Assignment 2 Test Book
For Reference from Scoring Results
(This Book Assumes the "interfaces" Package from GitHub Dated 3/9/19)
3/31/19

Test Book Usage Example

Given: test.ds.StackTests.test_pushPop(StackTests.java:80)
Do:

1. Find StackTests
2. Find "test_pushPop" and/or line 80
3. Reference test code and if desired, copy-paste it to your own "Tester" or "Lab" (change "assertEquals" to println or whatever you like, as desired

`StackTests`

```
67   public void test_pushPop(String label, Stack ds)
68   {
69        //println("Starting: " + label);
70        ds.push(10);
71        ds.push(20);
72        ds.push(30);
73        ds.push(40);
74        //test LIFO for pops
75        assertFalse(ds.isEmpty(), label + " -- isEmpty (full)");
76        assertEquals(4 , ds.size(), label + " -- size (full)");
77        assertEquals(40 , ds.pop(), label + " -- pop 1st");
78        assertEquals(30 , ds.pop(), label + " -- pop 2nd");
79        assertEquals(20 , ds.pop(), label + " -- pop 3rd");
80        assertEquals(10 , ds.pop(), label + " -- pop 4th");
81        assertTrue(ds.isEmpty(), label + " -- isEmpty (empty)");
82        assertEquals(0 , ds.size(), label + " -- size (empty)");
83   }
84
85   public void test_peek(String label, Stack ds)
86   {
87        //println("Starting: " + label);
88        initializeSample1(ds);
89        assertEquals(40, ds.peek(), label + " -- peek 1");
90        assertEquals(40, ds.peek(), label + " -- peek 2");
91        assertEquals(4, ds.size(), label + " -- size");
92        ds.pop();
93        assertEquals(30, ds.peek(), label + " -- peek (after pop)");
```

```
94        assertEquals(30, ds.peek(), label + " -- peek (after pop) 2");
95        assertEquals(3, ds.size(), label + " -- size");
96        ds.pop();
97        ds.pop();
98        assertEquals(10, ds.peek(), label + " -- peek (after pop, pop)");
99        assertEquals(10, ds.peek(), label + " -- peek (after pop, pop) 2");
100       assertEquals(1, ds.size(), label + " -- size");
101   }
102
103//     public void test_pushAll(String label, Stack ds)
104//     {
105//          //println("Starting: " + label);
106//          List list = newList('A', 'B', 'C');
107//          ds.pushAll(list);
108//          assertEquals(reverse(newList(list)), ds.asList());
109//     }
110
111   public void test_asList(String label, Stack ds)
112   {
113        //println("Starting: " + label);
114        initializeSample1(ds);
115        assertEquals(sample1ExpectedList(), ds.asList());
116   }
117
118   public void test_iterable(String label, Stack ds) {
119        //println("Starting: " + label);
120        initializeSample1(ds);
121        //Iterator iter = ((interfaces.Iterable)ds).asIterator();
122        Iterator iter = ((interfaces.UniIterable) ds).toIterator();
123        List actual = TestTools.iteratorToList(iter);
124        assertEquals(sample1ExpectedList(), actual);
125   }
```

