1. Testing

The tests from C6 were kept and reused with some small changes. The test suit was extended with new tests for view functionality inspired by C5’s unit test. This chapter reflects on the unit testing in C6 Continued and discusses the challenges.

// Something about Test driven development

// The principles I followed and why? Mikkel’s approach, but it is not enough

“Though the list is not exhaustive, these are some of the tests that reappear  
independently of the method’s specific behavior. Method specific tests are still required and do not appear on the list.”

* 1. Changes made in C6

// something about the existing tests. Did I change anything?

C6 creates unit tests class hierarchy in form of abstract classes that matches the interface hierarchy [C6, 5.3] Each abstract class tests the public methods of one interface. This approach eliminates the duplication of test TODO. I continued with the same approach and the data structures, implemented within this project, used the existing unit tests without the need of write completely new tests for each of the data structures. It was enough to extend the abstract test class and to implement its abstract members. The newly added test for views TODO

Some properties return a fixed value for a given data structure, demanded by its indented behavior. For example, hashed version of the array list doesn’t allow duplicates. This requires the test methods for the IList interface to be configured so that they test properly the methods of any data structures implementing that interface. The tests in C6 generally take care of that fact, but some methods missed to consider it.

[Test]

public void ContainsRange\_SubsetWithDuplicates\_False()

{

Run.If(AllowsDuplicates); - was missing

// Arrange

var count = GetCount(Random) / 2;

var items = GetStrings(Random);

var newItems = items.Take(count).Append(items.First()).ShuffledCopy(Random);

var collection = GetCollection(items, ReferenceEqualityComparer);

// Act

var containsRange = collection.ContainsRange(newItems);

// Assert

Assert.That(containsRange, Is.False);

}

In listing TODO we can see that the method ContainsRange is called with newItems, which contains a duplicated item, items.First(). The result of the assertion should be False, but only if the data structure allows duplicates. This is not the case for HashedArrayList and HashedLinkedList, where they by design behave like a set. If a hashed version of list contains a number x and we call ContainsRange() with only xs, than the method will return true. That’s why the assertion in the listing fails, expecting True. For that reason, couples of test units were updated, putting the given restriction in the beginning of the test methods. Do I know the list TODO.

There are two ICollectionTest methods updated for another reason - *Update\_IntegerCollectionUpdateExistingItem\_RaisesExpectedEvents and* UpdateOrAdd\_IntegerCollectionUpdateExistingItem\_RaisesExpectedEvents

*[Test]*

*public void Update\_IntegerCollectionUpdateExistingItem\_RaisesExpectedEvents()*

*{*

*// Arrange*

*var items = new[] { 4, 54, 56, 8 };*

*var collection = GetCollection(items, TenEqualityComparer.Default);*

*Run.If(!(collection is IList<int>)); -- was missing*

*var count = DuplicatesByCounting ? 2 : 1;*

*var item = 53;*

*var expectedEvents = new[] {*

*Removed(54, count, collection),*

*Added(item, count, collection),*

*Changed(collection)*

*};*

*// Act & Assert*

*Assert.That(() => collection.Update(item), Raises(expectedEvents).For(collection));*

*}*

The test unit in the listing TODO uses TenEqualityComparer. This means that if the integer division by 10 of two items is equal than they are equal. The other important thing in the code is the property DuplicatesByCounting. It says whether the collection stores the duplicates. If it is true, then only one copy is stored and for the other copies an internal counter is incremented. This property is very important for the bags or treebags [C5, 1.4.14], but it is not on the same way important for IList’s classes. Because of their set semantic, HashedArrayList and HasedLinkedList don’t have duplicates, although the DuplicatesByCounting is True by design. However, C5 documentation doesn’t say explicitly that the true value doesn’t have real meaning and it is only to stick to the ??? semantic. The test will fail for this two data structures, because Update method will update only one item equal to the item parameter, therefore it will not raise Remove event for 2 deleted items, as it is forced by the true value of DuplicatesByCounting. The test is edited by putting a condition that this test is relevant only for non-IList data structures (i.e. bag, treebag)

* 1. View tests

A view refers to the items of the underlying list and any change in the items of the underlying list affects the view and vice versa. A modification on a view (remove, insert, add etc.) affects the underlying list [C5, 1.4.11] and as a result it affects all the related views (overlapping, situated in the right ???). This requires me to test the reflections of a view operation on the underlying list. I also need to test how an operation on an underlying list changes its views. This introduces complexity in the tests since I need to check not only the correctness of the object under the question, but correctness of the other objects related to it (the underlying list and the other views).

// Mikkel’s approach is also used for the view methods.

In C5 the tests use hard-coded test values. A disadvantage of hard-coded values is that the tests could pass for the given specific values, but can fail for some other. I avoided using collections with fixed item values, random collections are created instead, but I gave fixed parameters to some of the view methods. The reason for this is TODO. Intentionally avoid overlapping of views TODO

Scenarios – different scenarios must be considered. Firstly, there are 7 methods creating views …, 50 public methods. Look the picture. This goes far away.

I decided to implement the following approach: I considered 9 scenarios – getting zero-item view, one-item view and n-item view in the beginning of the list, in the middle of the list and in the end of the list. See the picture???.

// I could copy the tests, but I didn’t. Instead I updated with this and this.

// duplications

* 1. New test helpers introduced – TODO. Getview, Getcount.