15-654: ANALYSIS OF SOFTWARE ARTIFACTS

ASSIGNMENT 3: MINING DESIGN PATTERNS

TEAM DIVERSITY

February 3, 2009

Majid Alfifi ● Vishal Dwivedi ● Brian Krausz

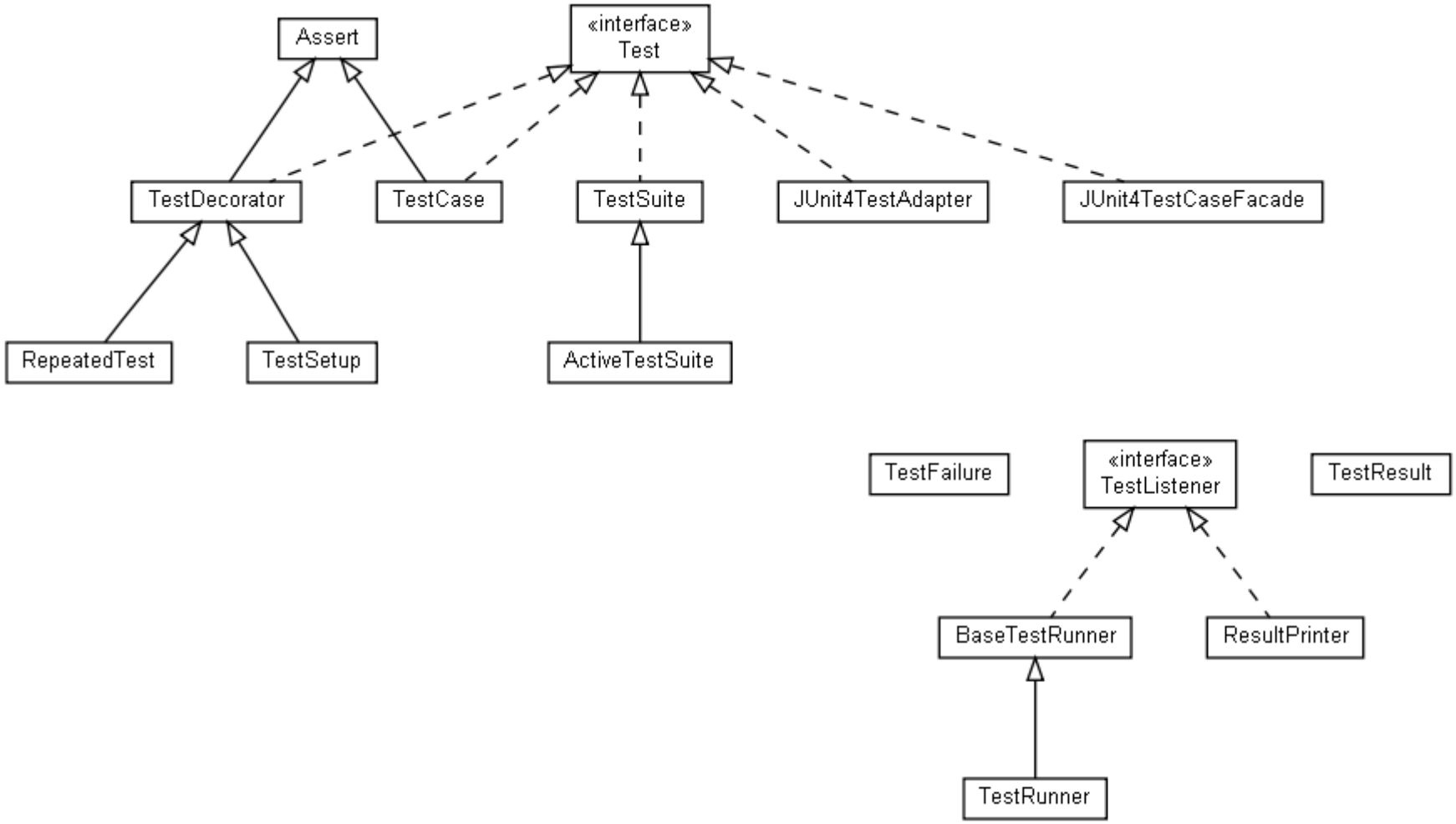
Outline

- Overview
- Template Method Pattern
- Composite Pattern
- Command Pattern
- Collecting Pattern
- Interesting Finds

