```
1  /// Written by Camille Rasmussen
2  /// UID: u0717763
3  /// CS 3500 Fall 2014
4
5  using System;
6  using Microsoft.VisualStudio.TestTools.UnitTesting;
7  using SS;
8  using SpreadsheetUtilities;
9  using System.Linq;
10 using System.Collections.Generic;
11
12 namespace SpreadsheetTester
13 {
14     /// <summary>
15     /// This tester class holds all the tests for public/protected methods
16     /// of the Spreadsheet class. Extends Spreadsheet class specially for
17     /// test methods that test it's protected methods. This tester class
18     /// provides reasonably close to 100% code coverage.
19     /// </summary>
20     [TestClass]
21     public class SpreadsheetTester : Spreadsheet
22     {
23         /// <summary>
24         /// Tests actions of non-parameterized constructor
25         /// </summary>
26         [TestMethod]
27         public void Public_Test_NonParameterized_Constructor()
28         {
29             AbstractSpreadsheet sheet = new Spreadsheet();
30
31             // imposes no extra validity testing
32             Assert.IsTrue(sheet.IsValid("anything can go here"));
33             // normalizes every cell to itself
34             Assert.AreEqual("UofU is better than BYU", sheet.Normalize("UofU is better than BYU"));
35             // has the version 'default'
36             Assert.AreEqual("default", sheet.Version);
37             // changed is false
38             Assert.IsFalse(sheet.Changed);
39         }
40
41         /// <summary>
42         /// Tests actions of 3-parameterized constructor
43         /// </summary>
44         [TestMethod]
45         public void Public_Test_3Parameterized_Constructor()
46         {
47             AbstractSpreadsheet sheet = new Spreadsheet(x => false, x => x.ToUpper(), "info");
48
49             // test fails for all
50             Assert.IsFalse(sheet.IsValid("anything can go here"));
51             Assert.IsFalse(sheet.IsValid("A1"));
52             // normalizes every cell to upper
53             Assert.AreEqual("UOFU IS BETTER THAN BYU", sheet.Normalize("UofU is better than BYU"));
54             Assert.AreEqual("SOME OTHER STUFF TO CAPITALIZE", sheet.Normalize("Some other stuff to
    CaPiTaLiZe"));
55             // version contains 'info'
56             Assert.AreEqual("info", sheet.Version);
57             // changed is false
58             Assert.IsFalse(sheet.Changed);
59         }
60
61         /// <summary>
62         /// Tests a valid cell entry with text string
63         /// </summary>
64         [TestMethod]
65         public void Public_Test_CellContents_Valid_1()
```

```
66          {
67              AbstractSpreadsheet sheet = new Spreadsheet();
68              sheet.SetContentsOfCell("Y6", "hellow");
69              // check Changed
70              Assert.IsTrue(sheet.Changed);
71              Assert.AreEqual(sheet.GetCellContents("Y6"), "hellow");
72              Assert.AreEqual(sheet.GetCellValue("Y6"), "hellow");
73          }
74
75          /// <summary>
76          /// Tests a valid cell entry with text string
77          /// </summary>
78          [TestMethod]
79          public void Public_Test_CellContents_Valid_2()
80          {
81              AbstractSpreadsheet sheet = new Spreadsheet(x => true, x => x.ToUpper(), "version info");
82              sheet.SetContentsOfCell("abc7", "Spongebob");
83              // check Changed
84              Assert.IsTrue(sheet.Changed);
85              Assert.AreEqual(sheet.GetCellContents("abc7"), "Spongebob");
86              // make sure Normalize is being used appropriately
87              Assert.AreEqual(sheet.GetCellValue("ABC7"), "Spongebob");
88          }
89
90          /// <summary>
91          /// Tests a valid cell entry with Formula, no dependencies
92          /// </summary>
93          [TestMethod]
94          public void Public_Test_CellContents_Valid_3()
95          {
96              AbstractSpreadsheet sheet = new Spreadsheet();
97              sheet.SetContentsOfCell("I63", "=16 - 8.3");
98              // check Changed
99              Assert.IsTrue(sheet.Changed);
100             Formula testFormula = new Formula("16 - 8.3");
101             Assert.AreEqual(sheet.GetCellContents("I63"), testFormula);
102             Assert.AreEqual((double)sheet.GetCellValue("I63"), 7.7, 0.000000001);
103         }
104
105         /// <summary>
106         /// Tests a valid cell entry with Formula, with dependencies
107         /// </summary>
