The Clean Coder

A notebook for software craftsmen and engineers.

Sunday, October 3, 2010

The Craftsman 62, The Dark Path.

Fri, 17 Mar 2002, 14:00

"Hey Alphonse," Jerry called as I walked by, "let's do a bit of practice. I've got a kata I'd like to show you."

I felt I could use the break so I walked over and sat next to Jerry.

"Sure, Jerry, what's a Kata?"

Jerry rolled his eyes. "You've never done a *kata*?"

I could feel my guard going up, but I tried to relax. "No, can't say I have."

Jerry smirked and then called over to Jasmine: "Hay Jaz, do you want to tell Alphonse what a kata is?"

Jasmine's long dark hair swished playfully as she turned her head to face me. She nailed me with those sparkling green eyes as she answered: "What, the hotshot's never done a kata?"

"He says not. Can you believe it?"

"Jeez, what do they teach these kids nowadays?"

"Oh come on!" I said, starting to get annoyed. "You guys are only a couple of years older than me, school hasn't changed that much."

Jasmine smiled at me, and I felt my annoyance evaporate. That smile... "Relax Alphonse, we're just poking fun at you. A kata is just a simple program that you write over and over again as a way to practice. We do them all the time. It's part of our normal discipline."

"You write the same code over and over?" This was new to me, and it didn't make a lot of sense.

Jerry nodded and explained: "Yeah. Sometimes we'll do a kata two are three times in a row, exactly the same each time. It's a good way to practice your hot-keys."

"And sometimes," Jasmine added, "we solve them in different ways, using different techniques as a way to learn different approaches and reinforce our disciplines."

"And sometimes we just do them for fun." Jerry concluded.

"Which one are you going to show him?" Jasmine asked.

"I was thinking about doing 'Word Wrap'."

► 2011 (8)
▼ 2010 (24)
► December (9)
► November (5)
▼ October (4)
Danger! Software Craftsmen at Work.
The Cost of Code?
Be a good one. #scna 2010
The Craftsman 62, The Dark Path.
► September (4)
► August (2)

Followers

"Oh, that's a *good* one. You're going to like this Alphonse. You guys have fun." And with that she turned back to her work.

I turned to Jerry and asked: "Word Wrap?"

"Yeah, it's a simple problem to understand, but it's oddly difficult to solve. The basic premise is really simple. You write a class called **`Wrapper`**, that has a single static function named **`wrap`** that takes two arguments, a string, and a column number. The function returns the string, but with line breaks inserted at just the right places to make sure that no line is longer than the column number. You try to break lines at word boundaries."

I thought about this for a moment and then said: "You mean like a word processor, right? You break the line by replacing the last space in a line with a newline."

Jerry nodded. "Yeah that's the idea. Pretty simple huh?"

I shrugged. "Sounds simple, yes."

Jerry pushed the keyboard in my direction. "OK, then why don't you start."

I knew this was a trap of some kind, but I couldn't see how. So I said: "I suppose we should begin with simple degenerate tests." I took the keyboard and began to type. I got the first few tests passing as Jerry watched.

```
1  @RunWith(Suite.class)
2  @Suite.SuiteClasses({
3    WrapperTest.DegenerateTests.class})
4
5  public class WrapperTest {
6    public static class DegenerateTests {
7      @Test
8      public void emptyString() throws Exception {
```

```
9         assertThat(wrap("", 1), equalTo(""));
10      }
11
12      @Test
13      public void stringShorterThanCol() throws Exception {
14        assertThat(wrap("this", 10), equalTo("this"));
15      }
16    }
17  }
```

1 WrapperTest.java hosted with by GitHub view raw

```
1  public class Wrapper {
2    public static String wrap(String s, int col) {
3       return s;
4    }
5  }
```

2 Wrapper.java hosted with by GitHub view raw

Jerry got real interested as I wrote this. When I got the second test working he said: "What's all that **`@RunWith`** and **`@Suite`** stuff you are typing?"

