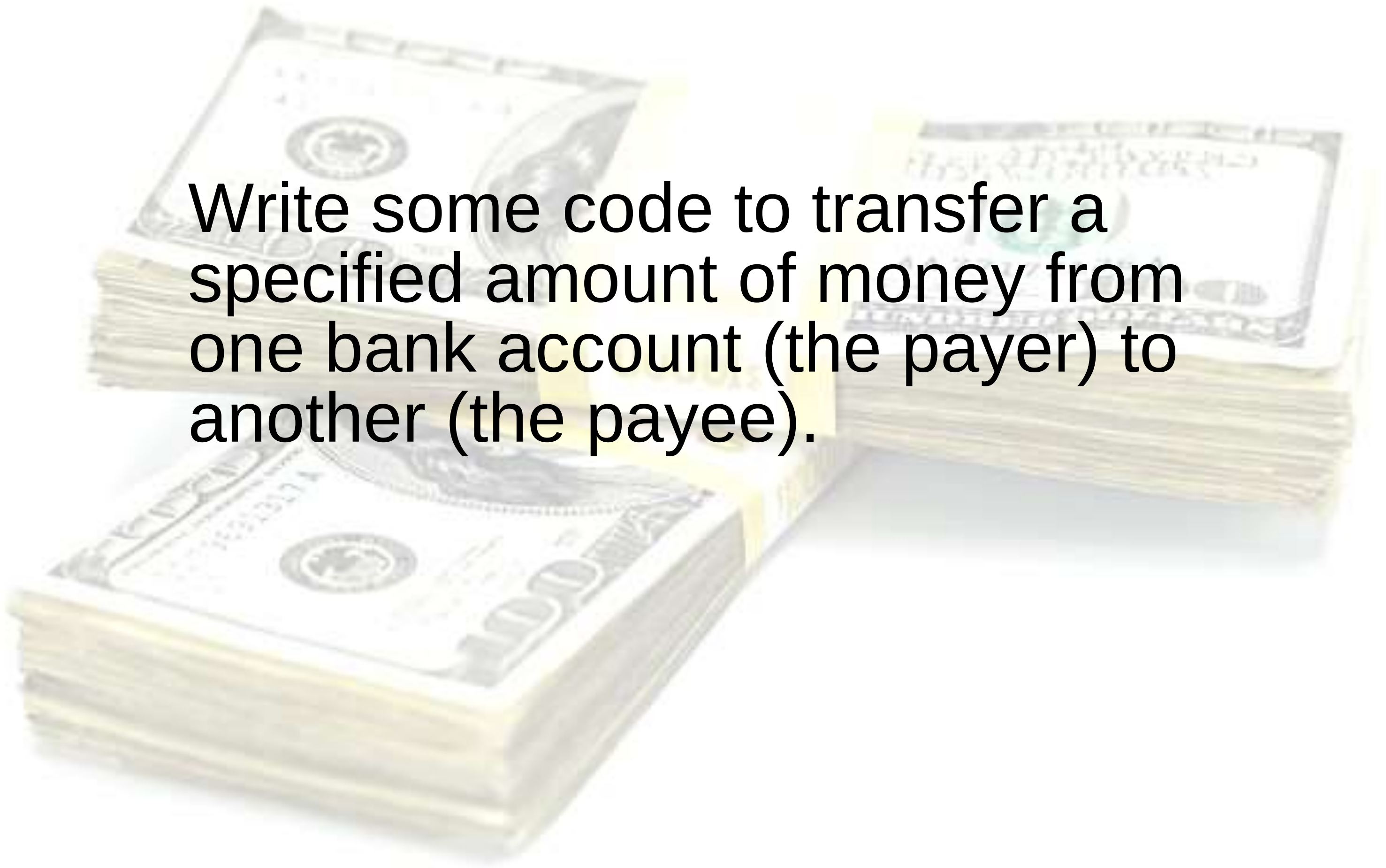
refactor to *remove duplication*

codemanship

## The TDD Cycle



## Example #1 – Bank Transfer