QueueTests

```
59    public void test_enqueueDequeue(String label, Queue ds) {
60         //println("Starting: " + label);
61         ds.enqueue(10);
62         ds.enqueue(20);
63         ds.enqueue(30);
64         ds.enqueue(40);
65         // test LIFO for pops
66         assertFalse(ds.isEmpty(), label + " -- isEmpty (full)");
67         assertEquals(4, ds.size(), label + " -- size (full)");
68         assertEquals(10, ds.dequeue(), label + " -- dequeue 1st");
69         assertEquals(20, ds.dequeue(), label + " -- dequeue 2nd");
70         assertEquals(30, ds.dequeue(), label + " -- dequeue 3rd");
71         assertEquals(40, ds.dequeue(), label + " -- dequeue 4th");
72         assertTrue(ds.isEmpty(), label + " -- isEmpty (empty)");
73         assertEquals(0, ds.size(), label + " -- size (empty)");
74    }
75
76    public void test_peek(String label, Queue ds) {
```

```
77        //println("Starting: " + label);
78        initializeSample1(ds);
79        assertEquals(10, ds.peek(), label + " -- peek 1");
80        assertEquals(10, ds.peek(), label + " -- peek 2");
81        assertEquals(4, ds.size(), label + " -- size");
82        ds.dequeue();
83        assertEquals(20, ds.peek(), label + " -- peek (after dequeue)");
84        assertEquals(20, ds.peek(), label + " -- peek (after dequeue) 2");
85        assertEquals(3, ds.size(), label + " -- size");
86        ds.dequeue();
87        ds.dequeue();
88        assertEquals(40, ds.peek(), label + " -- peek (after dequeue, dequeue)");
89        assertEquals(40, ds.peek(), label + " -- peek (after dequeue, dequeue) 2");
90        assertEquals(1, ds.size(), label + " -- size");
91    }
92
93  //    public void test_enqueueAll(String label, Queue ds) {
94  //         //println("Starting: " + label);
95  //         List list = newList('A', 'B', 'C');
96  //         ds.enqueueAll(list);
97  //         assertEquals(list, ds.asList());
98  //    }
99
100   public void test_asList(String label, Queue ds) {
101       //println("Starting: " + label);
102       initializeSample1(ds);
103       assertEquals(sample1ExpectedList(), ds.asList());
104   }
105
106   public void test_iterable(String label, Queue ds) {
107       initializeSample1(ds);
108       //Iterator iter = ((interfaces.Iterable) ds).asIterator();
109       Iterator iter = ((interfaces.UniIterable) ds).toIterator();
110       List actual = TestTools.iteratorToList(iter);
111       assertEquals(sample1ExpectedList(), actual);
112   }
```

DequeTests

```
73    public void test_addRemoveFront(String label, Deque ds) {
74        //println("Starting: " + label);
75
76        ds.addToFront(10);
77        ds.addToFront(20);
78        ds.addToFront(30);
79        ds.addToFront(40);
80
81        assertFalse(ds.isEmpty(), label + " -- isEmpty (full)");
82        assertEquals(4, ds.size(), label + " -- size (full)");
83        assertEquals(40, ds.removeFront(), label + " -- removeFront 1st");
84        assertEquals(30, ds.removeFront(), label + " -- removeFront 2nd");
85        assertEquals(20, ds.removeFront(), label + " -- removeFront 3rd");
86        assertEquals(10, ds.removeFront(), label + " -- removeFront 4th");
```

```
87        assertTrue(ds.isEmpty(), label + " -- isEmpty (empty)");
88        assertEquals(0, ds.size(), label + " -- size (empty)");
89    }
90
91    public void test_addRemoveBack(String label, Deque ds) {
92        //println("Starting: " + label);
93
94        ds.addToBack(10);
95        ds.addToBack(20);
96        ds.addToBack(30);
97        ds.addToBack(40);
98
99        assertFalse(ds.isEmpty(), label + " -- isEmpty (full)");
100       assertEquals(4, ds.size(), label + " -- size (full)");
101       assertEquals(40, ds.removeBack(), label + " -- removeBack 1st");
102       assertEquals(30, ds.removeBack(), label + " -- removeBack 2nd");
103       assertEquals(20, ds.removeBack(), label + " -- removeBack 3rd");
104       assertEquals(10, ds.removeBack(), label + " -- removeBack 4th");
105       assertTrue(ds.isEmpty(), label + " -- isEmpty (empty)");
106       assertEquals(0, ds.size(), label + " -- size (empty)");
107   }
108
109   public void test_addRemoveMixed(String label, Deque ds) {
110       //println("Starting: " + label);
111
112       ds.addToFront(10);
113       ds.addToBack(20);
114       ds.addToFront(30);
115       ds.addToBack(40);
116       //order should be 30 10 20 40
117
118       assertFalse(ds.isEmpty(), label + " -- isEmpty (full)");
119       assertEquals(4, ds.size(), label + " -- size (full)");
120       assertEquals(30, ds.removeFront(), label + " -- removeFirst 1st remove");
121       assertEquals(40, ds.removeBack(), label + " -- removeBack 2nd remove");
122       assertEquals(20, ds.removeBack(), label + " -- removeBack 3rd remove");
123       assertEquals(10, ds.removeFront(), label + " -- removeBack 4th remove");
124       assertTrue(ds.isEmpty(), label + " -- isEmpty (empty)");
125       assertEquals(0, ds.size(), label + " -- size (empty)");
126   }
127
128   public void test_get(String label, Deque ds) {
129       //println("Starting: " + label);
130       initializeSample1(ds);
131       assertEquals(10, ds.getFront(), label + " -- getFront 1");
132       assertEquals(40, ds.getBack(), label + " -- getBack 1");
133       assertEquals(4, ds.size(), label + " -- size");
134       ds.removeFront();
135       assertEquals(20, ds.getFront(), label + " -- getFront (after removeFront)");
136       assertEquals(40, ds.getBack(), label + " -- getBack (after removeFront) 2");
137       assertEquals(3, ds.size(), label + " -- size");
138       ds.removeBack();
139       ds.removeBack();
```

```
140       assertEquals(20, ds.getFront(), label + " -- getFront (size=1)");
141       assertEquals(20, ds.getBack(), label + " -- getBack (size=1)");
142       assertEquals(1, ds.size(), label + " -- size");
143   }
144
145//     public void test_addAll(String label, Deque ds) {
146//          //println("Starting: " + label);
147//          //B A Y Z
148//          ds.addAllToBack(newList('Y', 'Z'));
149//          ds.addAllToFront(newList('A', 'B'));
150//          ds.addAllToBack(newList());
151//          ds.addAllToFront(newList());
152//          assertEquals(newList('B', 'A', 'Y', 'Z'), ds.asList());
153//     }
154
155   public void test_asList(String label, Deque ds) {
156       //println("Starting: " + label);
157       initializeSample1(ds);
158       assertEquals(sample1ExpectedList(), ds.asList());
159   }
160
161   public void test_iterable(String label, Deque ds) {
162       //println("Starting: " + label);
163       initializeSample1(ds);
164       //Iterator iter = ((interfaces.Iterable) ds).asIterator();
165       Iterator iter = ((interfaces.UniIterable) ds).toIterator();
166       List actual = TestTools.iteratorToList(iter);
167       assertEquals(sample1ExpectedList(), actual);
168   }
```

