

SilverMark's Enhanced JUnit



Introduction and tour



Topics

- ✍ What is JUnit and why enhance it?
- ✍ Product overview
 - User interface changes
 - JUnit/*profile*
 - JUnit/*load*
 - JUnit/*generate*
 - JUnit/*util*
- ✍ Integration with SilverMark's Test Mentor

What is JUnit?

- ✍ Simple, open-source test execution framework
 - Very light-weight
 - Little automation in test creation, metrics, profiling, load testing
- ✍ Written by Kent Beck and Erich Gamma
- ✍ Commonly used to illustrate XP test-first practices
- ✍ Open-source additions:
 - Cactus
 - J2MEUnit, etc.

Why enhance JUnit?

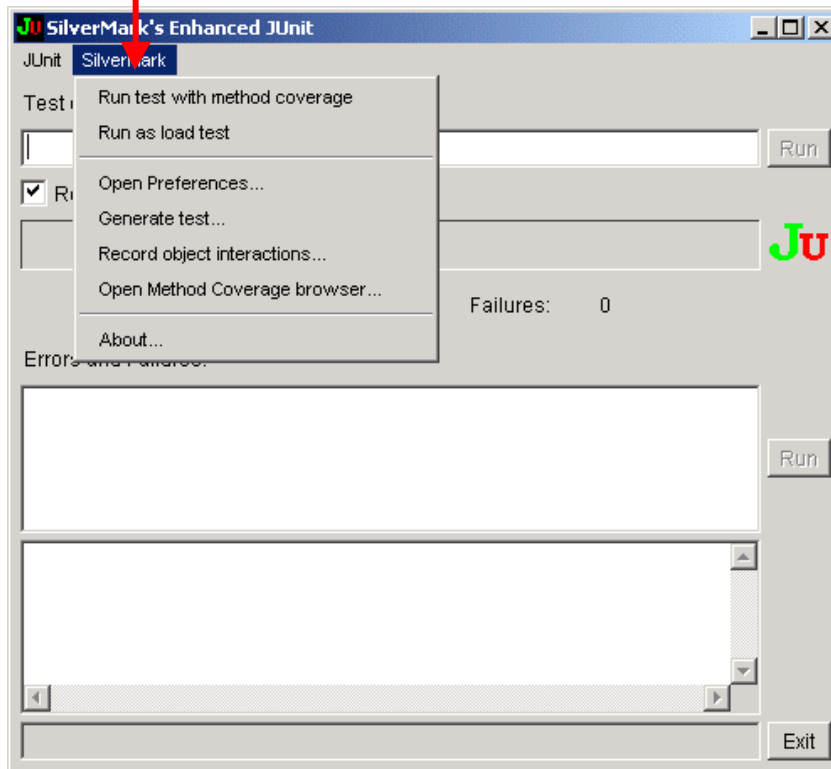
- ✍ Make test creation easier through automation
 - Test generation based on class structure
 - Test generation based on recorded object interactions
- ✍ Promote consistent test architecture based on common test design patterns
- ✍ Make it easy to perform load/stress testing
 - Distribute JUnit tests on multiple threads, VMs and CPUs
 - Show throughput response graphs
- ✍ Promote better test coverage
 - Report methods covered and not covered by test
- ✍ Add convenience APIs for
 - Test data-driven testing
 - Fine grained timings

What is SilverMark's Enhanced JUnit?

