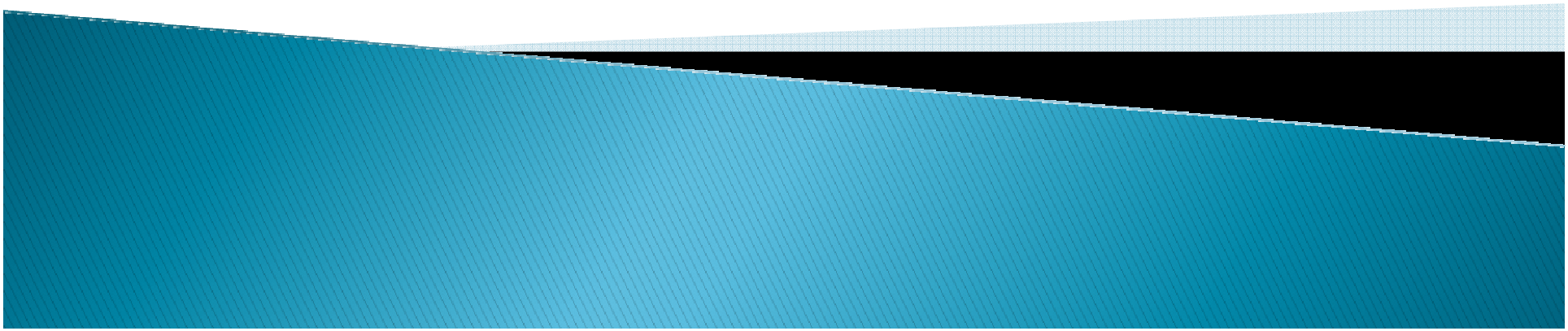
Analysis Assignment 3: Mining Design Patterns

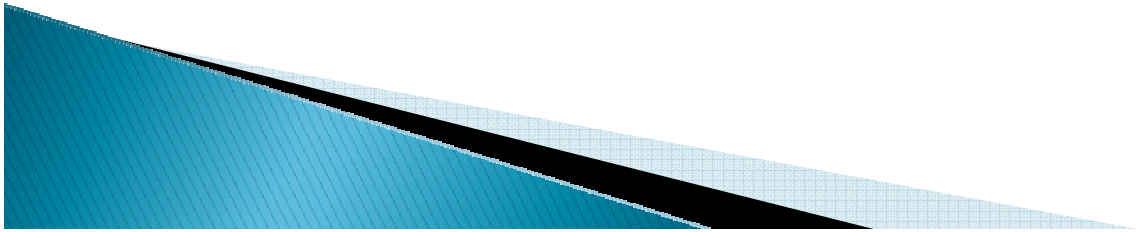
Square Root Team
(Group 14)



SQUARE ROOT

NUnit Project Context

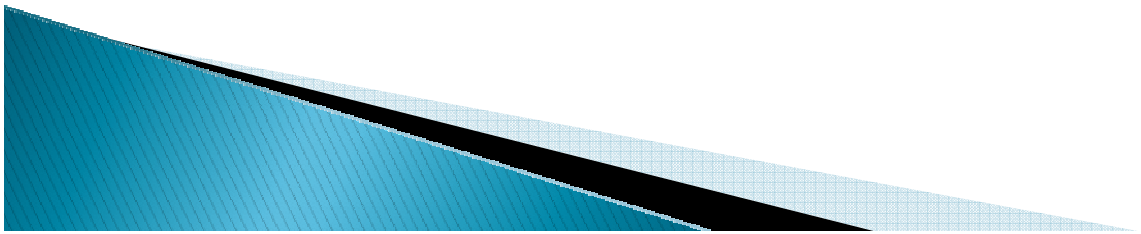
- De facto .Net unit testing framework
  - JUnit's .Net cousin
- Size
  - Core, Extensions, Fixtures, Framework, Mocks
    - About 400 classes
  - GUI, Command Line UI,
    - Over 300 classes