The Project - JUnit

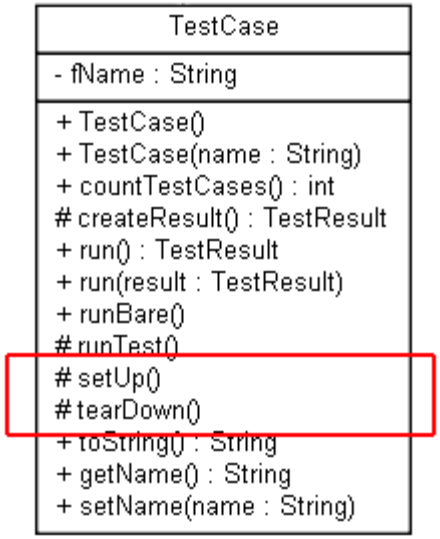
- Industry standard for Java testing
- Open source – IBM's CPL 0.5
- Over 10 years old
  - Based on SUnit, Smalltalk unit testing framework
- We worked with JUnit 4.5

Key Class Diagrams



Template Method Pattern

- Found in junit.framework
- Used for pre- and post-run commands
- Made explicitly to be overwritten



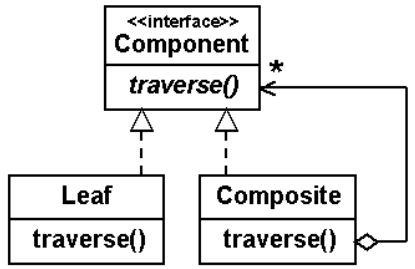

```
    * Sets up the fixture, for example, open a network connection.
protected void setUp() throws Exception {
}
    * Tears down the fixture, for example, close a network connection.
protected void tearDown() throws Exception {
}
```

Lines 179-190 http://junit.cvs.sourceforge.net/viewvc/junit/junit/src/main/java/junit/framework/TestCase.java?view=markup