- ✍ A suite of tools that enhance JUnit
 - **JUnit/profile** – method coverage and object interaction profiling
 - **JUnit/load** – load testing across any number of threads, JVMs and CPUs
 - **JUnit/generate** – automatic test generation
 - **JUnit/util** – handy utilities and APIs
- ✍ Does not change JUnit. Not one line of code in JUnit was changed to accommodate Enhanced JUnit. All Enhanced JUnit view classes are subclasses of corresponding JUnit classes

User interface additions

SilverMark menu added to JUnit **awt** and **swing** UIs



Access all enhanced JUnit functionality through this menu

Menus

- ✎ Run test with method coverage...
 - Run specified test and display method coverage profile results view
- ✎ Run as load test...
 - Distribute specified test to load test agents on same or remote machines. Display real-time execution time metrics over incremental load
- ✎ Open preferences...
 - Open preferences view to set global settings

Menus

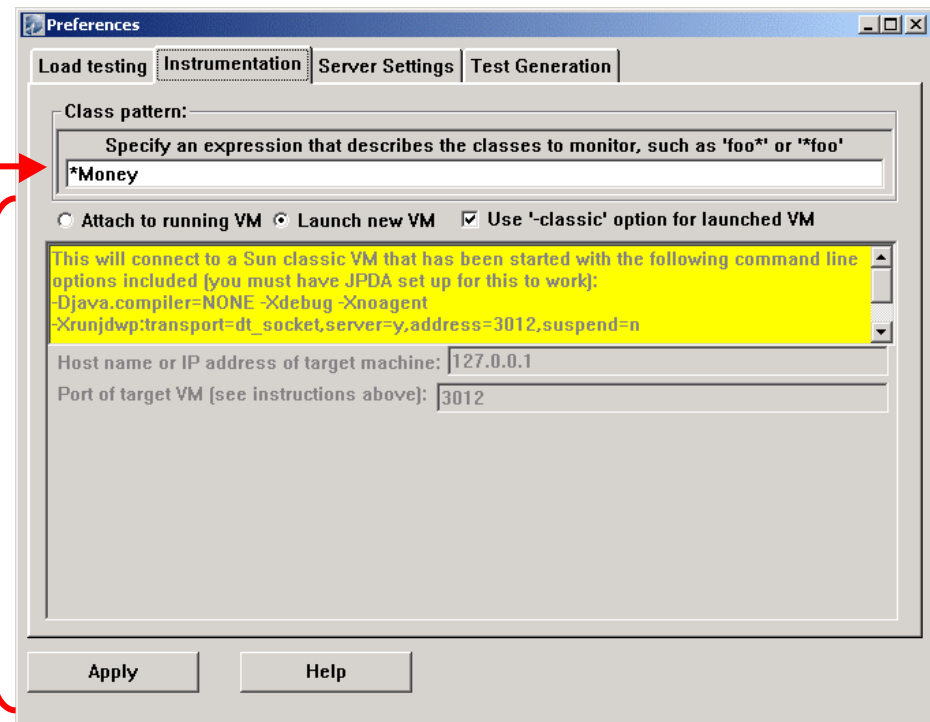
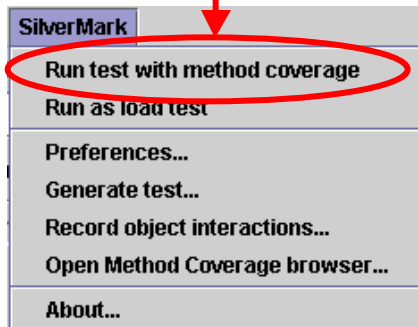
- ✍ **Generate test**
 - Generate tests according to global generation preferences
- ✍ **Record object interactions...**
 - Record interactions between objects in order to debug, or generate tests according to those interactions
- ✍ **Open method coverage browser...**
 - Open an empty browser. Can then load saved coverage data
- ✍ **About...**
 - Shows version, support info

JUnit/*profile*

- ✍ Monitor method coverage for designated classes or packages
 - determine which areas of your code are being tested and more importantly, which are not.
- ✍ Monitor interactions between designated objects.
 - Interactions are shown as interaction diagrams
 - ✍ Help you understand what's happening inside your system
 - ✍ Generate tests from recorded interactions

Setting up method coverage profiling

- Set preferences:
 - Specify pattern for classes to monitor
 - Specify VM options
- Run with coverage