SQUARE ROOT

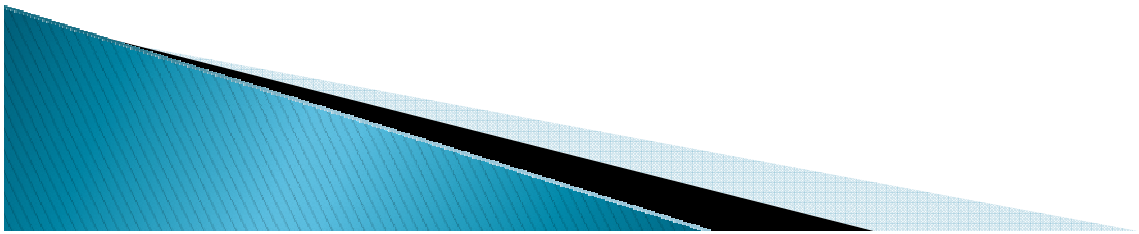
Approach to Identifying Patterns

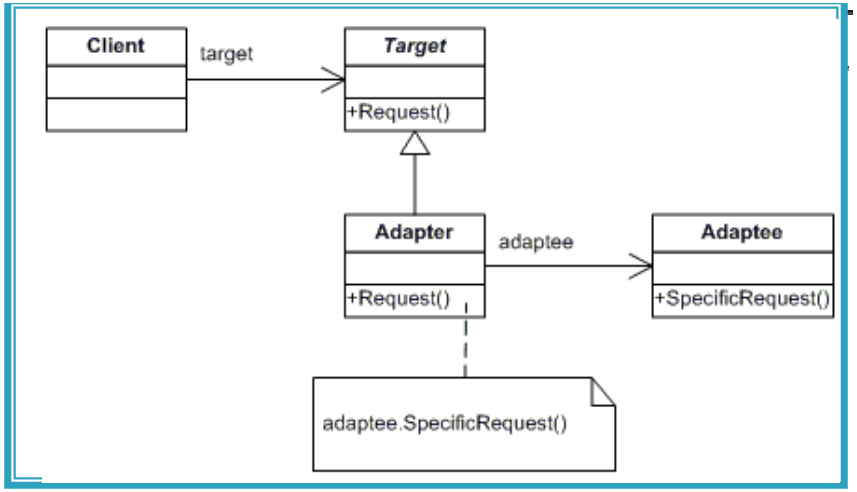
- "Classic" Implementations
  - By the book example scenarios
- Generate and examine UML in Visual Studio
- Word search
  - Still necessary to trace code, verify pattern
  - Example: fake singleton
- Examine pattern of each purpose
  - Creational
  - Structural
  - Behavioral



Adapter Pattern: Context of Use

- TextDisplayTabPage
  - Control used by NUnit GUI
  - Holds reference to adapter
  - Renders tabbed text boxes
- Deviation from “pure” adapter pattern
  - Client uses an object variable of type adapter, rather than using one of type target
- Adapting regular textbox to custom textbox for GUI





ADAPTER

ADAPTEE

CLIENT

TARGET

TabPage
Class
Panel

TextBoxDisplay
Class
RichTextBox
Fields
Properties
Methods
ContextMenu_...
copyMenuItem...
GetText
OnTestOutput
OnTestStarting
selectAllMenuIt...
Subscribe
TextBoxDisplay
WantOutputType
wordWrapMen...
Write (+ 1 overl...
WriteLine

RichTextBox
Class
TextBoxBase

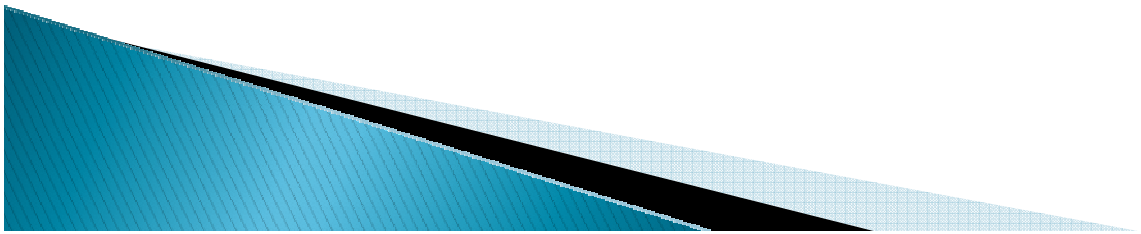
TextDisplayTabPage
Class
TabPage
Fields
display
Properties
Display
Methods
TextDisplayTabPage (+...