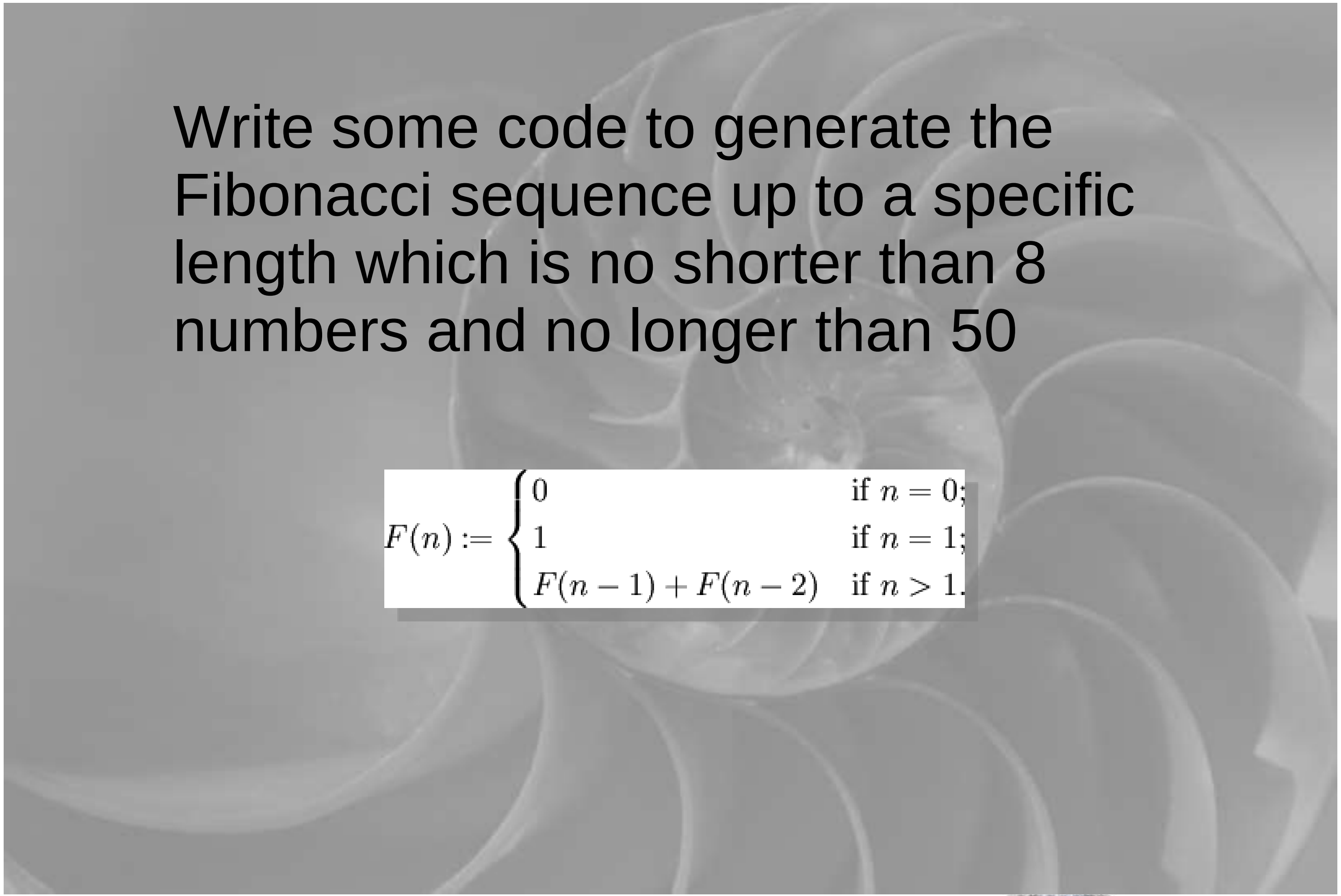


Write some code to transfer a specified amount of money from one bank account (the payer) to another (the payee).

