Working with Legacy Code

Agenda

1. What is Legacy Code
2. What is the problem with it
3. Techniques to deal with it
4. Exercise


Java 2, XHTML... we can toss all these out
I think we should keep this one...
When is it from?
2004
Are you kidding? It's 15 years old! We can toss it!
I don't know, it might be one of the only books that gets better with age
I don't know...
WORKING EFFECTIVELY WITH LEGACY CODE
CommitStrip.com

What is Legacy Code?

Legacy code can be

- Code without tests
- Code you’re not comfortable changing
- Code that is hard to understand

Legacy code can be

- Code without tests
- Code you’re not comfortable changing
- Code that is hard to understand
- Code that is vital to the business

What is the problem?

Problem(s)?

- Afraid of making changes

Problem(s)?

- Afraid of making changes
- Understanding the code base takes considerable time

Problem(s)?

- Afraid of making changes
- Understanding the code base takes considerable time
- Writing a test (if possible) takes time

Problem(s)?

- Afraid of making changes
- Understanding the code base takes considerable time
- Writing a test (if possible) takes time
- Extremely late feedback

The Legacy Code dilemma.

In order to make our changes we need to add tests in order to do our changes.

The Legacy Code dilemma.

In order to make our changes we need to add tests in order to do our changes.

In order to add tests we need to change the code before.

The Legacy Code dilemma.

In order to make our changes we need to add tests in order to do our changes.

In order to add tests we need to change the code before.

Why?

- Some of that code **might be difficult to test** or out right untestable.
- **Any change** in the code, even for the right reasons, **might cause a regression** somewhere unknown.

Techniques to deal with it

Characterization tests

What

Tests that characterize the actual behavior of a piece of code.

What

Tests that characterize the actual behavior of a piece of code.

Benefits

- Quickly build high coverage.
- Safety net for refactoring.

How

- Create random inputs
- Bombard the system under test
- Capture the outputs

How

- Create random inputs
- Bombard the system under test
- Capture the outputs

Golden Master

How

- Create random inputs
- Bombard the system under test
- Capture the outputs

Golden Master

Or ....

- Use a library like https://approvaltests.com/

How

- Create random inputs
- Bombard the system under test
- Capture the outputs

Or ....

- Use a library like https://approvaltests.com/

Change as little code as possible to get tests in place

Dependency breaking

What

Changing hard dependencies for loose ones

How

- Identify change points (Seams)

Seams

A place to alter the program behavior, without changing the code.

Seams

A place to alter the program behavior, without changing the code.

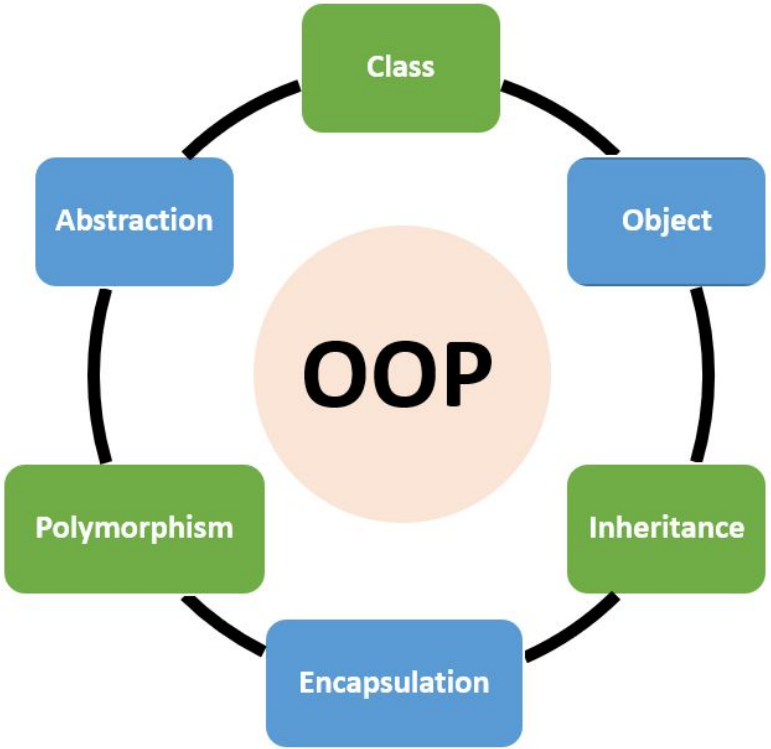
- Static calls
- In-method object creation