TextDisplay
TestObserver

TextDisplay
Interface
TestObserver
Properties
Methods
Clear
GetText
Write (+ 1 overloa...
WriteLine

Adapter Pattern: Analysis

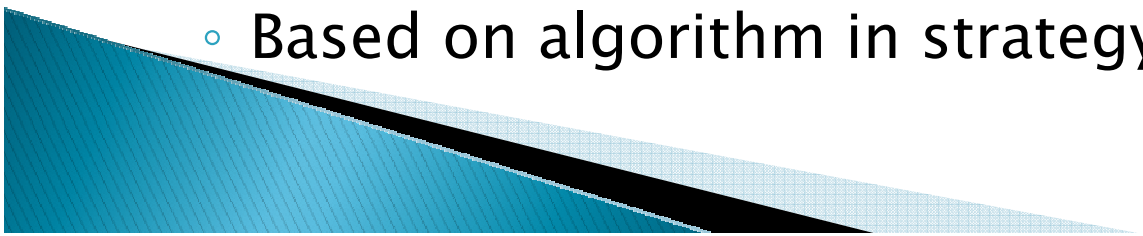
- Advantages
  - Usability – RichTextBox functionality can be used by any class that uses adapter without knowledge of internal details
  - Modifiability – possibility of side effects reduced when changing the Adaptee.
- Disadvantages
  - Maintainability – difficult to see what objects can adapt into other patterns

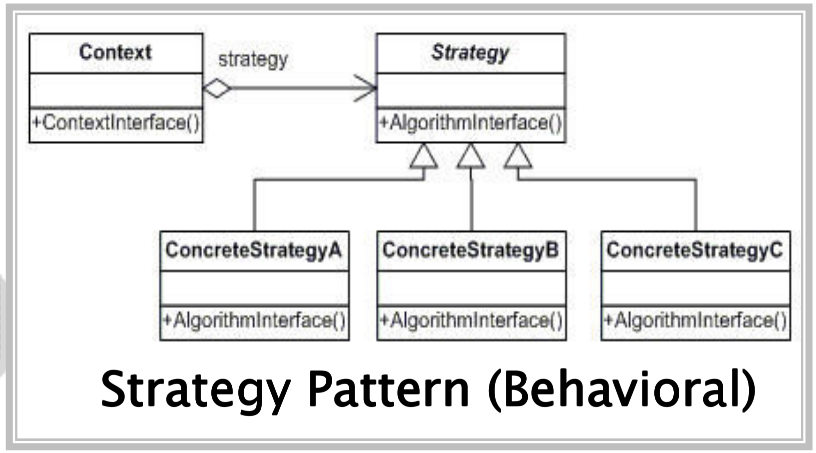


SQUARE ROOT

Strategy Pattern: Context of Use

- Failure messages shown to the user
  - Messages have different detail depending on the constraint tested
- Constraints are used to test objects
- Different kinds of constraints
  - Depends on object type and test
  - Example: empty collection constraints for collections
- TextMessageWriter writes different messages to client
  - Based on algorithm in strategy classes

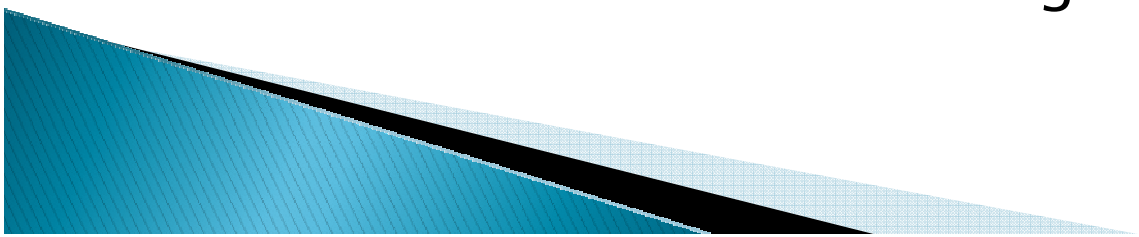



Context
strategy
Strategy
+ContextInterface()
+AlgorithmInterface()
ConcreteStrategyA
ConcreteStrategyB
ConcreteStrategyC
+AlgorithmInterface()
+AlgorithmInterface()
+AlgorithmInterface()
Strategy Pattern (Behavioral)

Abstract Class
StringWriter
TextMessageWriter
Class
MessageWriter
Context
Strategy
Abstract Class
Abstract Class
Constraint
Methods
Matches
Nested Types
Strategy
Concrete Strategy 1
Concrete Strategy 5
Class
CollectionConstraint
Methods
doMatch
WriteDescriptionTo
Fields
expected
CollectionContai…
CollectionEquiva…
CollectionSubset…

