The Craftsman: 43 Dosage Tracking XX Language Lawyers

Robert C. Martin
27 October 2005

...Continued from last month.

May, 1944. Henry Wallace had a brass paperweight in the form of a globe of the Earth. He kept it on his desk in the Oval Office. Embossed on the bottom was: "29 Apr, 1959, 0943 GMT". "Fifteen years." He mumbled to himself. "Fifteen years." He wondered how he could save a world that was so hell bent on destroying itself. He reckoned that he had two bad options: collaborate or conquer. The first seemed infeasible given the belligerence of the axis leaders; they yearned to conquer the Americas, not collaborate with them. The second option was feasible, but distasteful. He could, in fact, conquer the world, and he could do it quickly. But could he keep it dominated for the necessary fifteen years? Not likely.

One thing was certain. Option two would not last long. Having seen that nuclear explosions were possible; Hitler, Stalin, and Tojo had to be pushing their remaining scientists to the breaking point.

---

22 Feb 2002, 1100

Just as we were about to start on the next test case, Jasper walked over with that big depressing grin plastered all over his face.

"Hey there Jerry! Can I borrow you for a second or three?"

Jerry looked up at Jasper without a flinch and said: "Sure, Jasper. Alphonse, I'll be back in a few minutes." Jerry got up and walked away with Jasper as if nothing were wrong; but then he quickly turned around, rolled his eyes, and winked at me.

I smiled back at both of them and said: "OK see you soon. I'll go see what Avery is up to."

I sauntered over to where Avery was sitting and said: "Hi."

Avery looked up and gave me a desperate grimace. "Why did you leave me with that moron?"

"He's not a moron. Actually he's pretty smart. But he can be trying."

"Trying? If he smiles at me one more time I'm going to shove this keyboard into his mouth. And it'll fit too."

We both laughed, making sure no one could overhear us. Then Avery said: "Hey, have you looked at the new features in Java 5"

"Yeah, you mean autoboxing, scanners, generics, and stuff like that?"

"Yeah, have you used any of it yet?"

"Not so far, although we could have used generics yesterday when we wrote a mock database. I just didn't think of it then."

Avery's eye's sparkled. "Oh yeah? Show me!"

This sounded like fun. So I sat down next to him and brought up the `InMemorySuitDatabase`.

```
public class InMemorySuitGateway implements ISuitGateway {
  private Map suits = new HashMap();
  public void add(Suit suit) {
    suits.put(new Integer(suit.barCode()), suit);
  }

  public int getNumberOfSuits() {
    return suits.size();
  }

  public Suit[] getArrayOfSuits() {
    return (Suit[]) suits.values().toArray(new Suit[0]);
  }

  public boolean isSuitRegistered(int barcode) {
    return suits.containsKey(barcode);
  }
}
```

We both looked at this for a second and then Avery said: "Yeah, we can change that `HashMap` real easily." He grabbed the keyboard and made the following change:

```
public class InMemorySuitGateway implements ISuitGateway {
  private Map<Integer, Suit> suits = new HashMap<Integer,Suit>();
  public void add(Suit suit) {
    suits.put(new Integer(suit.barCode()), suit);
  }
  ...
}
```

The tests all still ran.

"Cool!" I said. Now we should be able to use autoboxing in the `add` method! So I grabbed the keyboard and did this:

```
public class InMemorySuitGateway implements ISuitGateway {
  private Map<Integer, Suit> suits = new HashMap<Integer,Suit>();

  public void add(Suit suit) {
    suits.put(suit.barCode(), suit);
  }
  ...
}
```

The tests all still ran!

"Way cool! That's much cleaner than before!" I said.

"Yeah! And now we can get rid of that ugly cast in `getArrayOfSuits`." Avery grabbed the keyboard and deleted the cast.

```
  public Suit[] getArrayOfSuits() {
    return suits.values().toArray(new Suit[0]);
  }
```

The tests ran just fine.

"Sweet!" I said. Too bad we don't have anything else to clean up with generics. This is fun!

"Well..." said Avery. "Have you considered whether or not there will be more than one kind of `Suit`?"

"What? You mean like different derivatives of `Suit`?"

"Yeah. Imagine that you have two derivaties like `MensSuit` and `WomensSuit`.