BagTests

```
73    public void test_addRemove(String label, Bag ds) {
74        //println("Starting: " + label);
75        ds.add(10);
76        ds.add(20);
77        ds.add(30);
78        ds.add(40);
79        // test LIFO for pops
80        assertFalse(ds.isEmpty(), label + " -- isEmpty (full)");
81        assertEquals(4, ds.size(), label + " -- size (full)");
82        ds.remove(40);
83        assertEquals(3, ds.size(), label + " -- size (1 removed)");
84        ds.remove(30);
85        assertEquals(2, ds.size(), label + " -- size (2 removed)");
86        ds.remove(10);
87        assertEquals(1, ds.size(), label + " -- size (3 removed)");
88        ds.remove(20);
89        assertTrue(ds.isEmpty(), label + " -- isEmpty (empty)");
90        assertEquals(0, ds.size(), label + " -- size (empty)");
91    }
92
93    public void test_any(String label, Bag ds) {
```

```
94        //println("Starting: " + label);
95        initializeSample1(ds);
96        assertTrue(ds.any() != null, label + " -- any 1");
97        assertTrue(ds.any() != null, label + " -- any 2");
98        assertEquals(4, ds.size(), label + " -- size");
99        ds.remove(20);
100       assertTrue(ds.any() != null, label + " -- any 3");
101       assertTrue(ds.any() != null, label + " -- any 4");
102       assertEquals(3, ds.size(), label + " -- size");
103   }
104
105   public void test_any_empty(String label, Bag ds) {
106       //println("Starting: " + label);
107       assertTrue(ds.any() == null, label + " -- any (empty)");
108   }
109
110   public void test_addAll(String label, Bag ds) {
111       //println("Starting: " + label);
112       List list = newList('A', 'B', 'C');
113       ds.addAll(list);
114       assertTrue(equalsIgnoreOrder(ds.asList(), list));
115   }
116
117   public void test_asList(String label, Bag ds) {
118       //println("Starting: " + label);
119       initializeSample1(ds);
120       assertTrue(equalsIgnoreOrder(ds.asList(), sample1ExpectedList()));
121   }
122
123   public void test_contains(String label, Bag ds) {
124       //println("Starting: " + label);
125       assertFalse(ds.contains(10), "contains on empty");
126       initializeSample1(ds);
127       assertTrue(ds.contains(30), "Should find 30");
128       assertFalse(ds.contains(-99), "Should not find -99");
129   }
130
131   public void test_iterable(String label, Bag ds) {
132       //println("Starting: " + label);
133       initializeSample1(ds);
134       Iterator iter = ((interfaces.UniIterable) ds).toIterator();
135       List actual = TestTools.iteratorToList(iter);
136       assertTrue(equalsIgnoreOrder(actual, sample1ExpectedList()));
137   }
```