Strategy Pattern: Analysis

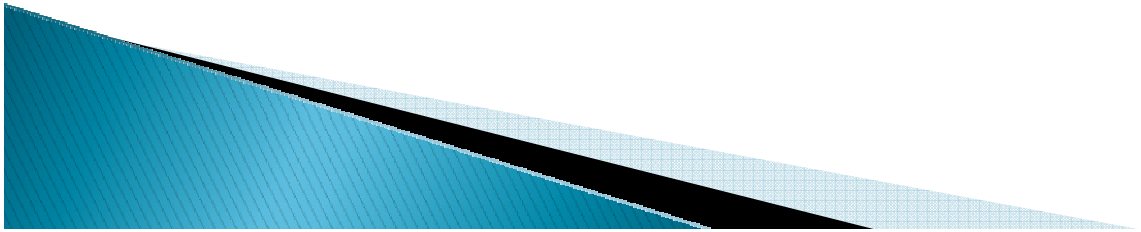
- Advantages
  - Modifiability – TextMessageWriter can use the same Constraint interface to write different descriptions
  - Modifiability – change behavior of constraints without changing the interface
  - Reuse – Constraint objects can be shared with other Context objects
- Disadvantages
  - Not all functionality exists in concrete strategy classes
    - Empty Collection Constraint –matches() method not implemented
  - Performance – Lots of strategies take up memory

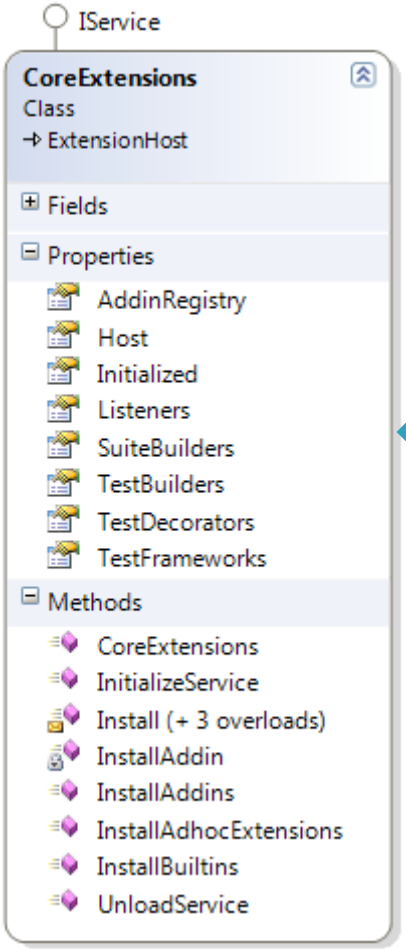


SQUARE ROOT

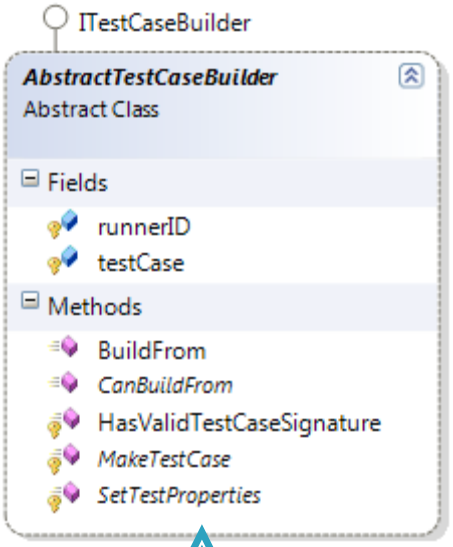
Builder Pattern: Context of Use

- NUnit executes a test suite
  - Test Suites are composed of test cases
- Extensible TestCase initialization
  - Optional Setup, Teardown methods
  - TestCases of different types
  - Complex object
- Client runs test suite
- CoreExtensions
  - Abstracts TestCase initialization details from client