Results of method coverage profiling

View of methods entered and not entered, broken down by test, package and class:

Item	Name	Methods	% Covered	Methods covered	Methods not covered
- Ju	All packages and	13	100.00	13	0
-	junit.samples.money	13	100.00	13	0
-	Money	13	100.00	13	0

Entries	Methods entered	Methods not entered
1 406	junit.samples.money.Money.currency()	
2 208	junit.samples.money.Money.amount()	
3 153	junit.samples.money.Money(int,java.lang.S	
4 65	junit.samples.money.Money.isZero()	
5 27	junit.samples.money.Money.equals(java.la	
6 19	junit.samples.money.Money.add(junit.samp	
7 17	junit.samples.money.Money.addMoney(juni	
8 13	junit.samples.money.Money.negate()	

Object interaction profiling

Use wizard (not shown) to specify classes to observe

- Diagram shows time-ordered interactions
- Interaction source and target method signatures
- Passed arguments

Interacting objects

Object Interaction Recording Wizard

Pause Resume Update End Clear Create test

Show diagram Show table

add_ActionPerformed(java.awt.event.ActionEvent)	addItem(java.lang.String)
getItemCountString()	getItemCount()
add_ActionPerformed(java.awt.event.ActionEvent)	addItem(java.lang.String)
getItemCountString()	getItemCount()
add_ActionPerformed(java.awt.event.ActionEvent)	addItem(java.lang.String)
getItemCountString()	getItemCount()
remove_ActionPerformed(java.awt.event.ActionEvent)	removeItemAtIndex(int)
getItemCountString()	getItemCount()
update_ActionPerformed(java.awt.event.ActionEvent)	replaceItemAtIndex(int, java.lang.String)