What

Changing hard dependencies for loose ones

How

Leverage OOP



How

Create new Sub Classes specifically for testing purposes

```JS
export class DatabaseConnector {
  // A lot of code…

  connect() {
    // Perform some calls to connect to the DB.
  }
}
```

```JS
class FakeDatabaseConnector extends DatabaseConnector {
  connect() {
    // Override the problematic calls to the DB
    console.log("Connect to the DB")
  }
}
```

Sprout & wrap

What

Instead of changing existing (untested) code, write new (tested) code and then have the old code call the new one.

Sprout

- Create your code somewhere else
- Unit test it
- Identify where you should call that code from the existing code: The insertion point,
- Call your code from the Legacy Code

Wrap

- Rename the old method you want to wrap
- Create a new method with the same name and signature
- Call the old method from the new one
- Put the new logic before/after the old method call

Scratch refactoring (exploratory)

What

Throwaway refactors

How

- Play with the code as much as you want.
- Extract functions, simplify code, rename variables, etc.
- Once you do, revert and start over with proper test
- Revert your changes when you’re done

Wrap & Own dependencies

What

Build your own layer on top of third party code

What

Build your own layer on top of third party code

Benefits

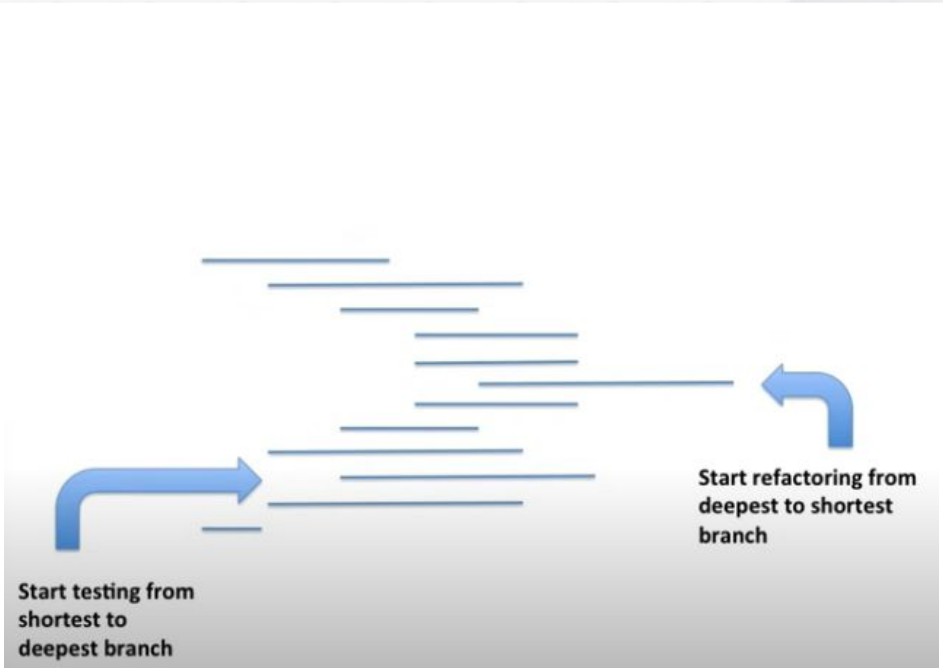
- Decreased coupling
- Isolated integration with external elements
- Great place for Seams

Guidelines

- Start testing from the shortest to deepest branch

- Start testing from the shortest to deepest branch
- Start refactoring from deepest to shortest branch

- Start testing from the shortest to deepest branch
- Start refactoring from deepest to shortest branch



Exercise N° 1

Trip Service Kata

- https://github.com/sandromancuso/trip-service-kata

Exercise N° 2 (Day 2)

Gilded Rose

- https://katalyst.codurance.com/gilded-rose

Thank you

If you have any questions,
please get in touch.