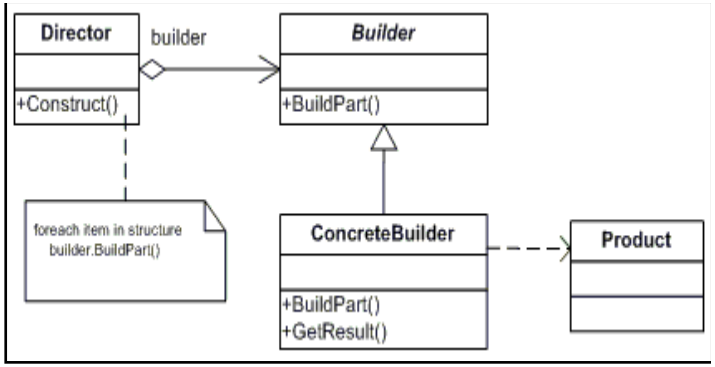




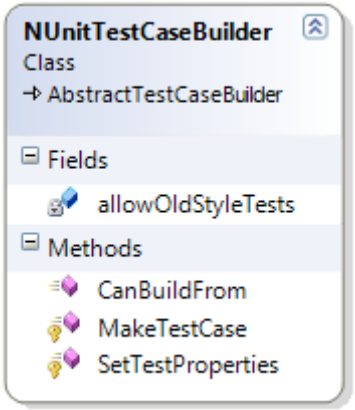
Director



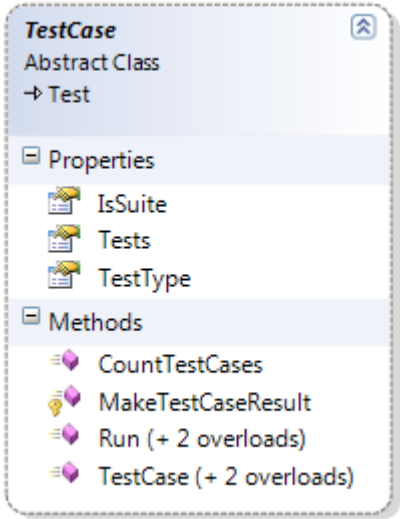
Builder



Builder Pattern(Creational)



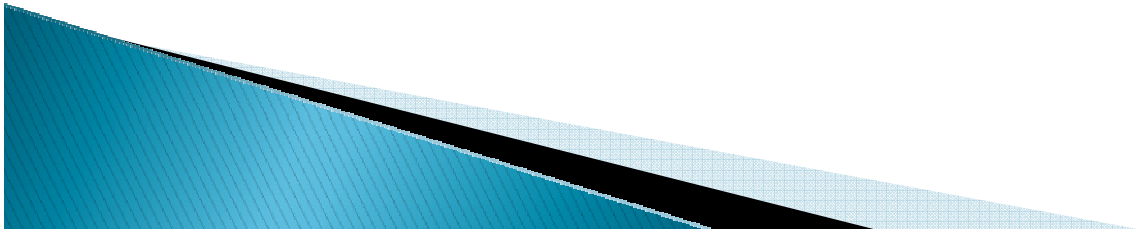
Concrete Builder



Product

Builder Pattern: Analysis

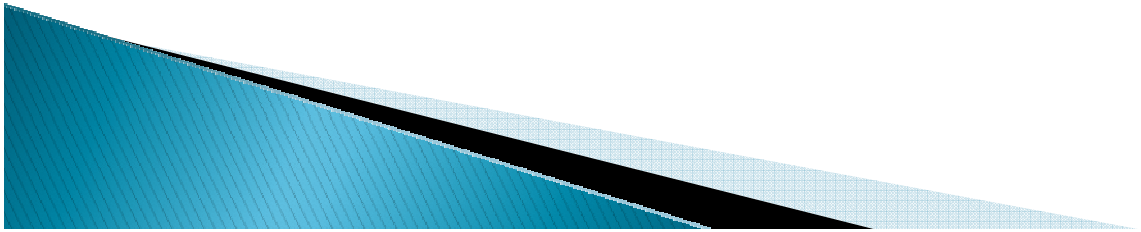
- Advantages
  - Modifiability – adding new types of Test Cases is easy, add new Concrete Builders
  - Maintainability – reduce probability of side effects when changing the client
  - Maintainability – code is easier to read
- Disadvantages
  - Performance – many intermediary objects instantiated