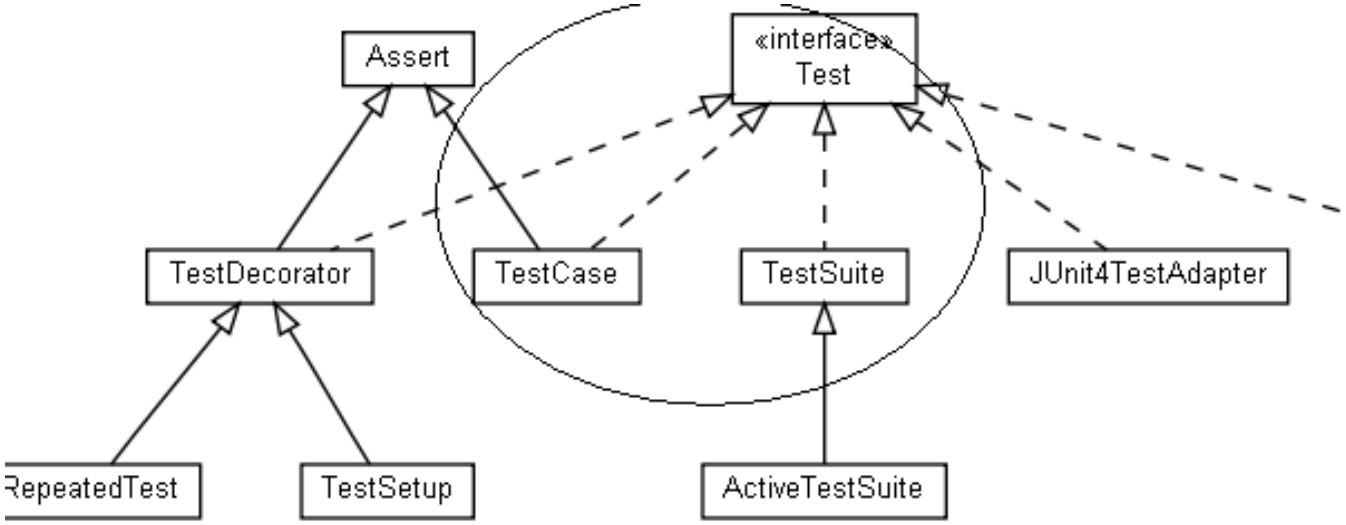
Template Method Pattern - Analysis

- Needed a way for programmers to add code before and after tests without having to rewrite entire run method
- Upside - Very simple to implement: overwrite 2 methods
- Downside - Requires knowledge of API, doesn't "announce" itself by causing errors if ignored
- Alternatives - Pluggable Selector
  - Doesn't allow tight integration with TestCase methods/variables
  - More overhead and complexity