108         [TestMethod]
109         public void Public_Test_CellContents_Valid_4()
110         {
111             AbstractSpreadsheet sheet = new Spreadsheet();
112             sheet.SetContentsOfCell("i63", "32");
113             // check Changed
114             Assert.IsTrue(sheet.Changed);
115             sheet.SetContentsOfCell("i64", "=(i63 * 5.4) / 2.66");
116             // check Changed
117             Assert.IsTrue(sheet.Changed);
118             sheet.SetContentsOfCell("i65", "=i64 - 7.8");
119             // check Changed
120             Assert.IsTrue(sheet.Changed);
121             Formula testFormula = new Formula("(i63*5.4)/2.66");
122             Formula testFormula2 = new Formula("i64 - 7.8");
123             Assert.AreEqual(sheet.GetCellContents("i63"), 32.0);
124             Assert.AreEqual(sheet.GetCellValue("i63"), 32.0);
125             Assert.AreEqual(sheet.GetCellContents("i64"), testFormula);
126             Assert.AreEqual((double)sheet.GetCellValue("i64"), 64.96240601, 0.00000001);
127             Assert.AreEqual(sheet.GetCellContents("i65"), testFormula2);
128             Assert.AreEqual((double)sheet.GetCellValue("i65"), 57.162406015, 0.00000001);
129         }
130
131         /// <summary>
```

```
132         /// Tests a valid cell name entry with double
133         /// </summary>
134         [TestMethod]
135         public void Public_Test_CellContents_Valid_5()
136         {
137             AbstractSpreadsheet sheet = new Spreadsheet();
138             sheet.SetContentsOfCell("TT89", "3990.293823");
139             // check Changed
140             Assert.IsTrue(sheet.Changed);
141             Assert.AreEqual(3990.293823, sheet.GetCellContents("TT89"));
142             Assert.AreEqual(3990.293823, sheet.GetCellValue("TT89"));
143         }
144 
145         /// <summary>
146         /// Tests GetCellContents and GetCellValue on cells that are empty
147         /// </summary>
148         [TestMethod]
149         public void Public_Test_GetCell_Empty_Cells()
150         {
151             AbstractSpreadsheet sheet = new Spreadsheet();
152             Assert.AreEqual("", sheet.GetCellContents("T9"));
153             Assert.AreEqual("", sheet.GetCellContents("HH765"));
154             Assert.AreEqual("", sheet.GetCellValue("T9"));
155             Assert.AreEqual("", sheet.GetCellValue("HH765"));
156         }
157 
158         /// <summary>
159         /// Tests SetContentsOfCell on an invalid cell name with double
160         /// </summary>
161         [TestMethod]
162         [ExpectedException(typeof(InvalidNameException))]
163         public void Public_Test_SetContentsOfCell_Invalid_1()
164         {
165             AbstractSpreadsheet sheet = new Spreadsheet();
166             sheet.SetContentsOfCell("B_0", "13.12");
167         }
168 
169         /// <summary>
170         /// Tests SetContentsOfCell on an invalid cell name with text string
171         /// </summary>
172         [TestMethod]
173         [ExpectedException(typeof(InvalidNameException))]
174         public void Public_Test_SetContentsOfCell_Invalid_2()
175         {
176             AbstractSpreadsheet sheet = new Spreadsheet();
177             sheet.SetContentsOfCell("_A5", "BeetleJuice");
178         }
179 
180         /// <summary>
181         /// Tests a invalid cell name with Formula, no dependencies
182         /// </summary>
183         [TestMethod]
184         [ExpectedException(typeof(InvalidNameException))]
185         public void Public_Test_SetContentsOfCell_Invalid_3()
186         {
187             AbstractSpreadsheet sheet = new Spreadsheet();
188             sheet.SetContentsOfCell("8i63", "=15 - 6.98");
189         }
190 
191         /// <summary>
192         /// Tests an invalid cell name entry with Formula, with dependencies
193         /// </summary>
194         [TestMethod]
195         [ExpectedException(typeof(InvalidNameException))]
196         public void Public_Test_SetContentsOfCell_Invalid_4()
197         {
```

```
198             AbstractSpreadsheet sheet = new Spreadsheet();
199             // this value will be a Formula Error at first just FYI
200             sheet.SetContentsOfCell("i63", "=i65 - 9");
201             sheet.SetContentsOfCell("i65", "23.60");
202             sheet.SetContentsOfCell("&i64", "i63");
203         }
204 
205         /// <summary>
206         /// Tests a valid cell name entry with invalid Formula, no dependencies
207         /// </summary>