# More TDD Basics

1. Write the assertion first and work backwards
2. Run the test to ensure it fails in the way you expect it to
3. Write meaningful tests that are self-explanatory
4. Tests should only have one reason to fail
5. Triangulate through concrete examples towards general solutions

# Example #2 – Fibonacci Sequence Generator



Write some code to generate the Fibonacci sequence up to a specific length which is no shorter than 8 numbers and no longer than 50

$$F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

# Yet More TDD Basics

1. Don’t refactor when tests are failing
2. Keep your test and model code separate
3. Isolate your tests so they run independently
4. Organise tests to reflect organisation of model code
5. Maintain your tests

# Example #3 - FizzBuzz

Write some code that will generate a string of integers, starting at 1 and going up to 100, all separated by commas. Substitute any integer which is divisible by 3 with “Fizz”, and any integer which is divisible by 5 with “Buzz”, and any integer divisible by 3 and 5 with “FizzBuzz”

1,2,Fizz,4,Buzz,Fizz,7,8,Fizz,Buzz,11,Fizz,13,14,FizzBuzz… etc

# Test Doubles

| | |
|---|---|
| Stub | Class with test-specific implementation (hard-coded responses) |
| Fake | A full implementation for test purposes (e.g., in-memory database) |
| Mock | An interface with expectations set for the test (implementation generated at runtime by mocking framework) |
| Dummy | A null or dummy object to be used as parameter value when test doesn’t care |

## Stub

```
public class TestStockPriceService implements StockPriceService {

	@Override
	public double fetchPriceForStock(String StockSymbol) {
		return 100;
	}
}
```

# Mock

```
public interface StockPriceService {

        double fetchPriceForStock(String StockSymbol);
}

…

@Test
public void tradeShouldUseLatestStockPrice() {
        String stockSymbol = "BP";

        StockPriceService mockStockPriceService = mock(StockPriceService.class);

        Trade trade = new Trade(mockStockPriceService, stockSymbol, 50);

        when(mockStockPriceService.fetchPriceForStock(stockSymbol)).thenReturn(0.01);

        assertEquals(0.5, trade.calculateCurrentTradePrice(), 0);

        verify(mockStockPriceService).fetchPriceForStock(stockSymbol);
}
```

codemanship

# Example #4 – JG Holidays Ltd

Write some code that will tell us what villas are available on weeks where flights are also available

Use stubs to act as facades to external systems.

1. Villa booking system – tell it the week and year (e.g. week 26, 2011) and it will return an array of villa names that are available
2. Flight booking system – tell it the week and the year and the start/destination airports (e.g., London Heathrow (LHR) -> Paphos International (PFO)) and it will tell you if flights are available in that week

Putting It All Together

## END-TO-END TDD

Where do we start?

## USERS & THEIR GOALS

## Donate a DVD

As a video library **member**, I want to **donate a DVD to the library** so that other members can borrow it and I can earn points for priority services

codemanship

What’s the outcome?

## TEST-DRIVEN

codemanship

Given a copy of a DVD title that *isn't in the library,*

When a member donates their copy, specifying the name of the DVD title

Then that title is added to the library *and* their copy is registered against that title so that other members can borrow it,
**AND** an email alert is sent to members who specified an interest in matching titles,
**AND** the new title is added to the list of new titles for the next member newsletter
**AND** the member is awarded priority points