I smiled. Apparently I was about to teach Jerry something. "Oh, yeah." I said nonchalantly. "That's the *TestNest* pattern. I learned it from Justin a few days ago. It lets you put more than one test class in a file. Each test class can have it's own setups and teardowns."

"Yeah, that's kind of slick. But who's this Justin dude?"

I pointed and counted ceiling lights. "He works just down the hall, beneath the 8th light."?

"You mean by those guys who are always walking on treadmills while they code?"

I nodded and kept on coding while Jerry stared back down the hall and recounted the lights.

```java
@RunWith(Suite.class)
@Suite.SuiteClasses({
  WrapperTest.DegenerateTests.class,
  WrapperTest.wrapWordsTest.class
})

public class WrapperTest {
  public static class DegenerateTests {
    @Test
    public void emptyString() throws Exception {
      assertThat(wrap("", 1), equalTo(""));
    }

    @Test
    public void stringShorterThanCol() throws Exception {
      assertThat(wrap("this", 10), equalTo("this"));
    }
  }

  public static class wrapWordsTest {
    @Test
    public void wrapTwoWordsAfterSpace() throws Exception {
      assertThat(wrap("word word", 6), equalTo("word\nword"));
    }

    @Test
    public void wrapThreeWordsAfterFirstSpace() throws Exception {
      assertThat(wrap("word word word", 6), equalTo("word\nword\nword"))
    }
  }
}
```

1 WrapperTest.java hosted with by GitHub view raw

```
public class Wrapper {
  public static String wrap(String s, int col) {
   if (s.length() <= col)
     return s;
    else
     return s.replaceAll(" ", "\n");
  }
}
```

2 Wrapper.java hosted with by GitHub view raw

Jerry looked back just in time to see that last test pass. He looked over the code and nodded. "Yes, that's just about exactly how I solved it the first time. That **`replaceAll`** is a bit of a hack isn't it."

"Yes, but it gets the test to pass. 'First make it work, then make it right.'"

Jerry nodded.

"Anyway, it's pretty straightforward so far." I said. And so I went on to write the next test.

```
    @Test
    public void wrapThreeWordsAfterSecondSpace() throws Exception {
      assertThat(wrap("word word word", 11), equalTo("word word\nword"));
    }
```

1 WrapperTest.java hosted with by GitHub view raw

```
java.lang.AssertionError:
Expected: "word word\nword"
     got: "word\nword\nword"
```

2 Test Results hosted with by GitHub view raw

Jerry nodded sagely. "Yes, that's the obvious next test."

"Yes, and with the obvious failure." I agreed. So then I looked back at the code.

I stared at it for a long time. But there did not seem to be any simple thing that I could do to make the test pass.

After a few minutes, Jerry said: "What's the matter Alphonse? Stuck?"

"No, this should be simple. I just..." In frustration I took the keyboard and began to type. I typed for quite a while, adding and erasing code. Jerry nodded knowingly, and sometimes grunted. After about five minutes Jerry stopped me. The code looked like this:

```
1  public class Wrapper {
2    public static String wrap(String s, int col) {
3      if (s.length() <= col)
4        return s;
5      else {
6        int lastSpace = 0;
7        int space;
8        while ((space = s.indexOf(" ", lastSpace)) != -1) {
9          if (space > col) {
10           s = s.substring(0, lastSpace) + "\n" + s.substring(lastSpace+1
11           //todo this doesn't look right.
12         }
13         lastSpace = space;
14       }
15       return s; // really?
16     }
17
18 //     return s.replaceAll(" ", "\n");
19   }
```

```
20 }
```

Wrapper.java hosted with by **GitHub** **view raw**

"Are you sure you're on the right track, Alphonse?"

I looked at the code and realized that I had been coding blindly. I ran the tests in desperation, but they just hung in an infinite loop. I could *kind of feel* what needed to be done, but it wasn't at all clear how I should proceed.