208         [TestMethod]
209         [ExpectedException(typeof(FormulaFormatException))]
210         public void Public_Test_SetContentsOfCell_Invalid_5()
211         {
212             AbstractSpreadsheet sheet = new Spreadsheet();
213             sheet.SetContentsOfCell("Xx0", "=3++ 8.6");
214         }
215 
216         /// <summary>
217         /// Tests a valid cell name entry with invalid Formula, with direct dependencies
218         /// </summary>
219         [TestMethod]
220         [ExpectedException(typeof(FormulaFormatException))]
221         public void Public_Test_SetContentsOfCell_Invalid_6()
222         {
223             AbstractSpreadsheet sheet = new Spreadsheet();
224             sheet.SetContentsOfCell("i64", "=8.8/i63");
225             sheet.SetContentsOfCell("i63", "=3.23 + 15)");
226         }
227 
228         /// <summary>
229         /// Tests a valid cell name entry with invalid Formula, with indirect and direct dependencies
230         /// </summary>
231         [TestMethod]
232         [ExpectedException(typeof(FormulaFormatException))]
233         public void Public_Test_SetContentsOfCell_Invalid_7()
234         {
235             AbstractSpreadsheet sheet = new Spreadsheet();
236             sheet.SetContentsOfCell("i63", "16");
237             sheet.SetContentsOfCell("i64", "=(77.9+ i63");
238             sheet.SetContentsOfCell("i65", "=i64");
239         }
240 
241         /// <summary>
242         /// Tests a valid cell name entry with valid Formula, and variable
243         /// names whose cells contains a text string.
244         /// The GetCellValue should return a FormulaError
245         /// </summary>
246         [TestMethod]
247         public void Public_Test_SetContentsOfCell_Invalid_8()
248         {
249             AbstractSpreadsheet sheet = new Spreadsheet();
250             sheet.SetContentsOfCell("X2", "I am Iron Man");
251             sheet.SetContentsOfCell("A1", "=567.0 + X2");
252             Assert.IsTrue(sheet.GetCellValue("A1") is FormulaError);
253         }
254 
255         /// <summary>
256         /// Tests a valid cell name entry with valid Formula, and variable
257         /// names whose cells contains an empty string (hasn't been referenced)
258         /// The GetCellValue should return a FormulaError
259         /// </summary>
260         [TestMethod]
261         public void Public_Test_SetContentsOfCell_Invalid_9()
262         {
263             AbstractSpreadsheet sheet = new Spreadsheet();
```

```
264             sheet.SetContentsOfCell("A1", "=567.0 + UVR77");
265             Assert.IsTrue(sheet.GetCellValue("A1") is FormulaError);
266         }
267 
268         /// <summary>
269         /// Tests a null cell name entry with double
270         /// </summary>
271         [TestMethod]
272         [ExpectedException(typeof(InvalidNameException))]
273         public void Public_Test_SetContentsOfCell_NullName_1()
274         {
275             AbstractSpreadsheet sheet = new Spreadsheet();
276             sheet.SetContentsOfCell(null, "3990.29");
277         }
278 
279         /// <summary>
280         /// Tests a null cell entry with text string
281         /// </summary>
282         [TestMethod]
283         [ExpectedException(typeof(InvalidNameException))]
284         public void Public_Test_SetContentsOfCell_NullName_2()
285         {
286             AbstractSpreadsheet sheet = new Spreadsheet();
287             sheet.SetContentsOfCell(null, "Jim is Great");
288         }
289 
290         /// <summary>
291         /// Tests a null cell entry with Formula, no dependencies
292         /// </summary>
293         [TestMethod]
294         [ExpectedException(typeof(InvalidNameException))]
295         public void Public_Test_SetContentsOfCell_NullName_3()
296         {
297             AbstractSpreadsheet sheet = new Spreadsheet();
298             sheet.SetContentsOfCell(null, "=3.21 * (7)");
299         }
300 
301         /// <summary>
302         /// Tests a null cell name entry with Formula, with indirect and direct dependencies
303         /// </summary>
304         [TestMethod]
305         [ExpectedException(typeof(InvalidNameException))]
306         public void Public_Test_SetContentsOfCell_NullName_4()
307         {
308             AbstractSpreadsheet sheet = new Spreadsheet();
309             sheet.SetContentsOfCell("i63", "=12 - 900.7");
310             sheet.SetContentsOfCell("i64", "=1.2222 / (i63 + 8)");