codemanship

# Examples: Be Specific…

<u>Given</u> a copy of The Abyss, which isn't in the library,

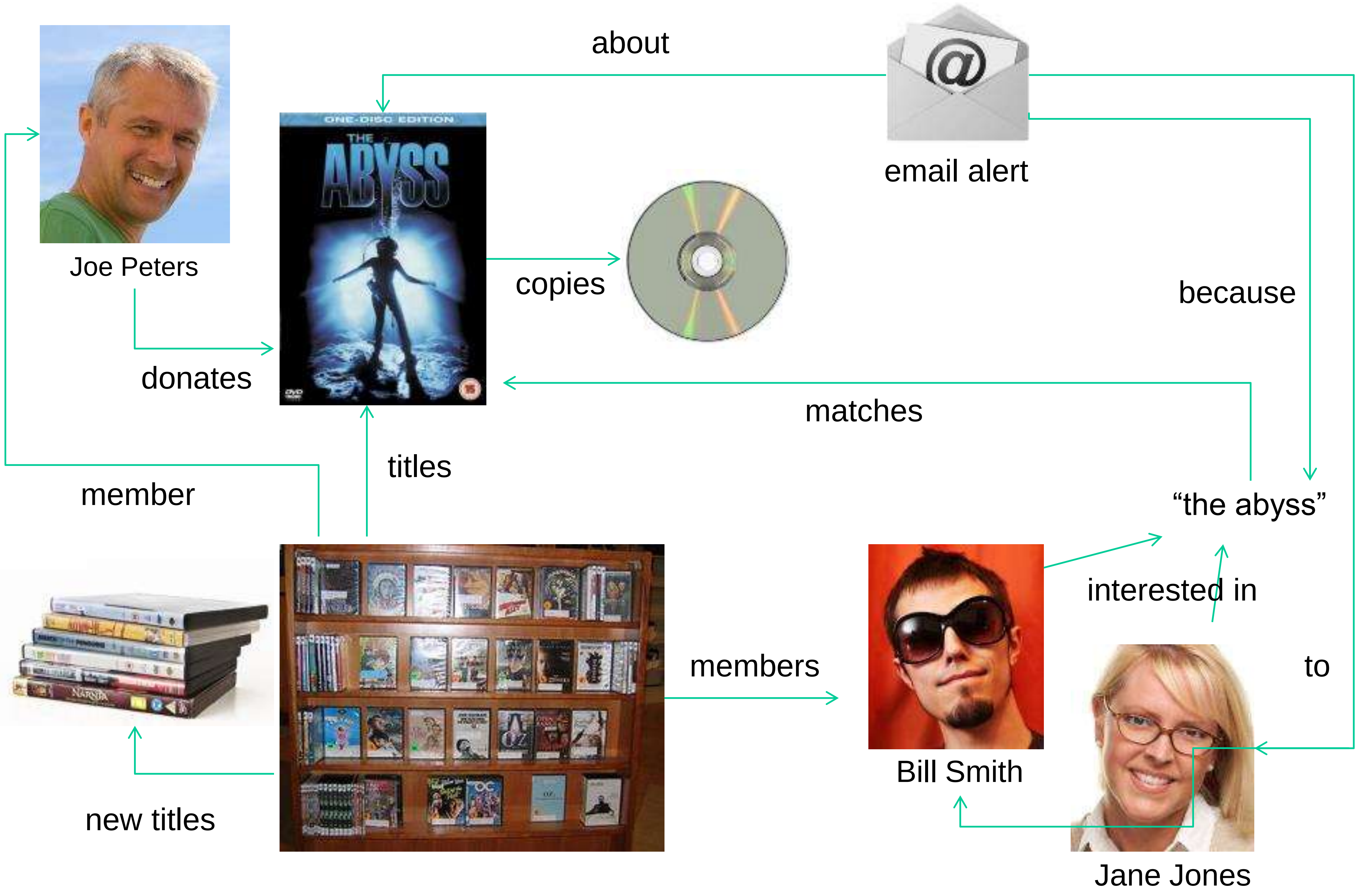
<u>When</u> Joe Peters donates his copy, specifying the name of the title, that it was directed by James Cameron and released in 1989

<u>Then</u> The Abyss is added to the library and his copy is registered against that title so that other members can borrow it,
**AND** an email alert with the subject "New DVD title" is sent to Bill Smith and Jane Jones, who specified an interest in titles matching "the abyss"(non-case-sensitive), stating "Dear <member's first name>, Just to let you know that another member has recently donated a copy of The Abyss (dir: James Cameron, 1989) to the library, and it is now available to borrow."
**AND** The Abyss is added to the list of new titles for the next member newsletter
**AND** Joe Peters receives 10 priority points for making a donation

codemanship

Who are the characters, and how do they know each other?