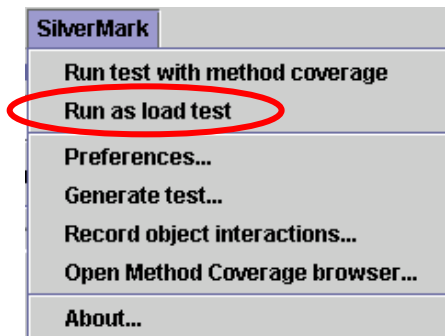
Argument values: <none>

<< Back Next >> Finish Cancel Help

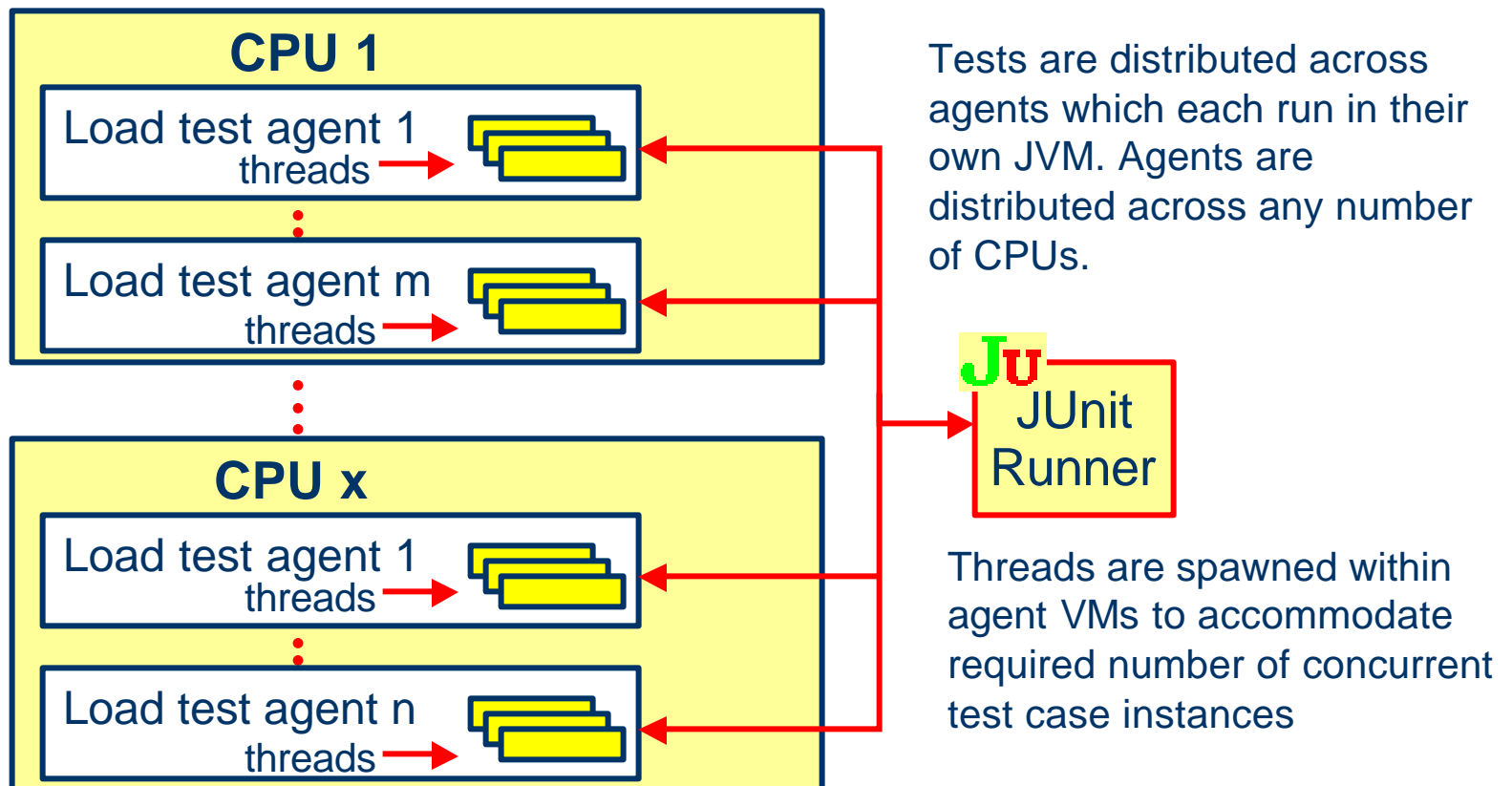
time

JUnit/load

- ✍ JUnit/load distributes your JUnit tests across multiple threads, JVMs, and CPUs to run multiple instances of your tests in parallel and report report execution times over a range of loads
- ✍ Run your JUnit tests across hundreds of computers to monitor throughput for shared resources, such as EJBs under load, or use it to unit test thread safety issues
- ✍ Use Run as load test to run any JUnit test as a load test:



JUnit/load topology



Tests are distributed across agents which each run in their own JVM. Agents are distributed across any number of CPUs.