311             sheet.SetContentsOfCell(null, "=i64 + 12.3");
312         }
313 
314         /// <summary>
315         /// Tests a valid cell entry with null text string
316         /// </summary>
317         [TestMethod]
318         [ExpectedException(typeof(ArgumentNullException))]
319         public void Public_Test_SetContentsOfCell_NullContent()
320         {
321             string hey = null;
322             AbstractSpreadsheet sheet = new Spreadsheet();
323             sheet.SetContentsOfCell("x86", hey);
324         }
325 
326         /// <summary>
327         /// Tests get cell contents on a referenced but altered cell
328         /// (Make sure the cell name wasn't duplicated, but rather the contents
329         /// were just altered)
```

```
330         /// </summary>
331         [TestMethod]
332         public void Public_Test_GetCellContents_AlteredCell()
333         {
334             AbstractSpreadsheet sheet = new Spreadsheet(x => true, x => x.ToUpper(), "Version
    information");
335             sheet.SetContentsOfCell("H45", "=67.5 - 976 / 2");
336             // Normalize should uppercase this before checking to update
337             sheet.SetContentsOfCell("h45", "this cell has changed!");
338             // should be modified, not added
339             Assert.AreEqual("H45", sheet.GetNamesOfAllNonemptyCells().ElementAt(0));
340             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().Count(), 1);
341             Assert.AreEqual("this cell has changed!", sheet.GetCellContents("h45"));
342             Assert.AreEqual("this cell has changed!", sheet.GetCellValue("H45"));
343         }
344 
345         /// <summary>
346         /// Tests the GetNamesOfNonEmptyCells method upon having no non-empty
347         /// cells.
348         /// </summary>
349         [TestMethod]
350         public void Public_Test_GetNamesOfNonEmptyCells_1()
351         {
352             AbstractSpreadsheet sheet = new Spreadsheet();
353             Assert.IsFalse(sheet.GetNamesOfAllNonemptyCells().Any());
354             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().Count(), 0);
355         }
356 
357         /// <summary>
358         /// Tests the GetNamesOfNonEmptyCells method upon having 2 valid non-
359         /// empty cells.
360         /// </summary>
361         [TestMethod]
362         public void Public_Test_GetNamesOfNonEmptyCells_2()
363         {
364             AbstractSpreadsheet sheet = new Spreadsheet();
365             sheet.SetContentsOfCell("FFFo8", "72.001");
366             sheet.SetContentsOfCell("HIJK9", "Shrek");
367             Assert.IsTrue(sheet.GetNamesOfAllNonemptyCells().Any());
368             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().Count(), 2);
369             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().ElementAt(0), "FFFo8");
370             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().ElementAt(1), "HIJK9");
371         }
372 
373         /// <summary>
374         /// Tests the GetNamesOfNonEmptyCells method upon having 5 valid non-
375         /// empty cells.
376         /// </summary>
377         [TestMethod]
378         public void Public_Test_GetNamesOfNonEmptyCells_3()
379         {
380             AbstractSpreadsheet sheet = new Spreadsheet();
381             sheet.SetContentsOfCell("B33", "0.04");
382             sheet.SetContentsOfCell("Iue4", "=477.22+ 0 * (3/55.4 + 8.94)");
383             sheet.SetContentsOfCell("U2", "Get him some melk");
384             sheet.SetContentsOfCell("W2", "Taxes");
385             sheet.SetContentsOfCell("I234567", "=(45)");
386             Assert.IsTrue(sheet.GetNamesOfAllNonemptyCells().Any());
387             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().Count(), 5);
388             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().ElementAt(0), "B33");
389             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().ElementAt(1), "Iue4");
390             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().ElementAt(2), "U2");
391             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().ElementAt(3), "W2");
392             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().ElementAt(4), "I234567");
393         }
394 
```

```
395         /// <summary>
396         /// Tests the GetNamesOfNonEmptyCells method upon having a cell modified to
397         /// contain an empty string (referenced, but still empty)
398         /// cells.