## OBJECT ORIENTED



codemanship

What's the story?

## DESIGN

## Responsibilities: Passive Story

1. “The Abyss” is added to the library with title name “The Abyss”, director “James Cameron” and year 1989
2. One rental copy is registered to “The Abyss”
3. An email is sent to any members who specified that they wanted to be alerted when a title matching “the abyss” (non-case sensitive) was donated
4. “The Abyss” is added to the list of new titles
5. Joe Peters is awarded 10 priority points

# Roles: Active Story

1. The **library** adds “The Abyss” to itself with title name “The Abyss”, director “James Cameron” and year 1989
2. The new **title** “The Abyss” registers one rental copy to itself
3. An **email alert** sends itself to any members who specified that they wanted to be alerted when a title matching “the abyss” (non-case sensitive) was donated
4. The **library** adds “The Abyss” to the list of new titles
5. **Member** Joe Peters awards himself 10 priority points

codemanship

## Collaborations: Characters Interact

1. The **library** adds “The Abyss” to itself with title name “The Abyss”, director “James Cameron” and year 1989, then tells
2. the new **title** “The Abyss” to register one rental copy to itself, who tells
3. an **email alert** to send itself to any members who specified that they wanted to be alerted when a title matching “the abyss” (non-case sensitive) was donated
4. The **library** adds “The Abyss” to the list of new titles, then tells
5. **member** Joe Peters to award himself 10 priority points

## Class-Responsibility-Collaboration (CRC) Cards

| Library | |
|---|---|
| • Knows about titles<br>• Knows about new titles<br>• Adds donated titles<br>• Adds new titles | • Title<br>• Member |

| Title | |
|---|---|
| • Knows its name, director & year of release<br>• Knows about rental copies<br>• Registers rental copy | • Email Alert |

| Email Alert | |
|---|---|
| • Sends email to members who specified matching title | ? |

| Member | |
|---|---|
| • Knows about priority points<br>• Awards priority points | |

