The Craftsman: 35
Dosage Tracking XII
What a day.

Robert C. Martin
22 February 2005

...Continued from last month.

Surprise, surprise!

In the early Summer of 1942, while britons were hopelessly battling Hitler's armies in the south and east of England, and the United States was frantically routing out the soviet spies from its upper echelons, the Empire of Japan struck at Pearl Harbor with a massive surprise attack.

Admiral Yamamoto had wanted to strike six months earlier, but the negotiations with Hitler and Stalin had caused delays. Even so, the U.S. was in such political turmoil that he felt confident he was about to eliminate the American threat and then run wild in the Pacific.

But that's not the way things happened. Not at all.

At 0740 on June 6th, Saburo Sakei was flying his zero south towards Oahu in a group of two dozen fighters and torpedo bombers. He kept a constant vigil over the sky and sea, looking left and right, up and down, hunting for planes and ships that could threaten him. When he was just 8 miles from Pearl he saw one of the zeros in his group suddenly explode without warning. He instinctively yanked on his stick to break out of formation, and a half second later saw another companion blow up.

Later, he told his captors that it was like being in a pop-corn popper. Plane after plane exploded before the pilots could even begin to take evasive action. Those few that managed to break out of formation fared little better. Sakei could just see the white vapor trails ripping northwards into the flight of zeros and following (!) *those who managed to break formation.*

Of course he didn't see the one that was following him.

---

21 Feb 2002, 1630

"Yeah, but we've really got to get rid of that `Utilities` class." replied Avery. "Let's see if we can do that before Jerry tells us to quit for the day."

"That sounds like a plan." I replied.

"Indeed!"

I pulled up the `Utilities` class on the screen.

```
public class Utilities {
  public static Date testDate = null;
  public static SuitGateway suitGateway = new InMemorySuitGateway();
  public static MockManufacturing manufacturing = new MockManufacturing();

  public static Date getDate() {
    return testDate != null ? testDate : new Date();
  }

  public static void acceptMessageFromManufacturing(Object message) {
    SuitRegistrationAccepted suitAck = (SuitRegistrationAccepted) message;
    Suit acceptedSuit = new Suit(suitAck.argument, getDate());
    suitGateway.add(acceptedSuit);
  }

  public static Suit[] getSuitsInInventory() {
    return suitGateway.getArrayOfSuits();
  }
}
```

We both stared at it for about 30 seconds. I said: "I suggest we get rid of that last method, `getSuitsInInventory`, I'd wager that it's quite easy to obliterate."

"I think I might agree with you." Avery invoked the 'find usages' command. The result showed that it was only used in one place: a fixture class named `SuitsInInventory`. He brought it up on the screen.

```
public class SuitsInInventory extends RowFixture {
  public Object[] query() throws Exception {
    return Utilities.getSuitsInInventory();
  }
  ...
}
```

Avery studied the screen for a few seconds and then said: "I think obliteration is imminent." He clicked on the `getSuitsInInventory` function and invoked the 'inline' operation. The call to the function was automatically replaced with its implementation...

```
public class SuitsInInventory extends RowFixture {
  public Object[] query() throws Exception {
    return Utilities.suitGateway.getArrayOfSuits();
  }

  public Class getTargetClass() {
    return Suit.class;
  }
}
```

...and the function was automatically removed from `Utilities`.

We looked at each other with evil grins and we both said: "Obliterated!"

All the unit tests still passed, and the FitNesse page still passed. Nothing was broken.

"On to the next function." I said.

We both stared at the acceptMessageFromManufacturing

"Hmmm." Said Avery. "I presume you agree that obliteration is not indicated in this case."

"Indeed." I said. "I think relocation is a more appropriate option."

"Quite, quite. But where?"

This was a good question. This function actually had a bit of policy logic in it. This was the function that accepted the acknowledgement of transfer from manufacturing. This was the function that stored the suit in our inventory.

"What do you call something that finalizes a registration?" I asked.

"A secretary? A clerk? Uh, a ... Registraar!"

We both looked at each other with a grin and repeated: "A Registraar!"

I grabbed the keyboard and clicked on the method. I invoked the 'move' operation and typed `dtrack.policy.Registraar`. After confirming that I did indeed want to create the `policy` package and the `Registraar` class, the method was automatically moved.

```
public class Registraar {
  public static void acceptMessageFromManufacturing(Object message) {
    SuitRegistrationAccepted suitAck = (SuitRegistrationAccepted) message;
    Suit acceptedSuit = new Suit(suitAck.argument, Utilities.getDate());
    Utilities.suitGateway.add(acceptedSuit);
  }
}
```

"Relocated!" we both shouted.