"Give me another shot at this." I said, as I erased all the code and started over. Jerry just smiled and watched as I flailed around for another five minutes or so. Finally, with lots of tests failing he stopped me again.

```
1  public class Wrapper {
2    public static String wrap(String s, int col) {
3      if (s.length() <= col)
4        return s;
5      else {
6        StringBuilder builder = new StringBuilder();
7        String[] strings = s.split(" ");
8        int column = 0;
9        for (String segment : strings) {
10         column += segment.length();
11         if (column < col)
12           builder.append(segment+" ");
13         else
14           builder.append(segment+"\n");
15       }
16       return builder.toString();
17     }
18
19 //     return s.replaceAll(" ", "\n");
20   }
```

```
21 }
```

Wrapper.java hosted with by **GitHub** **view raw**

"What are you doing wrong, Alphonse?"

I stared at the screen for a minute. Then I said: "I *know* I can get this working, give me another shot."

"Alphonse, I *know* you can get it working too; but that's not the point. Stop for a minute and tell me what you are doing wrong."

I could hear Jasmine stifling a giggle. I looked over at her, but she didn't meet my eye. Then I took my fingers off the keyboard and hung my head. "I can't seem to get this test to pass without writing a lot of untested code." I said.

"That's true." Said Jerry, but it's not quite the answer I was looking for. You were doing something wrong. Something *really* wrong. Do you know what it was?

I thought about it for awhile. I had been trying to figure out the algorithm. I had tried lots of different approaches. But all my guesses turned out wrong. -- Oh!

I looked Jerry square in the eye and said: "I was guessing."

"Right!" Jerry beamed. "And why were you guessing?"

"Because the test was failing and I couldn't figure out how to get it to pass."

Now Jerry narrowed his gaze, almost like he was looking *through* me. "And what does that tell you?"

"That the problem is hard?" I guessed.

"No, Alphonse, the problem is *not* hard. When you see the solution, you're going to be very angry at yourself. The reason you could not figure out how to pass that test, Alphonse, is that you were trying to pass the *wrong* test."

I looked at the tests again. They seemed perfectly logical. So I asked Jerry: "How could these be the wrong tests?"

Jerry smiled with a grin that rivaled Jasper's. "They are the wrong tests, Alphonse, because you could not figure out how to pass them."

I gave Jerry a stern look. "You're being circular, Jerry."

"Perhaps I am. Look at it this way. The test you are trying to pass is forcing you to solve a very large part of the problem. Indeed, it might just be the *whole* problem. In any case, the bite you are taking is too big."

"Yeah, but..."

Jerry stopped me and said: "Did you ever read *The Moon is a Harsh Mistress* Alphonse?"

"Uh... Heinlein, wasn't it? Yes, I read it a few years back. It was a great story."

"Indeed it was. Do you remember this quotation?"

> "[W]hen faced with a problem you do not understand, do any part of it you do understand, then look at it again."

"As a matter of fact, I do. I thought it was very profound."

"OK then Alphonse, apply that here. Find some part of this problem that you *do* understand."

"I understand the problem..."

"No, you *think* you understand the problem, but clearly you don't. If you understood it, you'd be able to solve it. Find some simpler tests to pass."

I thought about this for a few seconds. What was so hard about this problem? The thing I'd been struggling with was how to deal with breaking the lines at spaces? Each of my "solutions" was tangled up with hunting for just the right space to replace with a line end.

I looked at Jerry and said: "What if I solved the part of this problem that did not deal with spaces? Lines that have no spaces only need to be broken once they've hit the column limit."