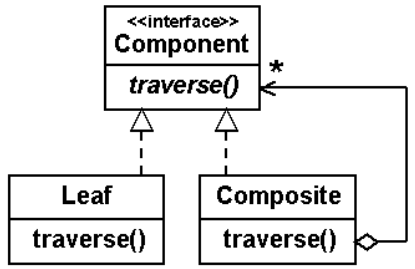
Composite Pattern

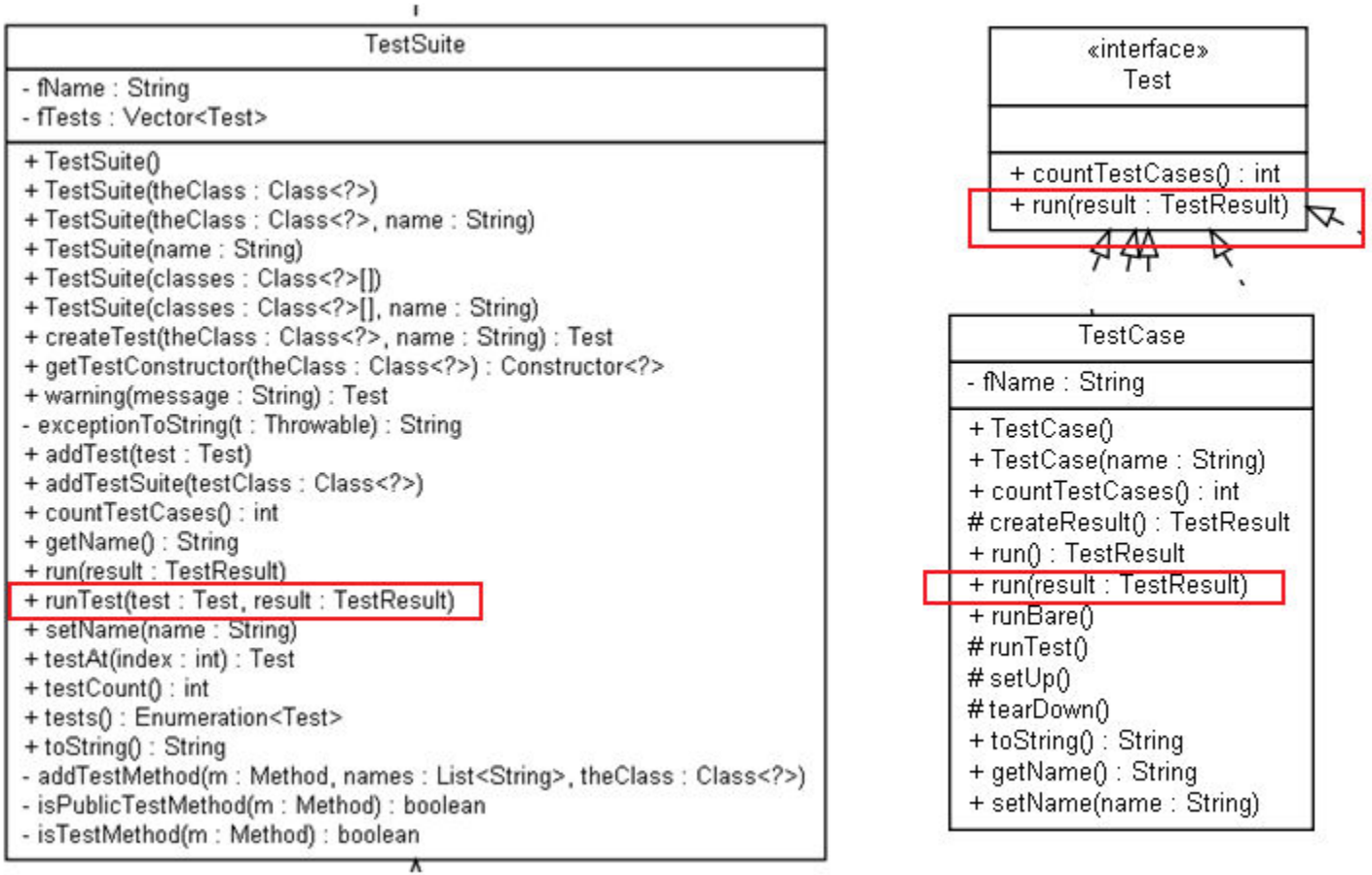


TestSuite defines the composite pattern as it combines many ***TestCase*** instances.



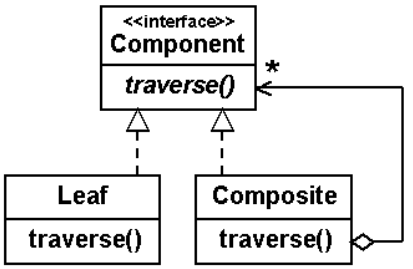
Composite Pattern - Classes





http://supportweb.cs.bham.ac.uk/docs/java/packages/junit/api/junit/framework/package-summary.html

Composite Pattern - Analysis



- The composite pattern is used here for
  - Composability
  - Extensibility
- A **TestSuite** could be a collection of **TestSuite** which is a collection of **TestCase**s where individual tests are defined by a **TestCase**

Pros:

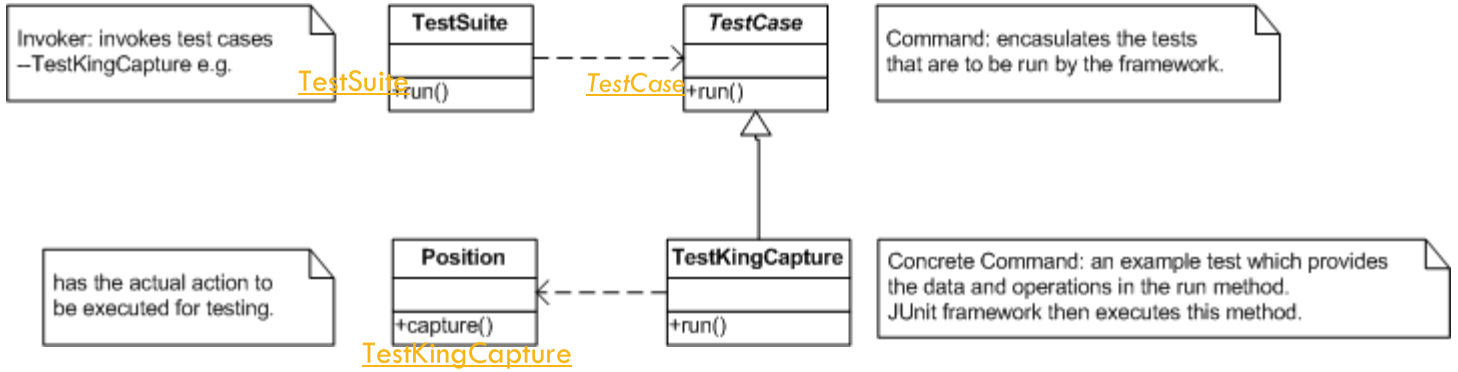
- Ability to represent part-whole hierarchies
- A **TestCase** can be treated like a **TestSuite** because of the implicit recursive hierarchy

Cons:

- Tradeoff between **transparency** (if child methods are on leaf node) and **safety** (if child methods are on the composite node)

Command Pattern





- JUnit framework provides an abstract class TestCase which provides a hock for behavior to be provided later in any shape.

- At runtime, the "command" objects which extend the TestCase class are executed by the framework. Those objects contain the details of what to do.