IndexableListTests

```
83    public void test_addRemove(String label, IndexableList ds) {
84        //println("Starting: " + label);
85        ds.add(10);
86        ds.add(20);
87        ds.add(30);
```

```
88        ds.add(40);
89        assertFalse(ds.isEmpty(), label + " -- isEmpty (full)");
90        assertEquals(4, ds.size(), label + " -- size (full)");
91        ds.remove(0);
92        assertEquals(3, ds.size(), label + " -- size (1 removed)");
93        ds.remove(2);
94        assertEquals(2, ds.size(), label + " -- size (2 removed)");
95        ds.remove(1);
96        assertEquals(1, ds.size(), label + " -- size (3 removed)");
97        ds.remove(0);
98        assertTrue(ds.isEmpty(), label + " -- isEmpty (empty)");
99        assertEquals(0, ds.size(), label + " -- size (empty)");
100   }
101
102   public void test_get(String label, IndexableList ds) {
103       //println("Starting: " + label);
104       initializeSample1(ds);
105       assertEquals(10, ds.get(0), label + " -- get(0)");
106       assertEquals(20, ds.get(1), label + " -- get(1)");
107       assertEquals(30, ds.get(2), label + " -- get(2)");
108       assertEquals(40, ds.get(3), label + " -- get(3)");
109   }
110
111   public void test_set(String label, IndexableList ds) {
112       //println("Starting: " + label);
113       initializeSample1(ds);
114       ds.set(0,  1000);
115       ds.set(3,  1003);
116       ds.set(2,  1002);
117       ds.set(1,  1001);
118       assertEquals(newList(1000, 1001, 1002, 1003), ds.asList());
119   }
120
121   public void test_reverse(String label, IndexableList ds) {
122       //println("Starting: " + label);
123       ds.reverse();       //edge case first (empty ds) -- just ensure it does not below
124       initializeSample1(ds);
125       ds.reverse();
126       assertEquals(TestTools.reverse(sample1ExpectedList()), ds.asList());
127   }
128
129   public void test_indexOf(String label, IndexableList ds) {
130       //println("Starting: " + label);
131       initializeSample1(ds);
132       assertEquals(2, ds.indexOf(30), label + " -- indexOf(30)");
133       assertEquals(-1, ds.indexOf(99), label + " -- indexOf(99)");
134   }
135
136   public void test_contains(String label, IndexableList ds) {
137       //println("Starting: " + label);
138       initializeSample1(ds);
139       assertEquals(true, ds.contains(30), label + " -- contains(30)");
140       assertEquals(false, ds.contains(99), label + " -- contains(99)");
```

```
141  }
142
143  public void test_copyFromTo(String label, IndexableList ds) {
144       //println("Starting: " + label);
145       assertEquals(newList(), ds.copyFromTo(0, 0).asList(), label + " -- copyFromTo for empty");
146       initializeSample1(ds);
147       assertEquals(newList(20, 30), ds.copyFromTo(1, 2).asList(), label + " -- copyFromTo(1, 2)");
148       assertEquals(sample1ExpectedList(), ds.copyFromTo(0, 3).asList(), label + " -- copyFromTo(0, 3)");
149       assertEquals(newList(40), ds.copyFromTo(3, 3).asList(), label + " -- copyFromTo(3, 3)");
150  }
151
152  public void test_iterable(String label, IndexableList ds) {
153       //println("Starting: " + label);
154       initializeSample1(ds);
155       //Iterator iter = ((interfaces.Iterable) ds).asIterator();
156       Iterator iter = ((interfaces.UniIterable) ds).toIterator();
157       List actual = TestTools.iteratorToList(iter);
158       assertEquals(sample1ExpectedList(), actual);
159  }
```