Jerry pointed at the keyboard, and I started again. I wrote the same degenerate tests.

```
1   public static class DegenerateTests {
2     @Test
3     public void wrap_EmptyString_ShouldBeEmpty() throws Exception {
4       assertThat(wrap("", 1), equalTo(""));
5     }
6
7     @Test
8     public void stringShorterThanColDoesNotWrap() throws Exception {
9       assertThat(wrap("word", 10), equalTo("word"));
10    }
11  }
```

1 WrapperTest.java hosted with by GitHub view raw

```
1  public class Wrapper {
2    public static String wrap(String s, int col) {
3        return s;
4    }
5  }
```

2 Wrapper.java hosted with by GitHub view raw

But then I changed tack and wrote a test that wrapped a line without spaces. That test was trivially easy to pass.

```
  public static class SplitWordTests {
    @Test
    public void splitOneWord() throws Exception {
      assertThat(wrap("word", 2), equalTo("wo\nrd"));
    }
  }
```

1 WrapperTest.java hosted with by GitHub view raw

```
public class Wrapper {
  public static String wrap(String s, int col) {
    if (s.length() <= col)
      return s;
    else
      return (s.substring(0, col) + "\n" + s.substring(col));
  }
}
```

2 Wrapper.java hosted with by GitHub view raw

The next test was pretty obvious. It should continue to wrap a string without spaces, creating lines that are no longer than the column limit.

```
    @Test
    public void splitOneWordManyTimes() throws Exception {
      assertThat(wrap("abcdefghij", 3), equalTo("abc\ndef\nghi\nj"));
```

```
    }
```

WrapperTest.java hosted with by **GitHub** **view raw**

Jerry looked at the test and nodded. "How will you solve that, Alphonse?"

"I just need to put a loop into the wrap function." I said.

"I think there's an easier way." He said.

I looked at the code for a bit, and then said: "Oh! Sure, I could recurse."

```
public class Wrapper {
  public static String wrap(String s, int col) {
    if (s.length() <= col)
      return s;
    else
      return (s.substring(0, col) + "\n" + wrap(s.substring(col), col));
  }
}
```

Wrapper.java hosted with by **GitHub** **view raw**

The tests passed, and Jerry nodded approvingly. "That looks like a framework you could build upon. What's next?"

"Now that I can wrap lines *without* spaces, it ought to be easier to wrap lines *with* spaces!"

"Give it a shot." He said. So I wrote the simplest test I could. A space right at the column limit.

```
public static class WrapTwoWords {
  @Test
  public void wrapOnWordBoundary() throws Exception {
    assertThat(wrap("word word", 5), equalTo("word\nword"));
  }
}
```

WrapperTest.java hosted with by **GitHub** **view raw**

"Do you remember how you made that test pass last time?" Jerry asked.

"Yeah." I grimaced. "I use the **`replaceAll`** hack."

"Is that how you're going to solve it now?"

I looked at the code, and the answer was obvious. "Of course not!" I exclaimed. "All I need to do is check to see if the character at the column limit is a space!" and I wrote the following code.

```
public class Wrapper {
  public static String wrap(String s, int col) {
    if (s.length() <= col)
      return s;
    else if (s.charAt(col-1) == ' ')
      return (s.substring(0, col-1) + "\n" + wrap(s.substring(col), col)
    else
      return (s.substring(0, col) + "\n" + wrap(s.substring(col), col));
  }
}
```

Wrapper.java hosted with by **GitHub** **view raw**

"Why'd you put that wrap call in there?" Jerry asked. "You're getting a little ahead of yourself, aren't you?"

"I guess, but it's kind of *obvious* that it belongs there. Just look at the symmetry!"

"I agree." Jerry said smiling. "Continue on."

The next test was just as obvious. The space should be *before* the column limit. So I typed the following:

```
1     @Test
2     public void wrapAfterWordBoundary() throws Exception {
3       assertThat(wrap("word word", 6), equalTo("word\nword"));
4     }
```

WrapperTest.java hosted with by **GitHub** **view raw**

"Passing this one is going to be tricky." I said under my breath.

"Is it?" Jerry queried.

I looked again, and it hit me. "Oh, no, it's just a small change!" And I typed the following.

```
1  public class Wrapper {
2    public static String wrap(String s, int col) {
3      if (s.length() <= col)
4        return s;
5      int space = (s.substring(0, col).lastIndexOf(' '));
6      if (space != -1)
7        return (s.substring(0, space) + "\n" + wrap(s.substring(space+1),
8      else
9        return (s.substring(0, col) + "\n" + wrap(s.substring(col), col));
10   }
```

```
11  }
```

Wrapper.java hosted with by **GitHub** **view raw**

The tests passed, and I was getting excited. "This is so strange, the whole algorithm is just falling into place!"

