Legacy Refactoring

Because it's fun

Version 2.3.4.3

Resources

- https://vimeo.com/144945100
  - A version of this talk
  - So you can enjoy it again

- https://vimeo.com/145232044
  - Resettable singleton

- https://github.com/schuchert/welc_examples_java_jmockit
  - The github repo so you can try it yourself

Design, Design, Design

- Here's a staring list to help with OOD

GRASP	Craig Larman
SOLID	Robert Martin
CODE SMELLS	Martin Fowler
WELC	Michael Feathers
TEST DOUBLES	Several
CODING KATAS	Several
DESIGN PATTERNS	Gang of 4

- http://schuchert.wikispaces.com/TddIsNotEnough

GRASP

Information Expert	Assign responsibility to the thing that has the information.
Controller	Assign system operations (events) to a non-UI class. May be system-wide, use case driven or for a layer.
Low Coupling	Try to keep the number of connections small. Prefer coupling to stable abstractions.
High Cohesion	Keep focus. The behaviors of a thing should be related. Alternatively, clients should use all or most parts of an API.
Polymorphism	Where there are variations in type, assign responsibility to the types (hierarchy) rather than determine behavior externally,
Pure Fabrication	Create a class that does not come from the domain to assist in maintaining high cohesion and low coupling.
Protected Variations	Protect things by finding the change points and wrapping them behind an interface. Use polymorphism to introduce variance.

SOLID Principles

- http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

S	**Single Responsibility**	Single Reason to Change
O	**Open/Closed**	Open for extension closed to change
L	**Liskov Substitution**	Derived types substitutable for base types
I	**Interface Segregation**	Interfaces should be focused (small) & client specific
D	**Dependency Inversion**	Dependencies should go from concrete to abstract

Package Cohesion/Coupling

- Guidelines for package cohesion

REP	Release/Reuse Equivalency	What you release is what you reuse.
CCP	Common Closure	Classes that change together should be packaged together
CRP	Common Reuse	Classes that are used together should be packaged together

- Guidelines for package coupling

ADP	Acyclic Dependencies	No cycles in your dependencies
SDP	Stable Dependencies	Dependencies should go from less to more stable. Depend on stable things
SAP	Stable Abstractions	Abstraction increase with stability

A Few Code Smells

- A few of Martin's code smells:

Poor Names	Name suggests wrong intent
Long methods	More than 1 thing/multiple levels of abstraction
Large classes	More than one concept/multiple levels of abstraction
Long parameter list	Too many arguments to keep straight (> 3)
Duplicated code	Same or similar code appears in more than one place
Divergent change	The class/method changes for dissimilar reasons
Shotgun Surgery	Single change affects multiple classes/methods
Feature Envy	One class uses another class' members
Switch Statements	Duplicated switches/if-else's over same criterion

- http://c2.com/cgi/wiki?CodeSmell

Some Legacy Refactorings

- From Working Effectively with Legacy Code

Adapt Parameter	326	Change parameter to an adapter when you cannot use extract interface
Break Out Method Object	330	Convert method using instance data into a class with a ctor and single method
Encapsulate Global References	339	Move access to global data into access via a class to allow for variations during test
Extract and Override Call	348	Turn chunk of code into overridable method and then subclass in test
Extract and Override Getter	352	Turn references into hard-coded object into call to getter and then subclass
Extract Interface	362	Extract interface for concrete class, then use interface. Override in test.
Introduce Instance Delegator	317	Add instance methods calling static methods. Call through instance, which test subclasses.
Parameterize Constructor **Parameterize Method**	379 383	Examples of Inversion of Control (IoC)
Subclass and override method	401	Test creates subclass & passes it in/requires some IoC
Sprout Method **Sprout Class**	59 63	Create a method or class out of existing code.

Test Doubles

- Gerard Meszaros
  http://xunitpatterns.com/Test%20Double%20Patterns.html

DUMMY	Empty implementation. Not called or don't care if it is
STUB	Canned replies – "snapshot in time"
SPY	Watches and Records
FAKE	Partial Simulator
MOCK	Has & Validates expectations
SABOTEUR	Designed to always fail, e.g., always throws an exception.

Design Patterns

- From: Design Patterns: Elements of Reusable Object-Oriented Software

Strategy	Define a function or algorithm as a class. Form a wide but shallow hierarchy of different algorithms.
Template Method	Write an algorithm in a base class with extension points represented as abstract methods. Subclass and override.
Abstract Factory	A base interface for creating one or a family of objects through a standard API. Create implementations for each family of objects that need to be created.
Composite	A class that implements some other interface and also holds onto zero or more instances of that same interface.
State	Similar to strategy, though the states are interdependent. States can cause a so-called context to change from one state to another during its lifetime.

Additional Resources

- Video Series

C++	Dice Game	http://vimeo.com/album/254486
C#	Shunting Yard	http://vimeo.com/album/210446
JAVA	Rpn Calculator	http://vimeo.com/album/205252
iPHONE	iPhone & TDD	http://vimeo.com/album/1472322

- Mocking

JAVA	Mockito	http://schuchert.wikispaces.com/Mockito.LoginServiceExample
C#	Moq	http://schuchert.wikispaces.com/Moq.Logging+In+Example+Implemented

- Other

JAVA	FitNesse	http://schuchert.wikispaces.com/FitNesse.Tutorials
RUBY	Several	http://schuchert.wikispaces.com/ruby.Tutorials
JAVA	UI	http://schuchert.wikispaces.com/tdd.Refactoring.UiExample

Thank

Fin.

You!