399         /// </summary>
400         [TestMethod]
401         public void Public_Test_GetNamesOfNonEmptyCells_4()
402         {
403             AbstractSpreadsheet sheet = new Spreadsheet();
404             sheet.SetContentsOfCell("E6", "what's in that shoe?");
405             // Modify
406             sheet.SetContentsOfCell("E6", "");
407             Assert.AreEqual(sheet.GetNamesOfAllNonemptyCells().Count(), 0);
408         }
409 
410         /// <summary>
411         /// Test to make sure cells are being recalculated as they are modified
412         /// </summary>
413         [TestMethod]
414         public void Public_Test_RecalculatingCells()
415         {
416             AbstractSpreadsheet sheet = new Spreadsheet();
417             sheet.SetContentsOfCell("A0", "=B0 + B0");
418             sheet.SetContentsOfCell("B0", "45");
419             Assert.AreEqual(90.0, sheet.GetCellValue("A0"));
420         }
421 
422         /// <summary>
423         /// This large test tests the capabilities of the 4-parameter constructor
424         /// and the save method.
425         /// </summary>
426         [TestMethod]
427         public void Public_Test_SaveAndLoad()
428         {
429             // new empty spreadsheet
430             AbstractSpreadsheet sheet1 = new Spreadsheet(x => true, x => x.ToUpper(), "version1");
431             Assert.IsFalse(sheet1.Changed);
432 
433             // add some cells
434             sheet1.SetContentsOfCell("Y6", "=V8*V8");
435             Assert.IsTrue(sheet1.Changed);
436             sheet1.SetContentsOfCell("V8", "=(6+2)");
437             Assert.IsTrue(sheet1.Changed);
438             sheet1.SetContentsOfCell("T2", "Sublime");
439             Assert.IsTrue(sheet1.Changed);
440 
441             // save the spreadsheet
442             sheet1.Save("sheet1.xml");
443             Assert.IsFalse(sheet1.Changed);
444 
445             // new spreadsheet from the first
446             AbstractSpreadsheet sheet2 = new Spreadsheet("sheet1.xml", x => true, x => x.ToUpper(),     ↙
    "version1");
447             Assert.IsFalse(sheet2.Changed);
448 
449             // check cells were copied correctly
450             Assert.AreEqual(new Formula("V8*V8"), sheet2.GetCellContents("Y6"));
451             Assert.IsFalse(sheet2.Changed);
452             Assert.AreEqual(64.0, sheet2.GetCellValue("Y6"));
453             Assert.IsFalse(sheet2.Changed);
454             Assert.AreEqual(new Formula("(6+2)"), sheet2.GetCellContents("V8"));
455             Assert.IsFalse(sheet2.Changed);
456             Assert.AreEqual(8.0, sheet2.GetCellValue("V8"));
457             Assert.IsFalse(sheet2.Changed);
458             Assert.AreEqual("Sublime", sheet2.GetCellContents("T2"));
459             Assert.IsFalse(sheet2.Changed);
```

```
460             Assert.AreEqual("Sublime", sheet2.GetCellValue("T2"));
461
462             // add and update some cells
463             sheet2.SetContentsOfCell("BB7", "56.17");
464             sheet2.SetContentsOfCell("V8", "12.3");
465
466             // save this one
467             sheet2.Save("sheet2.xml");
468
469             // one more spreadsheet
470             AbstractSpreadsheet sheet3 = new Spreadsheet("sheet2.xml", x => true, x => x.ToUpper(),
    "version1");
471
472             // check cells were copied and updated correctly
473             Assert.AreEqual(new Formula("V8*V8"), sheet3.GetCellContents("Y6"));
474             Assert.AreEqual(151.29, (double)sheet3.GetCellValue("Y6"), 0.00000001);
475             Assert.AreEqual(12.3, sheet3.GetCellContents("V8"));
476             Assert.AreEqual(12.3, sheet2.GetCellValue("V8"));
477             Assert.AreEqual("Sublime", sheet2.GetCellContents("T2"));
478             Assert.AreEqual("Sublime", sheet2.GetCellValue("T2"));
479             Assert.AreEqual(56.17, sheet3.GetCellContents("BB7"));
480             Assert.AreEqual(56.17, sheet3.GetCellValue("bb7"));
481
482             // save
483             sheet3.Save("sheet3.xml");
484         }
485
486         /// <summary>
487         /// Tests the 4-parameter constructor and makes sure it throws exception
488         /// upon receiving mismatched version information.