"When you choose the right tests, Alphonse, they usually do."

"OK, so now let's make the column boundary really small so that it has to chop the string up into lots of little lines."

```
1     @Test
2     public void wrapWellBeforeWordBoundary() throws Exception {
3       assertThat(wrap("word word", 3), equalTo("wor\nd\nwor\nd"));
4     }
```

WrapperTest.java hosted with by **GitHub** **view raw**

"That one passes right away!" I said. Wow, I think we're done.

"Not quite." Jerry said. "There's another case."

I studied the tests. "Oh, there's the case where the character *after* the column limit is a space." I wrote the tests, and it was trivial to pass.

```
1     @Test
2     public void wrapJustBeforeWordBoundary() throws Exception {
3       assertThat(wrap("word word", 4), equalTo("word\nword"));
4     }
```

1 **WrapperTest.java** hosted with by **GitHub** **view raw**

```
public class Wrapper {
  public static String wrap(String s, int col) {
    if (s.length() <= col)
      return s;
    int space = (s.substring(0, col).lastIndexOf(' '));
    if (space != -1)
      return (s.substring(0, space) + "\n" + wrap(s.substring(space + 1)
    else if (s.charAt(col) == ' ')
      return (s.substring(0, col) + "\n" + wrap(s.substring(col + 1), co
    else
      return (s.substring(0, col) + "\n" + wrap(s.substring(col), col));
  }
}
```

2 **Wrapper.java** hosted with by **GitHub** **view raw**

Jerry smiled as the tests passed. "That's the algorithm all right. But I bet you could clean this up a bit."

"Yeah, there *is* a lot of duplication in there." So I cleaned up my work with the following result.

```
public class Wrapper {
  public static String wrap(String s, int col) {
    return new Wrapper(col).wrap(s);
  }

  private int col;

  private Wrapper(int col) {
    this.col = col;
```

```
10    }
11
12    private String wrap(String s) {
13      if (s.length() <= col)
14        return s;
15      int space = (s.substring(0, col).lastIndexOf(' '));
16      if (space != -1)
17        return breakLine(s, space, 1);
18      else if (s.charAt(col) == ' ')
19        return breakLine(s, col, 1);
20      else
21        return breakLine(s, col, 0);
22    }
23
24    private String breakLine(String s, int pos, int gap) {
25      return s.substring(0, pos) + "\n" + wrap(s.substring(pos + gap), col
26    }
27  }
```

Wrapper.java hosted with by **GitHub** **view raw**

I looked at the code in some astonishment. This really was a very simple algorithm! Why couldn't I see it before?

"You were right." I said to Jerry. "Now that I see this algorithm for what it is, it's kind of obvious. I guess choosing the right tests is pretty important."

"It's not so much choosing the *right* tests, Alphonse; it's about realizing that you are trying to solve the *wrong* test."

"Yeah, the next time I get stuck like that, and start guessing and flailing, I'm going to re-evaluate the tests. Perhaps there'll be simpler tests that will give me a clue about the real solution."

And then I stopped myself and asked: "Is that true, Jerry? Is there always a simpler test that'll get me unstuck?"

Jerry was about to answer when a spitwad hit him in the side of the face. Jasmine was laughing and running down the hall. Jerry lept out of his seat to chase after her.

I just shook my head and wondered.

Posted by Uncle Bob at 2:35 PM

23 comments:

Meister October 3, 2010 at 3:05 PM

Such a great post! I've just finished reading TDD by Kent Beck and wanted to try it out. I had the first few passing tests really quickly, but afterwords I just couldn't get the next test FAIL! Fail u say? Exactly fail. (The reason for that was that i was finished, without having all the cases tested, but the reason for this reason was having wrong tests in the first place).