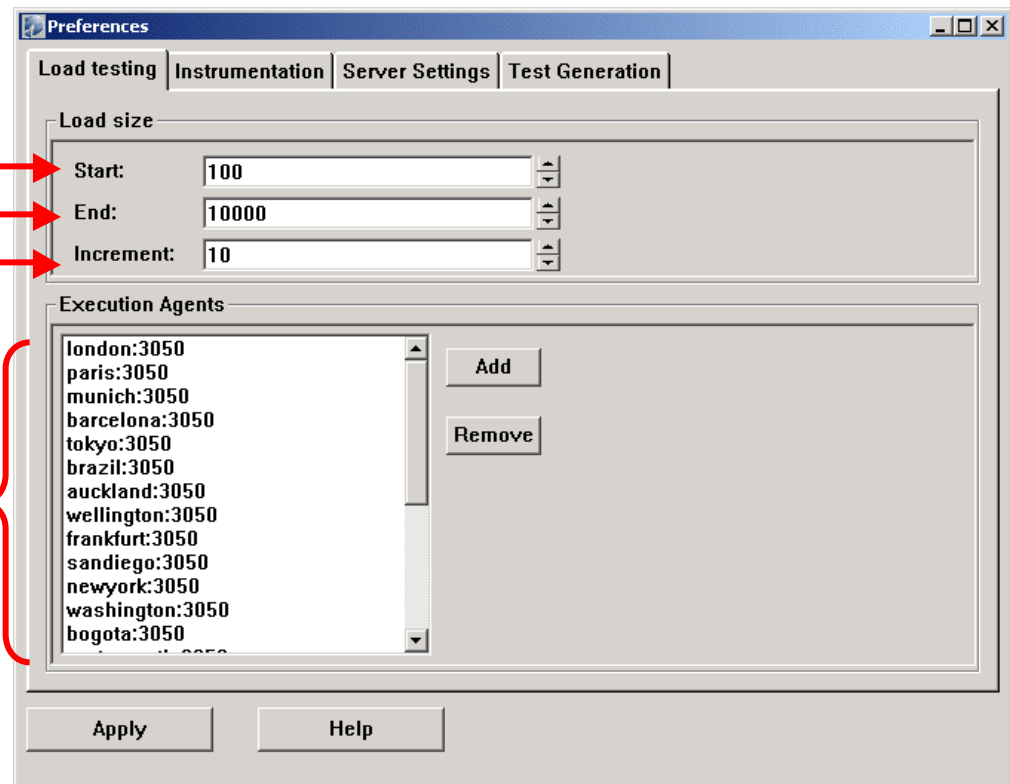
Ju
JUnit
Runner

Threads are spawned within agent VMs to accommodate required number of concurrent test case instances

You get better concurrency with more CPUs

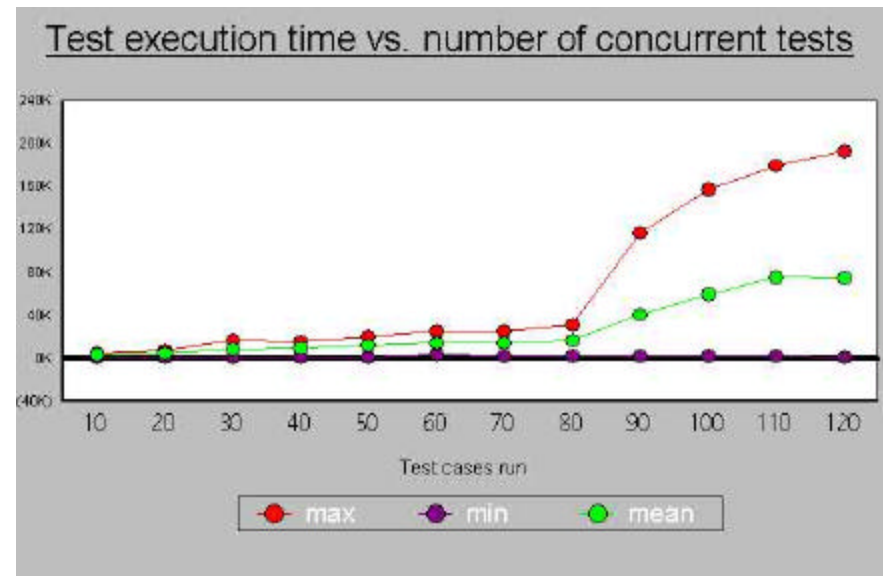
Running JUnit/load

- ✎ Specify number of concurrent tests to run in preferences
 - Starting load
 - Ending load
 - Number of tests to add for each run
- ✎ Specify list of machines to distribute tests on
- ✎ Select “Run as load test” menu



JUnit/load output

- ✍ Shows individual test case execution time (max, min, mean) vs. number of concurrent instances
- ✍ Typically used to measure shared resource reaction to multiple concurrent requests