SQUARE ROOT

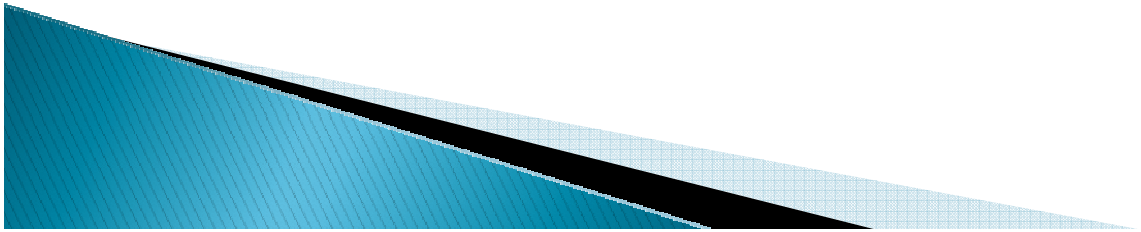
Bonus: NUnit vs. JUnit

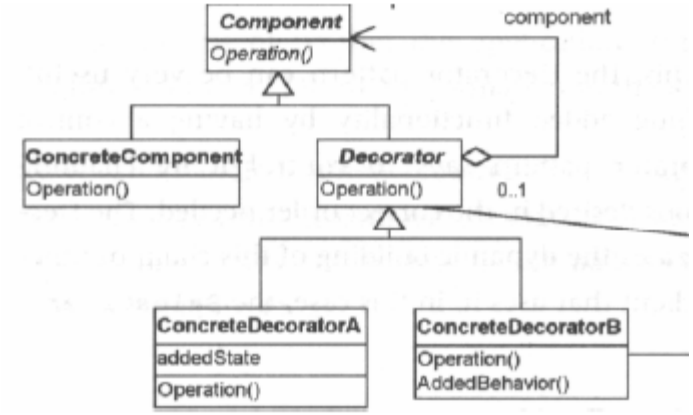
- Cousin frameworks
  - Similar languages
  - Similar functionality
  - Similar purpose
- What does the design look like?
  - Our theory: NUnit heavily inspired by JUnit, design should be similar
- Examine test suite creation



Decorator Pattern: Context of Use with JUnit

- Decorator used to decorate multi-threaded test-execution
- *TestSuite* is the decorator that runs tests
  - Calls *test* interface's run method
  - *ActiveTestSuite* decorates *test* with its own run method to run multiple tests simultaneously
  - Monitors running tests until completion
- Can be extended like in NUnit
  - Setup, teardown





Decorator

Concrete Component

Component

Concrete Decorator

TestSuite
+createTest(theClass : Class<? extends TestCase>, name : String) : ...
+getTestConstructor(theClass : Class<? extends TestCase>) : Constr...
+warning(message : String) : Test
-exceptionToString(t : Throwable) : String
+TestSuite()
+TestSuite(theClass : Class<? extends TestCase>)
+TestSuite(theClass : Class<? extends TestCase>, name : String)
+TestSuite(name : String)
+TestSuite(classes : Class<?> ...)
+TestSuite(classes : Class<? extends TestCase> [], name : String)
+addTest(test : Test) : void
+addTestSuite(testClass : Class<? extends TestCase>) : void
+countTestCases() : int
+getName() : String
+run(result : TestResult) : void
+runTest(test : Test, result : TestResult) : void
+setName(name : String) : void
+testAt(index : int) : Test
+testCount() : int
+tests() : Enumeration<Test>
+toString() : String
-addTestMethod(m : Method, names : List<String>, theClass : Class<...
-isPublicTestMethod(m : Method) : boolean
-isTestMethod(m : Method) : boolean

-fTests
*

TestCase
-fName : String
+TestCase()
+TestCase(name : String)
+countTestCases() : int
+run(result : TestResult) :...
+runBare() : void
+toString() : String
+run() : TestResult
+getName() : String
+setName(name : String) ...

<<Interface>>
Test
+countTestCases() : int
+run(result : TestResult) : v...

ActiveTestSuite
-fActiveTestDeathCount : int
+ActiveTestSuite()
+ActiveTestSuite(theClass : Class<? extends TestCase>)
+ActiveTestSuite(name : String)
+ActiveTestSuite(theClass : Class<? extends TestCase...
+run(result : TestResult) : void
+runTest(test : Test, result : TestResult) : void
+runFinished() : void

Decorator vs. Builder

- Advantages of Decorator
  - Adds behavior before and after a test execution, without changing the actual test-execution method.
  - The decorated component (Test) can be redecorated, or new decorations(setup/teardown) can be added to this.
  - A builder multiplies the number of testsuites if there are several combinations of pre and post tasks

Disadvantage of Decorator

  - If there are not many combinations of pre and post tasks, a builder pattern is going to be more appropriate