"You mean like this?" And I grabbed the keyboard and typed the classes in a junk package.

```
public class WomensSuit extends Suit {
  public WomensSuit(int barCode, Date nextInspectionDate) {
    super(barCode, nextInspectionDate);
  }
}

public class MensSuit extends Suit {
  public MensSuit(int barCode, Date nextInspectionDate) {
    super(barCode, nextInspectionDate);
  }
}
```

Avery watched as I typed, and then said: "Yeah. Now create a class that has two lists. One a list of `MensSuit` and the other a list of `WomensSuit`."

"OK." I said, and I typed the following:

```
public class SuitInventory {
  private List<MensSuit> mensSuits = new ArrayList<MensSuit>();
  private List<WomensSuit> womensSuits = new ArrayList<WomensSuit>();
}
```

Avery nodded and said: 'Great! Now let's assume we want to inspect suits. We can use a method named `inspect` that takes a list of suits."

"You mean like this?" And I typed the following:

```
  void inspect(List<Suit> suits) {
      for (Suit s : suits) {
        // inspect s.
      }
  }
```

"Impressive, Alphonse! You made fine use of the new iteration syntax."

I had missed the old formal banter. Sometimes working with Avery was a lot of fun. Especially after working with Jasper. "Indeed I did, Avery, you are quite observant!"

"No more observant than you are creative; but shall we continue?"

"Indeed, let's."

"Well then, suppose that we now decide to create a method for inspecting only `WomensSuit` objects?"

"I imagine I can write such a method. Shall I?"

"Please do."

And so I typed the following:

```
  void inspectWomensSuits() {
    inspect(womensSuits);
  }
```

"Hay!" I said, forgetting teh formal banter for the moment. "That doesn't compile!"

"As, indeed, it should not!" Quipped Avery. He maintained his formal composure.

"But I don't understand, it should compile!"

"No, Alphonse, if you look carefully you'll notice that we are attempting to pass a `List<WomensSuit>` to an argument defined to take a `List<Suit>`."

"Sure, but a `List<WomensSuit>` *is* a `List<Suit>`!"

"Would you like to reconsider that statement Alphonse?"

"Uh..." I thought for a moment and realized that that wasn't quite right. "OK, technically you're right, a `List<WomensSuit>` does not derive from `List<Suit>`, but a list of womens suits is still a list of suits."

"Not a all! You can add an instance of `MensSuit` to a `List<Suit>`, but you can't add a `MensSuit` to a `List<WomensSuit>`."

"OK, that's true, but so what?"

"So, my dear Alphonse, a `List<WomensSuit>` is not substitutable for a `List<Suit>`, and is therefore not a subtype of `List<Suit>`. You are of course aware of the Liskov Substitution Principle[1], are you not?"

That's when it clicked. "Of course. Of course! Substitutability is the basis of subtying. Therefore a list of derivatives is not a derivative of the list of bases."

"Precisely Alphonse!"

"Yes. Precisely! But that leaves us with a problem, doesn't it my dear Avery?"

"Indeed it does! If a list of derivatives is not subtitutable for a list of superclasses, what is?"

"And is there an answer to this connundrum?" I asked?

"Indeed there is. Shall I demonstrate?"

"Please do!"

So Avery took the keyboard and made the following change:

```
void inspect(List<? extends Suit> suits) {
    for (Suit s : suits) {
      // inspect s.
    }
}
```

I looked at the strange syntax that Avery had written. "Now that is one strange syntax; but I think I understand the intent. The question mark indicates that the type is unknown, and the remainder of the clause tells the compiler that the unknown type is a derivative of `Suit`."

"Indeed, indeed. That is the correct interpretation Alphonse. Notice also that the whole program now compiles."

"Yes, I see that. This is all fascinating! There is great power and expressiveness in these new features."

"Yeah, and they're fun too!" Avery said, switching modes.

I gave him a poke in the ribs, and he bopped me on the head with a one of the books on the table.

"Now, now boys, don't go breaking anything." It was Jean. She must have come by to see what we were up to. "Goodness! You boys certainly have a lot of energy." She looked at the screen and exclaimed: "Oh! I see you boys are having fun with the rather odd new syntax for generics. For my money it's all a bit involved, isn't it. So much syntax for so little effect. I fear the language lawyers may be leading us into a deep and unecessary complexities. Such a shame too. But then I suppose youngters like you revel in such intricacies, don't you. Well then, carry on, carry on. You are such fine young men."

Jean waddled off, while Avery and I suppressed giggles.

The code for this article can be located at:

---

[1] See http://www.objectmentor.com/resources/articles/lsp.pdf

http://www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_43_DosageTrackingSystem.zip

To be continued...