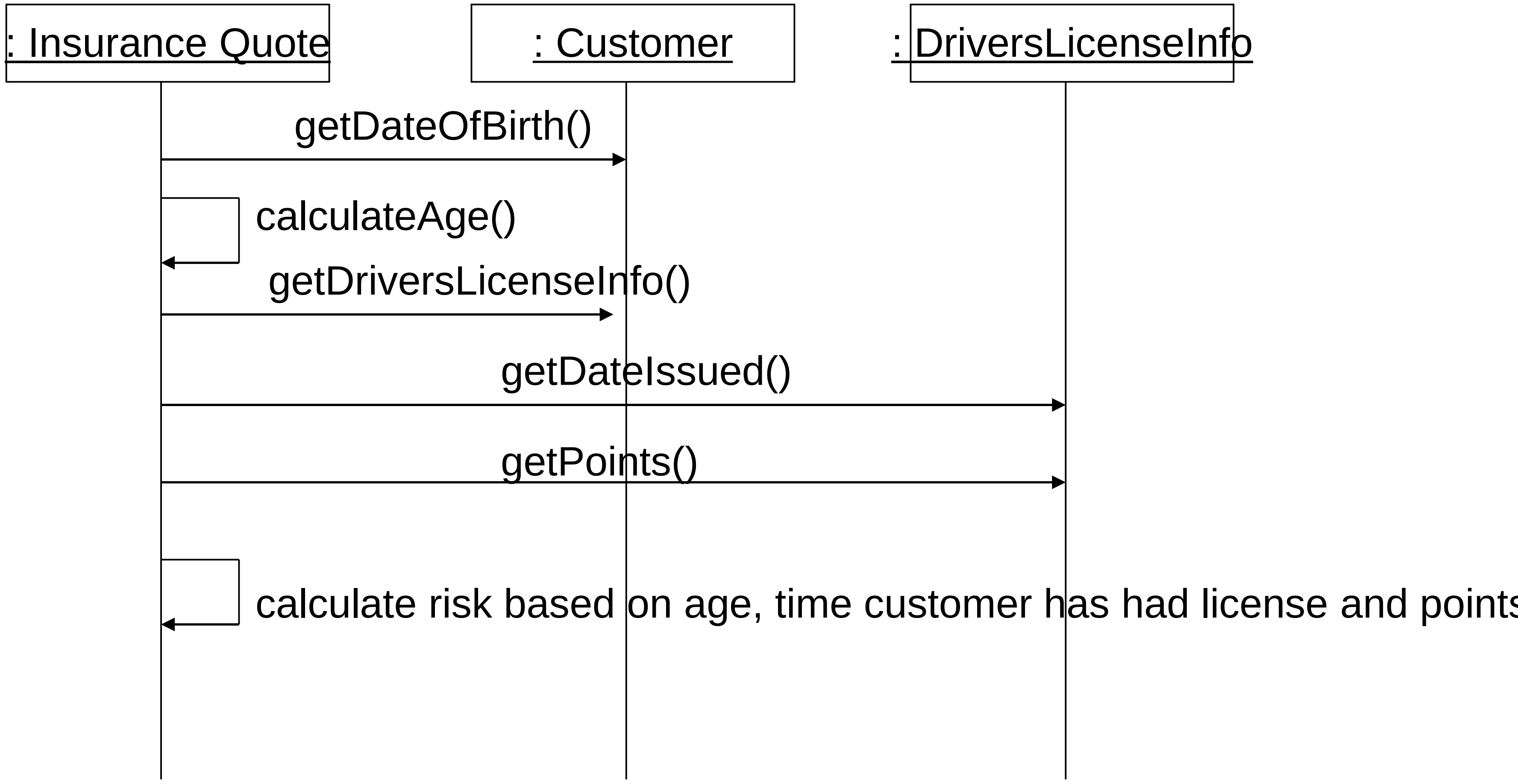
codemanship

Put the behaviour where the data is

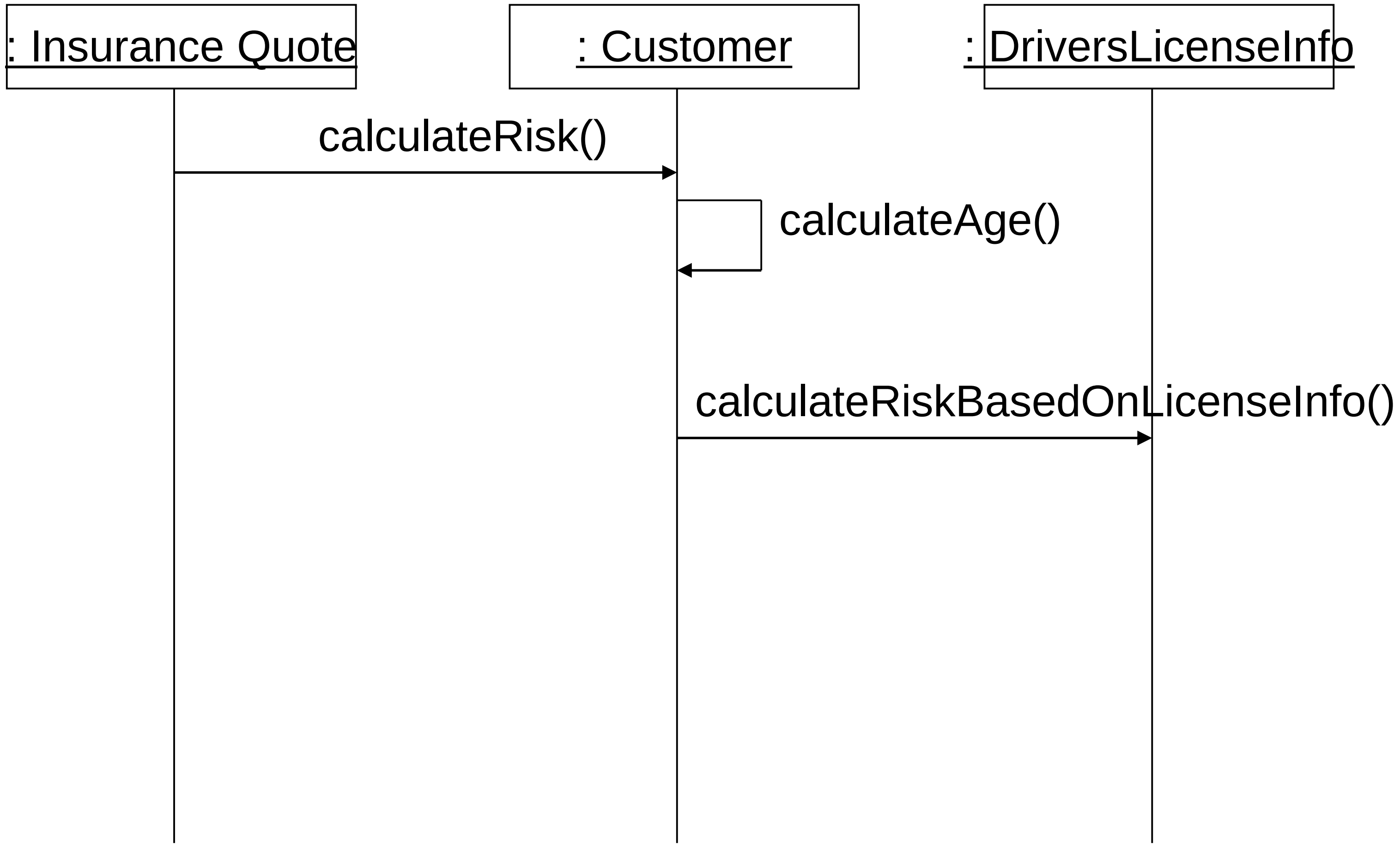
## ENCAPSULATION

codemanship

# Data-driven Design



# Tell, Don't Ask



Classic TDD + London School

## TESTS!

| Title | |
|---|---|
| • Knows its name, director & year of release<br>• Knows about rental copies<br>• Registers rental copy | • Email Alert |