489         /// </summary>
490         [TestMethod]
491         [ExpectedException(typeof(SpreadsheetReadWriteException))]
492         public void Public_Test_Incorrect_Version()
493         {
494             AbstractSpreadsheet sheet = new Spreadsheet(x => true, x => x, "version 34");
495             sheet.SetContentsOfCell("A8", "Hello Mr. Rogers");
496             sheet.Save("Rogers.xml");
497
498             Assert.AreEqual("version 34", sheet.GetSavedVersion("Rogers.xml"));
499
500             AbstractSpreadsheet sheet1 = new Spreadsheet("Rogers.xml", x => true, x => x, "version 35")
    ;
501         }
502
503         /// <summary>
504         /// Tests the capabilities of the GetSavedVersion method
505         /// </summary>
506         [TestMethod]
507         public void Public_Test_GetSavedVersion_1()
508         {
509             AbstractSpreadsheet sheet1 = new Spreadsheet();
510             sheet1.Save("firstsheet.xml");
511             Assert.AreEqual("default", sheet1.GetSavedVersion("firstsheet.xml"));
512
513             AbstractSpreadsheet sheet2 = new Spreadsheet(x => true, x => x, "version 16");
514             sheet2.Save("secondsheet.xml");
515             // see how we can access this from any sheet
516             Assert.AreEqual("version 16", sheet1.GetSavedVersion("secondsheet.xml"));
517         }
518
519         /// <summary>
520         /// Tests the capabilities of the GetSavedVersion method upon incorrect
521         /// file / empty file
522         /// </summary>
523         [TestMethod]
```

```
524         [ExpectedException(typeof(SpreadsheetReadWriteException))]
525         public void Public_Test_GetSavedVersion_2()
526         {
527             AbstractSpreadsheet sheet1 = new Spreadsheet();
528             sheet1.GetSavedVersion("EmptyFile.xml");
529         }
530 
531         /// <summary>
532         /// Tests the GetDirectDependents method with one dependent.
533         /// </summary>
534         [TestMethod]
535         public void Protected_Test_GetDirectDependents_1()
536         {
537             base.SetContentsOfCell("t64", "6.1");
538             base.SetContentsOfCell("U750", "=t64 + 7");
539             Assert.AreEqual(base.GetDirectDependents("t64").Count(), 1);
540             Assert.AreEqual(base.GetDirectDependents("t64").ElementAt(0), "U750");
541         }
542 
543         /// <summary>
544         /// Tests the GetDirectDependents method with a few dependents.
545         /// </summary>
546         [TestMethod]
547         public void Protected_Test_GetDirectDependents_2()
548         {
549             base.SetContentsOfCell("A1", "19.776");
550             base.SetContentsOfCell("A2", "1");
551             base.SetContentsOfCell("A3", "=A1 / A2");
552             base.SetContentsOfCell("A4", "=A3 + 7.8");
553             Assert.AreEqual(base.GetDirectDependents("A1").Count(), 1);
554             Assert.AreEqual(base.GetDirectDependents("A2").Count(), 1);
555             Assert.AreEqual(base.GetDirectDependents("A3").Count(), 1);
556             Assert.AreEqual(base.GetDirectDependents("A4").Count(), 0);
557             Assert.AreEqual(base.GetDirectDependents("A1").ElementAt(0), "A3");
558             Assert.AreEqual(base.GetDirectDependents("A2").ElementAt(0), "A3");
559             Assert.AreEqual(base.GetDirectDependents("A3").ElementAt(0), "A4");
560         }
561 
562         /// <summary>
563         /// Tests the GetDirectDependents method with no dependents.
564         /// </summary>
565         [TestMethod]
566         public void Protected_Test_GetDirectDependents_3()
567         {
568             base.SetContentsOfCell("Y1", "6.1");
569             base.SetContentsOfCell("Y2", "=9 + 7");
570             Assert.AreEqual(base.GetDirectDependents("Y1").Count(), 0);
571             Assert.AreEqual(base.GetDirectDependents("Y2").Count(), 0);
572         }
573 
574         /// <summary>
575         /// Tests the GetDirectDependents method with null name.