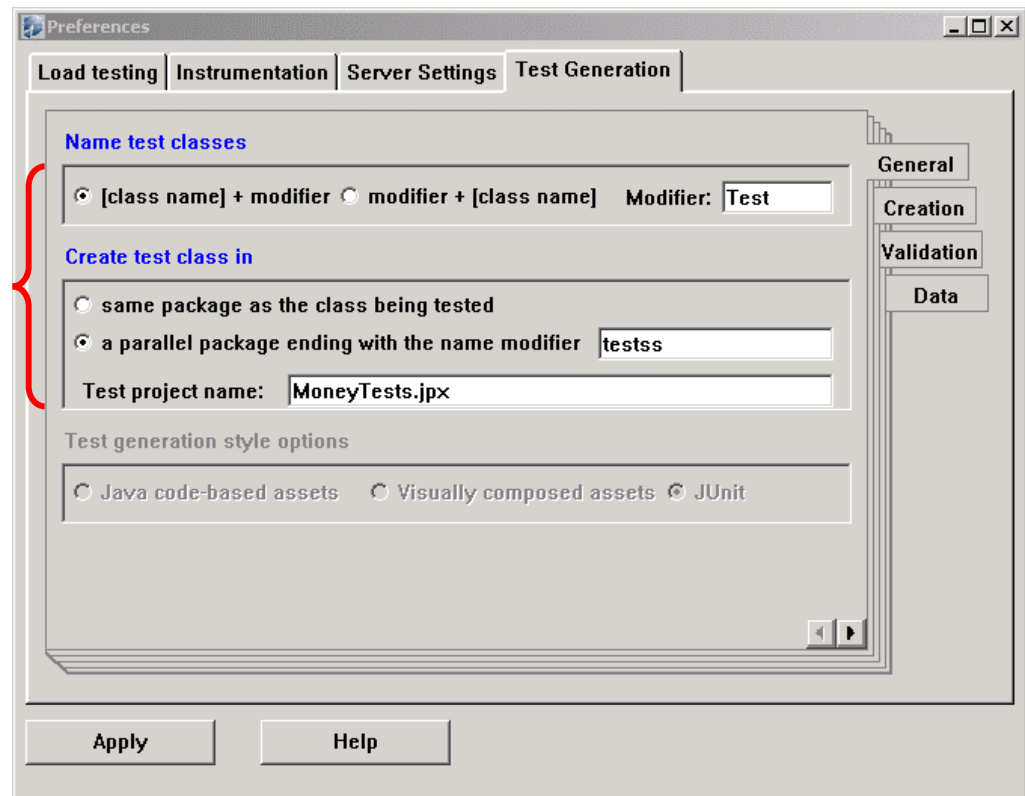


JUnit/generate

- ✍ JUnit/generate generates JUnit tests according to common design patterns.
 - Use these tests to start you out with a basic test organization and coverage, rather than always starting from scratch
 - Select “Generate test...” menu. Will prompt for class or package
- ✍ Also generates tests directly from interaction diagrams created by JUnit/profile.
 - Enables you to create unit tests for classes simply by running your system, which is useful when you would like to add unit tests to legacy code