Now the `Utilities` class was looking very anemic.

```
public class Utilities {
  public static Date testDate = null;
  public static SuitGateway suitGateway = new InMemorySuitGateway();
  public static MockManufacturing manufacturing = new MockManufacturing();

  public static Date getDate() {
    return testDate != null ? testDate : new Date();
  }
}
```

"Death is coming to this class." I said with a sneer.

"Without a doubt. And I think I see our next objective. What would you say to removing that `suiteGateway` variable?"

"I think I'd enjoy it very much." I replied.

Avery grabbed the keyboard, clicked on the doomed variable, and invoked the 'move' operation. He selected the `SuiteGateway` class as the target. And the variable was automatically moved.

```
public interface SuitGateway {
  SuitGateway suitGateway = new InMemorySuitGateway();

  public void add(Suit suit);
  int getNumberOfSuits();
  Suit[] getArrayOfSuits();
}
```

"Interesting." I said. "The initialization moved too. I'm not at all certain that I like that. It seems to me that `SuitGateway` should not know about its derivatives."

"I'm afraid I must agree with you, Alphonse. The initialization needs to go somewhere else. I wonder where?"

This was taking some thought, and we were beginning to slip out of our formal banter.

"We need some kind of initialization functions somewhere... But who would call it?"

"Yeah, I don't know. Let's put a *todo* next to this one and ask Jerry about it when he comes back. I don't want this problem to derail us from obliterating the `Utilities` class."

"Agreed! Obliteration is our objective!"

And Avery annotated the initialization with a *todo* comment.

```
public interface SuitGateway {
  // todo move initialization somewhere else.
  SuitGateway suitGateway = new InMemorySuitGateway();

  public void add(Suit suit);
  int getNumberOfSuits();
  Suit[] getArrayOfSuits();
}
```

"OK, now let's find out who uses that variable." I said. I grabbed the keyboard and did a 'find usages' on the variable. There were 6 usages. They all looked about like this:

```
SuitGateway.suitGateway.getNumberOfSuits()
```

"I'd say that's somewhat redundant." I bantered.

"I'd agree. I propose we change the name of the variable." I clicked on the variable and invoked the 'rename' command, changing the variable's name to `instance`. Now all the usages looked about like this:

```
SuitGateway.instance.getNumberOfSuits()
```

Avery looked at the screen for a moment and said: "That's certainly an improvement. But I think we can do better." Avery used the 'rename' function to change the name of `SuitGateway` to `ISuitGateway`. Next he moved the `instance` variable to a new class named `SuitGateway`. Next he found all the usages of the `instance` variable and altered the first by eliminating the variable from the expression as shown below.

```
SuitGateway.getNumberOfSuits()
```

This caused the statement to turn red, showing that it would not compile. Avery clicked on the red lightbulb next to that line of code and selected the 'create method `getNumberOfSuits`' option. This automatically created the method in the `SuiteGateway` class. Finally, Avery modified that method to delegate through the `instance` variable. All the tests still passed.

Avery changed all the other usages of the `instance` variable in similar fashion. Now `SuitGateway` looked like this:

```
public class SuitGateway {
  // todo move initialization somewhere else.
  public static final ISuitGateway instance = new InMemorySuitGateway();

  public static int getNumberOfSuits() {
    return instance.getNumberOfSuits();
  }

  public static Object[] getArrayOfSuits() {
    return instance.getArrayOfSuits();
  }

  public static void add(Suit suit) {
    instance.add(suit);
  }
}
```

"Avery, you are a genius!" I said. I was duly impressed.

"I know." He said with a smirk. And we both laughed.

"This is really pretty." I went on. The `SuitGateway` is really simple to use, and yet it's also polymorphic. We can put any derivative if `ISuitGateway` into that `instance` variable that we want."

"Yes, but we still have to figure out how to initialize it."

"Yeah. So what does our Utilities class look like now?"

I grabbed the keyboard and brought it up on the screen.

```
public class Utilities {
  public static Date testDate = null;
  public static MockManufacturing manufacturing = new MockManufacturing();

  public static Date getDate() {
    return testDate != null ? testDate : new Date();
  }
}
```

“Wow!” I said. “Nothing left but the date stuff, and that odd `MockManufacturing` variable.”

“Yeah, and I’ll bet you that we can get rid of that variable the same way we got rid of the `SuitGateway` variable.”

“Yeah! A `Manufacturing` class, holding a reference to an `IManufacturing` object. Neat!”

Just then, Jerry stepped up. “Hay guys, it’s time for dinner.”

“Huh!” Avery and I looked at each other in surprise. Where had the time gone?

Jerry stared at the `Utilities` class. “Gee, it looks like you guys have been busy cleaning things up, eh? Good. Tell me all about it at dinner. Let’s go.”

So we left the lab for the day...and what a day it had been!

To be continued...

The source code for this article can be found at:
`www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_35_DosageTrackingSystem.zip`