Detailed Step-by-Step Replace Inheritance with Composition on Fitnesse ResultResponder.

1a)	Replace constructor with factory method	Replace constructor with factory method in each of the **ResultResponder** subclasses: - **WhereUsedResponder** - **SearchReplaceResponder** - **SearchReplaceResponder** - **ExecuteSearchPropertiesResponder** Run the tests & check-in if green.
1b)	Change the return type returned by the factory method to be the super type & fix the compile breaks	3 out of 4 of the tests compile ok with the simple change of changing the return type. **ExecuteSearchPropertiesResponderTest** has tests that access the **getPageTypesFromInput()** method and **getAttributesFromInput()** method. These methods (and methods they call) do not access any fields from their declaring class, and so can be made static. By making them static we can change the test to reference the methods statically to get the code compiling again. Run the tests & check-in if green.
2)	Create a new concrete subclass which implements the public interface by delegating to an delegator	Create **DelegatingResultResponder**. The constructor takes a single argument, a **ResultResponder**, which is assigned to a field. The **DelegatingResultResponder** implements **getTitle**() and **traverse(observer)** by delegating to the field. Run the tests & check-in if green.
3)	Extract an interface from the superclass and use that type in the delegating implementation	Interfaces need to have public methods, so before extracting the interface, change the signature on **getTitle()** so that its visibility is public. Run the tests & check-in if green.
		Extract an interface (**ResultResponderStrategy**?) from **ResultResponder** which has 2 methods: the **getTitle**() and the **traverse(observer).** You should be able to replace the type of **ResultResponder** with the new interface **ResultResponderStrategy** in the delegating subclass **DelegatingResultResponder**. Run the tests & check-in if green.
4a)	Now for each subclass, change it from extending the superclass to implementing the new strategy interface	Looking at the 4 subclasses, it looks like **WhereUsedResponder** is the simplest, so we start with that one. Change **WhereUsedResponder** to implement the **ResultResponderStrategy** interface instead of extending **ResultResponder**. Fix the factory method by returning a new **DelegatingResultRespoder()** wrapping a **WhereUsedResponder()**.

		We note that the implementation of **traverse** does not compile as it references fields that were on the **ResultResponder** class – the **page** and **root** fields. We fix this by passing **page** & **root** as method arguments to **traverse** from the **DelegatingResultResponder**. N.B. to change the signature of **traverse** without affecting the existing **ResultResponder** class we will need to first stop **ResultResponder** implementing the strategy interface. Once the code compiles, we discover that the unit test **ResponderFactoryTest.testCreateWhereUsedResponder()** is broken. The test is asserting about the created type of the **WhereUsedResponder**, which is now a **DelegatingResultResponder** wrapping a **WhereUsedResponder.** One way to fix the test without too much thought about it being a good test or not would be to change its assertion to cast to the **DelegatingResultResponder** and the assert on the type of the delegate. It's not pretty – should make a note to fix this in a better way later. Run the tests & check-in if green.
4b)	Next subclass	Next easiest is **ExecuteSearchPropertiesResponder.** It needs 2 extra parameters to traverse – **request** and **response.** Once these are added, the **ResultResponderFactoryTest** needs to change to reflect the new delegating structure. Run the tests & check-in if green.
4c)	Next subclass	Next we tackle **SearchReplaceResponder**. Some of the methods on this class – **getSearchString()**, **getReplacementString()** refer to the **request** object so we need to change them to take the **request** object as a parameter. Doing that makes **traverse()** compile ok, but then **getTitle()** also calls **getSearchString()** and **getReplacementString()** and it doesn't have a **request** object. So we change that – in **DelegatingResultResponder** we pass the **request** object into the **getTitle()** method. Once again, we need to change **ResultResponderFactoryTest** to reflect the new delegating structure. Run the tests & check-in if green.
4d)	Final subclass	The last remaining subclass of **ResultResponder** is **SearchResponder**. Following the pattern of the previous subclasses, we can make this implement the **ResultResponderStrategy.** The main difference with **SearchResponder** compared to the other classes we have refactored is that it has an overriding protected method **shouldRespondWith404()**. As soon as **SearchResponder** is changed from extending **ResultResponder** and implementing **ResultResponderStrategy** the **shouldRespondWith404()** behaviour is lost. [actually, the tests still pass without the shouldRespondWith404() override so maybe it's not needed, but let's assume it is…] There are at least a couple of ways to fix the 404() behaviour. One way would be to add a constructor parameter to the **DelegatingResultResponder** class which indicates whether it should respondWith404() or not. For the factory method building the **SearchResponder** we will set this flag to false. For the

		other 3 factory methods we will set the flag to true. The **DelegatingResultResponder** should override the **shouldRespondWith404()** method and return the flag that was passed in at construction time. Run the tests & check-in if green.
5	Merge the delegating subclass and the superclass	Inline **ResultResponder** into **DelegatingResultResponder** since it is the only subclass Run the tests & check-in if green.
6	Fix up any remaining issues – e.g. naming and/or tests	Rename the strategy implementations to be called **...Strategy**. Look at the tests and see if they can be split up – maybe the **Strategy** can be tested without having to create a **ResultResponder**. Run the tests & check-in if green.