576         /// </summary>
577         [TestMethod]
578         [ExpectedException(typeof(ArgumentNullException))]
579         public void Protected_Test_GetDirectDependents_4()
580         {
581             base.GetDirectDependents(null);
582         }
583 
584         /// <summary>
585         /// Tests the GetDirectDependents method with invalid cell name.
586         /// </summary>
587         [TestMethod]
588         [ExpectedException(typeof(InvalidNameException))]
589         public void Protected_Test_GetDirectDependents_5()
```

```
590         {
591             base.GetDirectDependents("$33");
592         }
593 
594         /// <summary>
595         /// Tests the GetCellsToRecalculate method with a few dependents.
596         /// No Circular Dependencies
597         /// </summary>
598         [TestMethod]
599         public void Protected_Test_GetCellsToRecalculate_1()
600         {
601             base.SetContentsOfCell("A1", "5");
602             base.SetContentsOfCell("B1", "7");
603             base.SetContentsOfCell("C1", "=A1 + B1");
604             base.SetContentsOfCell("D1", "=B1 * C1");
605             base.SetCellContents("E1", "=15");
606             IEnumerable<string> recalculate = base.GetCellsToRecalculate("A1");
607             Assert.AreEqual(recalculate.Count(), 3);
608             var cells = recalculate.Cast<string>().Take(4).ToArray();
609             CollectionAssert.AreEqual(cells, new[] { "A1", "C1", "D1" });
610         }
611 
612         /// <summary>
613         /// Tests the GetCellsToRecalculate method with a few dependents.
614         /// No Circular Dependencies
615         /// </summary>
616         [TestMethod]
617         public void Protected_Test_GetCellsToRecalculate_2()
618         {
619             base.SetContentsOfCell("Y0", "5");
620             base.SetContentsOfCell("Y1", "=Y0");
621             base.SetContentsOfCell("Y2", "=Y0 + Y1");
622             base.SetContentsOfCell("Y3", "=Y2 * 3");
623             base.SetContentsOfCell("Y4", "=Y3");
624             IEnumerable<string> recalculate = base.GetCellsToRecalculate("Y0");
625             Assert.AreEqual(recalculate.Count(), 5);
626             var cells = recalculate.Cast<string>().Take(5).ToArray();
627             CollectionAssert.AreEqual(cells, new[] { "Y0", "Y1", "Y2", "Y3", "Y4" });
628         }
629 
630         /// <summary>
631         /// Tests the GetCellsToRecalculate method with a few dependents.
632         /// No Circular Dependencies
633         /// </summary>
634         [TestMethod]
635         public void Protected_Test_GetCellsToRecalculate_3()
636         {
637             base.SetContentsOfCell("F0", "5");
638             base.SetContentsOfCell("G0", "=F0");
639             base.SetContentsOfCell("H0", "=14");
640             base.SetContentsOfCell("J0", "=G0 * 3");
641             base.SetContentsOfCell("K0", "=(8+F0) - 7.9");
642             IEnumerable<string> recalculate = base.GetCellsToRecalculate("F0");
643             Assert.AreEqual(recalculate.Count(), 4);
644             var cells = recalculate.Cast<string>().Take(4).ToArray();
645             // remember, direct dependencies are added to list first **
646             CollectionAssert.AreEqual(cells, new[] { "F0", "K0", "G0", "J0" });
647         }
648 
649         /// <summary>
650         /// Tests the GetCellsToRecalculate method with small number of dependents.
651         /// should throw Circular Dependency Exception
652         /// </summary>
653         [TestMethod]
654         [ExpectedException(typeof(CircularException))]
655         public void Protected_Test_GetCellsToRecalculate_4()
```

```
656         {
657             AbstractSpreadsheet sheet = new Spreadsheet();
658             sheet.SetContentsOfCell("F0", "13");
659             sheet.SetContentsOfCell("G0", "=F0");
660             sheet.SetContentsOfCell("F0", "=G0");
661         }
662 
663         /// <summary>
664         /// Tests the GetCellsToRecalculate method with cell depending on itself
665         /// which is not allowed -
666         /// should throw Circular Dependency Exception
667         /// </summary>
668         [TestMethod]
669         [ExpectedException(typeof(CircularException))]
670         public void Protected_Test_GetCellsToRecalculate_5()
671         {
672             AbstractSpreadsheet sheet = new Spreadsheet();
673             sheet.SetContentsOfCell("TTT4", ("=TTT4"));
674         }
675     }
676 }
677 
```