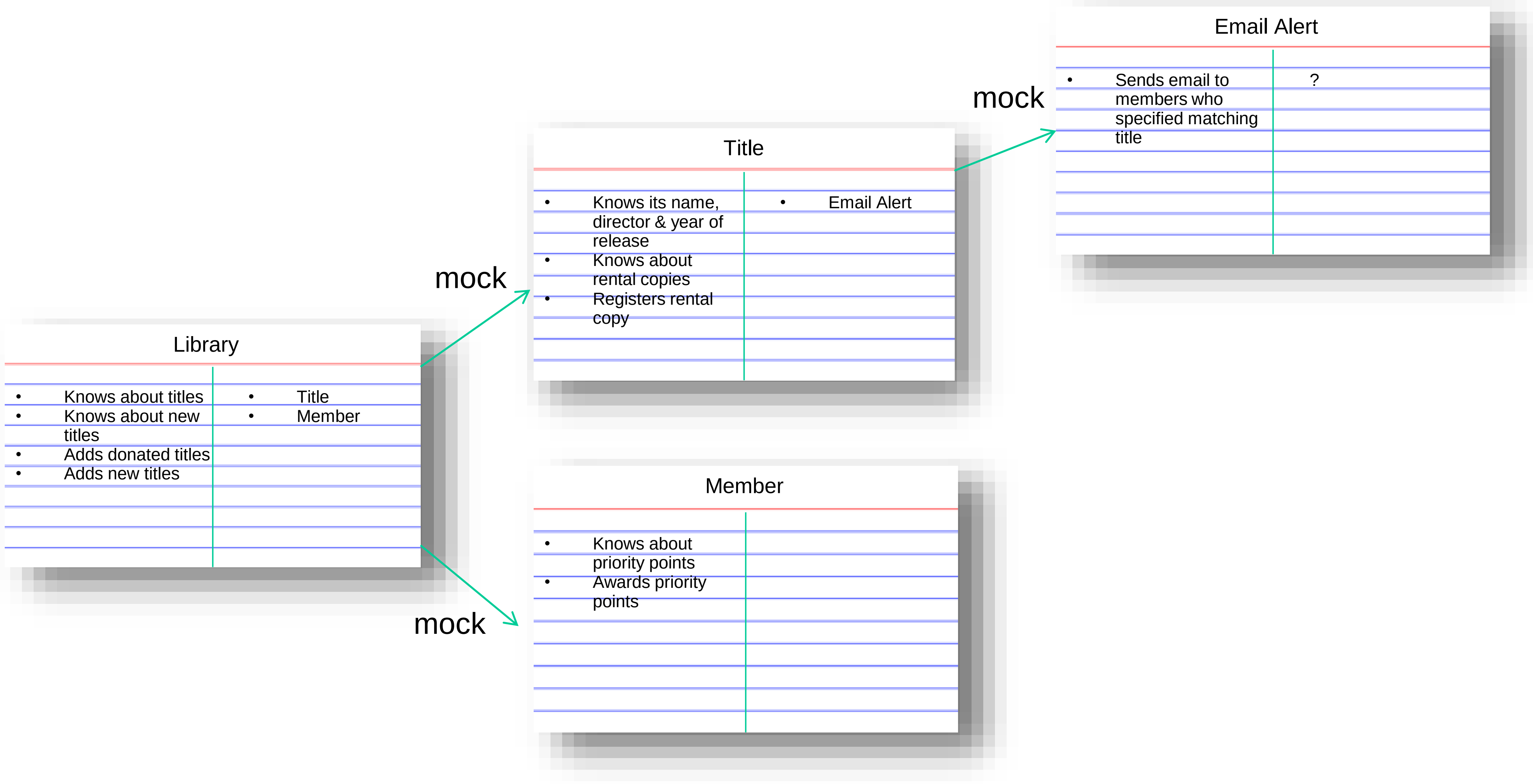
```
@Test
public void registersRentalCopy() {
    EmailAlert emailAlert = mock(EmailAlert.class);
    Title title = new Title(null, null, null, emailAlert);
    title.registerCopy();
    assertEquals(1, title.getRentalCopyCount());
}
```

```
@Test
public void tellsEmailAlertToSend() {
    EmailAlert emailAlert = mock(EmailAlert.class);
    Title title = new Title(null, null, null, emailAlert);
    title.registerCopy();
    verify(emailAlert).send(title);
}
```

Gluing It All Together From The

## OUTSIDE-IN

codemanship



codemanship

Getting Into The Habit

## TDD PRACTICE REGIMES

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Write a failing test | | | | | | | | | |
| Write the simplest code to pass the test | | | | | | | | | |
| Refactor to remove duplication | | | | | | | | | |
| Write the assertion first and work backwards | | | | | | | | | |
| See the test fail | | | | | | | | | |
| Write meaningful tests | | | | | | | | | |
| Triangulate | | | | | | | | | |
| Keep your test and model code separate | | | | | | | | | |
| Isolate your tests | | | | | | | | | |
| Organise your tests to reflect model code | | | | | | | | | |
| Maintain your tests | | | | | | | | | |
| Tests should test one thing | | | | | | | | | |
| Don’t refactor with a failing test | | | | | | | | | |