Reply

Dan Bergh Johnsson October 3, 2010 at 11:51 PM

Excellent post! The concept of "smallest bite" have always been central to my understanding of TDD. This post makes a wonderful point in stressing that choosing a "bite small enough to swallow" is not all about the test itself - it is about the solution it drives. I will definitely include a ref to this post in my future classes on TDD.

Reply

pellepim October 4, 2010 at 12:00 AM

Hahahahaah! This and some cinnamon buns totally made my monday morning. Fantastic funny and educational read. Thanks!

Reply

EastarLee October 4, 2010 at 12:08 AM

Great Post!

I've practiced TDD for two years but still a newbie.

Realizing that you are trying to solve the wrong test is pretty important.

Reply

helino October 4, 2010 at 1:17 AM

I've never commented on a blog post before, but I have to say, this was a very nice post! I learned a lot reading through it, thank you!

Reply

rainerhilmer October 4, 2010 at 3:21 AM

Wow this is not only a great story. I also like it because it's written like an entertaining novel. This could be a good way to design tutorials in general! I will keep that in mind.
What's even more important to me is the fact that this is a great way to teach TDD newbies!

I will bookmark this blog post and the next time someone asks me, "hey what's that TDD thingy all about?", I will give him this link. :-)

Greetings,
Rainer Hilmer

Reply

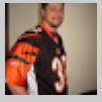
David Saff October 4, 2010 at 5:37 AM

FYI, TestNest is built into JUnit:

http://kentbeck.github.com/junit/javadoc/latest/org/junit/experimental/runners/Enclosed.html

Yes, there needs to be better docs...

Reply



Matt Rutledge October 5, 2010 at 3:48 PM

Where can I find the Craftsman articles #53 to #61?

I looked at http://objectmentor.com/resources/publishedArticles.html under the Crafstman link but that only goes up to 52.

I have read all of them up to 52 and then this one came out and it is 63. I feel like I have missed out on some of the Ruby exploration that 52 left on.

Reply

Carlos Peix October 7, 2010 at 2:10 AM

Hi Matt,

I asked Bob via Twitter and he answered this: "You'll find them at Informit. http://bit.ly/cM9bgW Clean Code Tips 1-12"

Good luck

Reply

tieTYT October 8, 2010 at 11:13 AM

Just thought you should know these code snippets don't show up in the RSS feed (at least not with google reader).

Reply

Bahador October 10, 2010 at 8:00 PM

Thanks!
A thought-provoking post

Reply

Tobias Schulte October 17, 2010 at 3:30 PM

assertThat(wrap("word word word", 9), equalTo("word word\nword"));

does not work.

int space = (s.substring(0, col+1).lastIndexOf(' '));

and removing the else-if does the trick.

Reply

Matt Rutledge October 20, 2010 at 8:49 AM

Thanks Carlos! Now I can catch up on the ones I missed.

Reply

Dennis van der Stelt October 22, 2010 at 7:28 AM

What Tobias said, I got automatically from the tests and he's right. Here's my code, I hope it's received well... Oh, and it's C#

```
namespace WordWrapKata
{
public class Wrapper
{
public static string Wrap(string value, int column)
{
if (column >= value.Length)
return value;

int space = value.Substring(0, column + 1).LastIndexOf(' ');

if (space != -1)
return value.Substring(0, space) + "\n" + Wrap(value.Substring(space + 1), column);

return value.Substring(0, column) + "\n" + Wrap(value.Substring(column), column);
}
}

[TestClass]
```

```
public class WordWrapTests
{
private const string STR_TellusIsEenMooiBedrijf = "Tellus is een mooi bedrijf";

[TestMethod]
public void Does_ReturnEmptyString_When_EmptyStringProvided()
{
// Act
string result = Wrapper.Wrap("", 10);

// Assert
Assert.AreEqual(string.Empty, result);
}

[TestMethod]
public void Does_ReturnSameString_When_ColumnIsHigherThanLengthOfString()
{
// Act
string result = Wrapper.Wrap(STR_TellusIsEenMooiBedrijf,
STR_TellusIsEenMooiBedrijf.Length + 1);

// Assert
Assert.AreEqual(STR_TellusIsEenMooiBedrijf, result);
}

[TestMethod]
public void Does_SplitAtExactColumn_When_NoSpacesProvided()
{
// Act
string result = Wrapper.Wrap("WordWordWord", 4);

// Assert
Assert.AreEqual("Word\nWord\nWord", result);
```

```
}

[TestMethod]
public void Does_SplitAtSpace_When_WithinTheSmallestString()
{
// Act
string result = Wrapper.Wrap("Word Word Word", 4);

// Assert
Assert.AreEqual("Word\nWord\nWord", result);
}

[TestMethod]
public void Does_EverythingCorrect_When_ProvidingComplexAndLastTest()
{
// Act
string result = Wrapper.Wrap(STR_TellusIsEenMooiBedrijf, 6);

// Assert
Assert.AreEqual("Tellus\nis een\nmooi\nbedrij\nf", result);
}
}
}
```