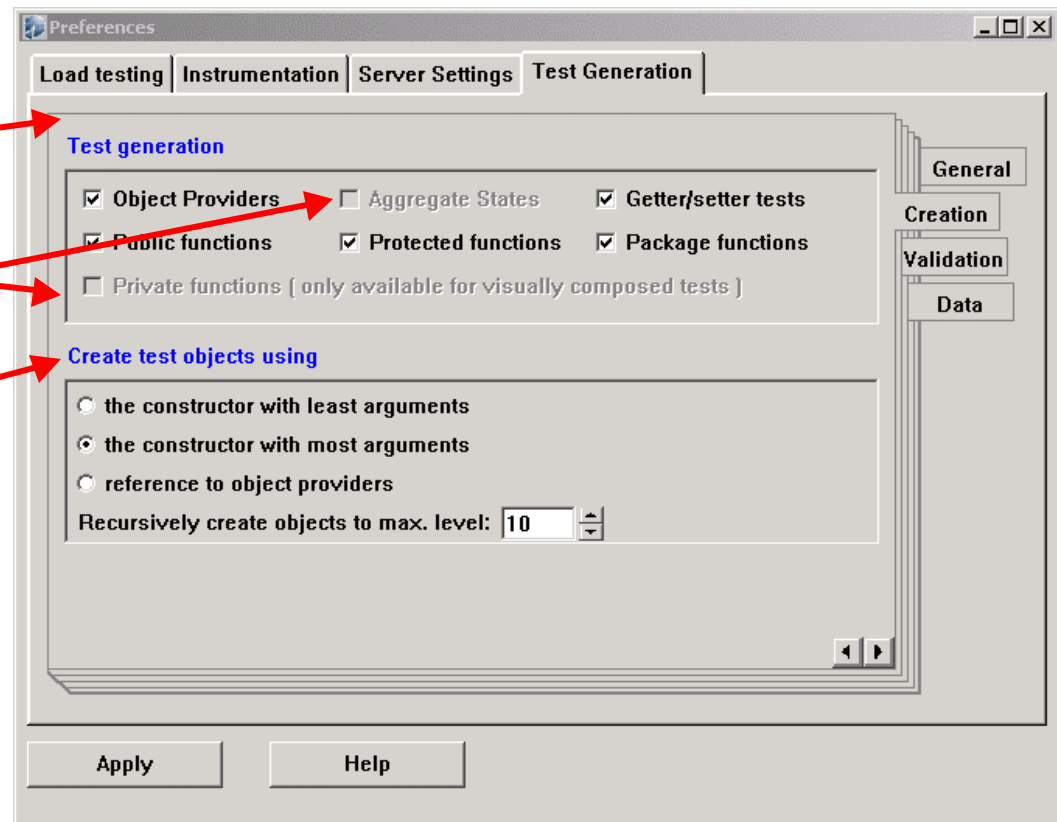
JUnit/generate preferences

 Test case naming and placement options



JUnit/generate preferences

- ✍ Specify which members to generate tests for
- ✍ Disabled options are only available in Test Mentor
- ✍ Object under test, and method and constructor parameter generation options



JUnit/generate generation from recorded object interactions

- ✍ Press Create test in object interaction view to launch wizard
- ✍ Specify name and description of generated test
- ✍ Specify perspective via actor
 - Test takes place of actor
- ✍ Object under test
 - Target of method calls
- ✍ Add assertions afterward to validate expected state

Sequence Test Wizard

Test

Name: Add and update

Description: Add/remove/update

Actor

Name: TodoList

Class: com.silvermark.examples.views.TODOList

Object under test

Name: TodoListModel

Class: com.silvermark.examples.views.TODOListModel

<< Back Next >> Finish Cancel Help

Generated test (part 1)

```
/**
 * Test the Add and update
 * Add/remove/update
 *
 *
 * This asset has (9) step(s):
 * <ul>
 * <li>addItem(String)</li>
 * <li>itemCount()</li>
 * <li>addItem(String)</li>
 * <li>itemCount()</li>
 * <li>addItem(String)</li>
 * <li>itemCount()</li>
 * <li>removeItemAtIndex(int)</li>
 * <li>itemCount()</li>
 * <li>replaceItemAtIndex(int,String)</li>
 * </ul>
 *
 * Creation date: (1/21/2002 9:58:44 AM)
 * @author Generated by SilverMark's Enhanced JUnit
 */
```