- E.g statements to assert that the king is captured if 4 warriors surround him.

http://supportweb.cs.bham.ac.uk/docs/java/packages/junit/api/junit/framework/package-summary.html

Command Pattern



Invoker: invokes
–TestKingCaptu

has the actu
be executed

TestSuite

- fName : String
- fTests : Vector<Test>

+ TestSuite()
+ TestSuite(theClass : Class<?>)
+ TestSuite(theClass : Class<?>, name : String)
+ TestSuite(name : String)
+ TestSuite(classes : Class<?>[])
+ TestSuite(classes : Class<?>[], name : String)
+ createTest(theClass : Class<?>, name : String) : Test
+ getTestConstructor(theClass : Class<?>) : Constructor<?>
+ warning(message : String) : Test
- exceptionToString(t : Throwable) : String
+ addTest(test : Test)
+ addTestSuite(testClass : Class<?>)
+ countTestCases() : int
+ getName() : String
+ run(result : TestResult)
+ runTest(test : Test, result : TestResult)
+ setName(name : String)
+ 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

TestCase

- fName : String

+ TestCase()
+ TestCase(name : String)
+ countTestCases() : int
# createResult() : TestResult
+ run() : TestResult
+ run(result : TestResult)
+ runBare()
# runTest()
# setUp()
# tearDown()
+ toString() : String
+ getName() : String
+ setName(name : String)

•JUnit fr ock
for behc

•At runti
executec do.

•E.g stat im.

http://supportweb.cs.bham.ac.uk/docs/java/packages/junit/api/junit/framework/package-summary.html

Command Pattern - Why?

- It's required that the user be able to write the statements to be tested and the framework should be able to execute those statements. The user shouldn't worry about other related issues.

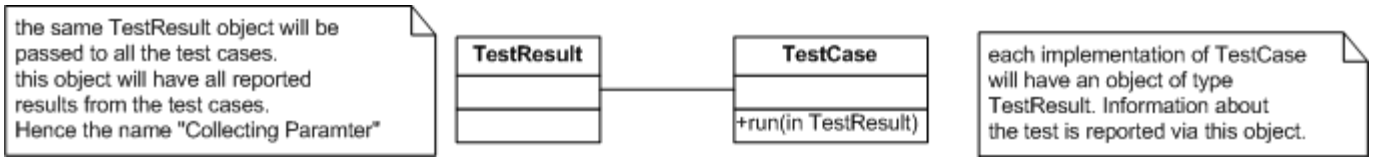
- The command pattern is the best choice for this behavior as it provides a way for testers to plugin their tests

Command Pattern - Analysis

- **Pros**:
  - Provides good isolation as each class has its own details of what to do and the data it operates on
  - Flexible as the design is for interface (TestCase) and not specific implementation class. Many classes can be created as needed.
  - Decouples invoker (TestSuite) from the receiver (Position)

- **Cons:**
  - TestCase has only one method – run(). No centralizations based on functions. For example one method for exceptions testing and one for object creation etc. this will help centralize common actions for easier maintenance.