Reply

Dennis van der Stelt October 22, 2010 at 7:28 AM

Oh, and something else, the original code isn't thread safe with the static member variable. My code is! :)

Reply

Dennis van der Stelt October 22, 2010 at 7:32 AM

Just paste the code above in a test project, btw.

Once you get the recursion, I guess the solution is almost always the same. Not that it matters though. I'll try again in two weeks or so, see what happens than. I did not know this kata yet. Great story btw! Especially the ending! :)

Reply

Mamuf December 8, 2010 at 1:41 PM

Great article, great story. Although I've read this via Instapaper and the code snippets were also missing like in the RSS. But it wasn't a real problem. The TDD part of the story is very insightful but you really got me in the first few paragraphs with the code katas!

It reminded me of my old friend I met in school years ago. He was actually the one who thaught me to program for the first time. But what he did was that he almost constantly programmed the game of snake. You know the one where you control a snake on the X and Y axis and it eats food and growths in length etc. He must have written it many times again and again from the ground up through the years.

Now I think it's never too late to start doing code katas so I will start with them now.

Reply

Thomas ten Cate July 6, 2012 at 2:55 AM

That recursion doesn't look like a tail call. So now test it with a string that should become 10000 lines and see your call stack overflow...

Reply

Joe Bowbeer December 24, 2012 at 11:51 PM

A shorter, more efficient last-index computation:

int space = s.lastIndexOf(' ', col);

Reply

thedeemon March 2, 2013 at 9:05 AM

>That's the algorithm all right.

This is the funniest part because the algorithm is not right at all, as Tobias noticed. This is a brilliant story showing that TDD should not be used for algorithms development.

Reply

Dos_Santos August 20, 2013 at 7:19 AM

Hey, I just checked out the Transformation Priority and Sorting, but, what if you introduce a test with repeated numbers?, like [4,4,4,4] (or something like that). Cause to me it is a fail test.

Reply

Ralf Westphal - One Man Think Tank August 26, 2013 at 11:40 AM

There is another case where the solution seems to fail. Consider this:

Wrapper.wrap("line1 l ine2", 7)

It returns "line1\nl ine2" instead of "line1 l\nine2".

I guess, the quality of a TDD solution really depends on the test cases it´s based on. So the question is: how can you come up with a list of prioritized test cases if you don´t understand the problem and/or don´t have a clue as to how the solution should look like? Because that seems to be the case in this demonstration.

This is not to blame Uncle Bob. Rather I guess that´s so often the case we need to be aware of it and don´t just trust some TDD process. TDD does not solve our problems. It´s only as good as our understanding of problem domain and solution are before (!) we start coding.

Reply

mikolalex September 30, 2013 at 8:52 AM

People, aren't you a morons? Just look at this with a fresh look! Because Java has no just functions, you are to create class, than static method, which creates dynamic method, which actually does what you want.
A good example of how clumsy OOP can be.
Dammit, I want to unsee this.

Reply

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)