Generated test (part 2)

```
public void addAndUpdate(){
    com.silvermark.examples.views.TODOListModel subject = null;

    // Add and update:
    // Test the Add and update operations
    // Add/remove/update

    // Create test subject:
    // Create a test instance of TODOListModel as the receiver for this test
    subject = new com.silvermark.examples.views.TODOListModel();

    // addItem(String):
    // Apply the method addItem as a stimulus to the
    com.silvermark.examples.views.TODOListModel test instance
    subject.addItem("Wake up");
```

Generated test (part 3)

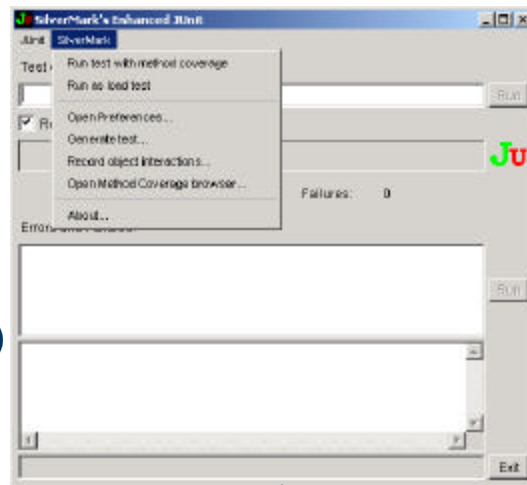
```
// itemCount():  
// Apply the method itemCount as a stimulus to the  
// com.silvermark.examples.views.TODOListModel test instance  
subject.itemCount();  
  
// addItem(String):  
// Apply the method addItem as a stimulus to the  
// com.silvermark.examples.views.TODOListModel test instance  
subject.addItem("Eat breakfast");  
  
// itemCount():  
// Apply the method itemCount as a stimulus to the  
// com.silvermark.examples.views.TODOListModel test instance  
subject.itemCount();  
  
// addItem(String):  
// Apply the method addItem as a stimulus to the  
// com.silvermark.examples.views.TODOListModel test instance  
subject.addItem("go to work");
```

JUnit/*util*

- ✍ A growing set of handy utilities and APIs that simplify common tasks, such as iterating over test data files or adding fine-grained performance assertions.

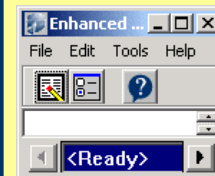
Architecture

Runner user interface
(AWT and Swing UI subclasses of JUnit AWT and Swing UI)

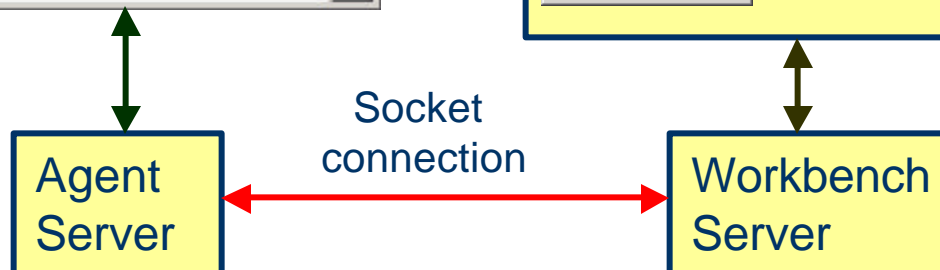


Enhanced JUnit Workbench

- Test generation
- Load test results
- Profiling results



Workbench Console UI



Integration with SilverMark's Test Mentor – *Java Edition*

- ✍ SilverMark's Test Mentor is a commercial enterprise Java component testing tool
 - Enables developers and testers to collaborate on component testing
 - Provides a visual test composition environment to enable testers to create component tests without knowing how to write code
- ✍ Test Mentor runs JUnit tests so testers can run the same tests developers create

Conclusion

- ✍ SilverMark's Enhanced JUnit brings features of high-priced tools directly to JUnit, at a low cost
 - Profiling
 - Load testing
 - Test generation
- ✍ Enhanced JUnit does not change the way you normally work with JUnit – it enhances it