Collecting Parameter Pattern



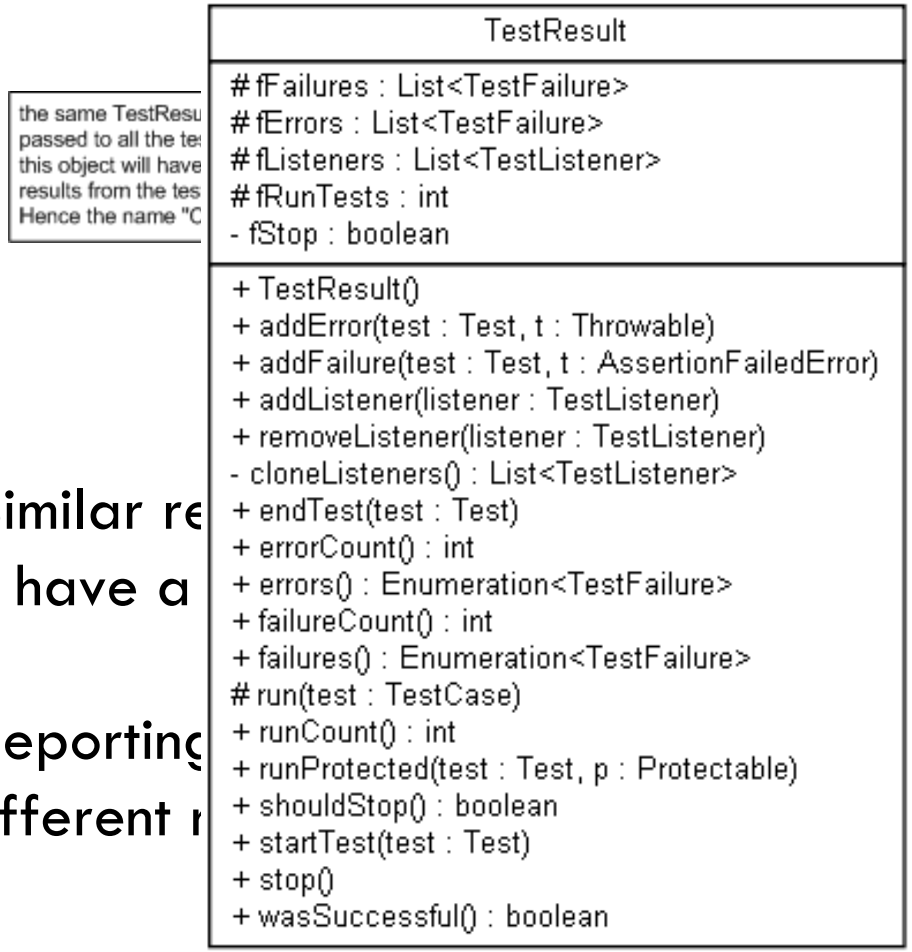
TestResult *TestCase*

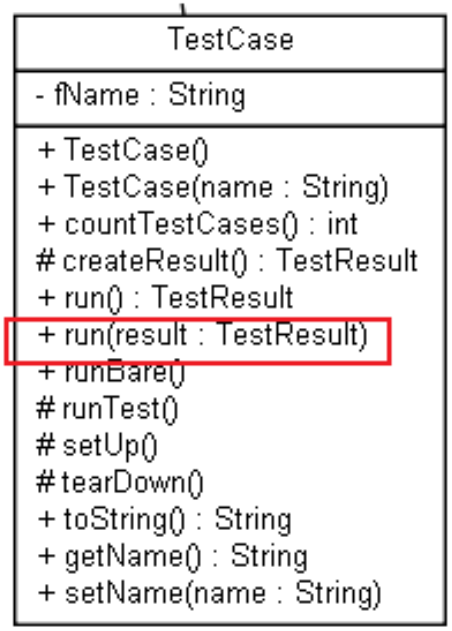
- Similar results need to be reported from different tests. It's nice to have a single way of collecting such results.

- Reporting of results should be easy to modify to accommodate different results

http://supportweb.cs.bham.ac.uk/docs/java/packages/junit/api/junit/framework/package-summary.html

Collecting Parameter Pattern





•Similar re nice to have a

•Reporting late different r

http://supportweb.cs.bham.ac.uk/docs/java/packages/junit/api/junit/framework/package-summary.html

Collecting Parameter Pattern - Overview

- A single object of type *TestResult* is passed to the *run*() method in TestCase whatever the number of implantations there exist.
- This means data reported from results will accumulate in the *TestResult* which is then reported to the user by the JUnit framework.
- JUnit has different implementations of TestResult each providing different kind of results.

Collecting Parameter Pattern - Analysis

- **Pros**:
  - Unifies error reporting from all objects under test.
  - Extensible: a different implementation of TestResult can be used to report different data about the tests
- **Cons:**
  - We have to check for synchronizations issues to make sure no data is lost as TestResult may be accessed at the same time by different objects.
  - Results in the signature of run() to have an additional parameter for all the tests. This is currently solved in JUnit by having a helper method that is aware of the TestResult but this complicated the design.

Interesting Findings

- Multiple level of abstractions through composite classes
- Direct invocation of JUnit TestCase constructor is not possible, and a setup() method is used for every TestCase

  This is designed to provide safety; Although it really adds to the complexity for JUnit usage.