# TDD Practices

Sang Shin
JPassion.com
"Code with Passion!"

# Topics

- Characteristics of good tests
- What is and why TDD?
- TDD development cycle
- TDD best practices
- TDD anti-patterns
- Testable code best practices

# Characteristics of Good Tests

# Characters of Good Tests

- Run fast
  - Short setup, run time, tear down
- Run in isolation
  - Tests should not rely on data or state created/modified by other tests
  - Tests should be able to be reordered
- Use data that makes them easy to read and understand
- Use real data whenever possible
  - Use copies of production data if possible
- Repeatable
  - Should return same result each time it runs

# Goals for Well-Written Tests

- Readable by others
- Should serve as sample client code for others
- Should serve as a specification (document)
- Should test what they're supposed to test (i.e. that the requirements have been met) and no more
- Not tightly coupled to the target code
- Should follow good coding principles (single responsibility, etc.)

# What is and Why TDD?

# What is TDD?

- Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle of Red, Green, Refactor
  - As opposed to writing a big chunk of code at a time
- Requirements are turned into very specific test cases, then the software is improved to pass the new tests, only
  - As opposed to writing software beyond the requirements

# Why TDD?

- It helps you focus on design
  - "The act of writing a unit test is more an act of design than of verification.  It is also more an act of documentation than of verification" (Uncle Bob)
- It helps you to think from customer requirements
  - The act of writing a unit test closes a remarkable number of feedback loops
- It helps you to build better written better tested software
  - It helps you to write testable code

# TDD Development Cycle

# TDD Cycle: Red-Green-Refactor

- Repeat the Red-Green-Refactor cycle
  - Add failing test (Red)
  - Write the code to correct the test failure (Green)
  - Refactor code (Refactor)
- For each cycle
  - Prefer to add a few new lines of functional code, typically less than ten, before rerunning tests
  - Prefer to limit the TDD cycle to be less than ~5 minutes (could be longer as you become more fluent in TDD)

# TDD Best Practices

# TDD Best Practices

- Only specify what is important
- Make small steps
  - Solve things as simply as possible
  - Follow 2 to 5 minute rule – each TDD cycle should not take more than ~2 to ~5 minutes
- Perform refactoring after each passing test
  - Don't lose the best opportunity to refactor
- Make testing code expressive and readable
  - Testing should serve as pseudo-documentation
  - Example: should_result_this_when_xyz_occurs()

# Anti-Patterns

# Examples of Anti-Patterns

- The free ride
  - Rather than write a new test method for another feature, a new assertion rides along an existing test
- The loud mouth
  - Clutters up the console with diagnostic messages, logging messages, and other chatter, even when passing
- The stranger
  - A test case that doesn't even belong in the unit tests it's part of. It's really testing a separate object
- The local hero
  - A test case that depends on something specific to the development environment, passes on one machine, fails on another

# Writing Testable Code Best Practices

# Writing Testable Code

- Do not use Singleton class
  - It is impossible to mock a Singleton
- Get dependencies to be injected
  - Is it impossible to mock the internally created dependencies

# Lab:

## Exercise 1: String Calculator
## Exercise 2:  Movie Rental Application
## 1655_tdd_practices.zip

# Code with Passion!
# JPassion.com