The Craftsman: 47
Brown Bag IV
Jinx!

Robert C. Martin
7 February 2006

...Continued from last month.

Sept, 1944.

President Henry Wallace sat in private and buried his face in his hands. "Perhaps we deserve to be destroyed." he said to an uncaring universe. The extermination of the human race seemed assured on several fronts. Clyde was coming in fifteen years, and there didn't seem to be anything to be done about it. The Third Reich was systematically exterminating whole races of people. Stalin continued his purges against any hint of contrary ideologies. The American monopoly on atomic weapons could not last much longer -- and then there would be war and destruction on a scale that only Clyde could rival. Indeed, perhaps Clyde was simply God's undertaker; sent to bury an Earth that was already dead.

And then there was a knock at the door. "Mr. President? Dr. Von Braun is here to see you."

Thursday, 28 Feb 2002, 1100

Avery and I arrived together at the small conference room that we used for our daily discussions. I assumed that the scowl on his face had something to do with the fact that Jerry and Jasmine were following right behind. We all got seated and then Jerry said: "OK, whose turn is it today?" Avery's scowl deepened, but he remained silent.

"Usually Avery and I take turns." I said, "But Avery didn't get his turn yesterday, so..."

Avery sent a cold glance at Jerry, but then said: "Never mind Alphonse, it's your turn. Go ahead. I don't have anything prepared."

"OK, actually I don't have much prepared either, but I do have a question. Has anyone seen the new `enum` feature in Java 5? Apparently you can define `enum` constants that have variables and methods."

Avery suddenly became very alert. "Can you write an example on the wall, Alphonse?"

"Sure", and so I used my finger to write the following code on the wall panel.

```
public class Taco {
  enum Color {red, green, burntumber}
}
```

"OK", I said, "we all know that Java 1.5 has `enum`s that look like this. They are essentially integer constants."

Avery jumped to the wall and started writing and talking at the same time. "Yeah, before 1.5 we

would have done this."

```
public class Taco {
  public static final int RED = 0;
  public static final int GREEN = 1;
  public static final int BURNT_UMBER = 2;
}
```

The wall panel recognized Avery's writing and made sure his code was in a separate sandbox from mine, so that they didn't conflict.

"Right." I said. But that's where the similarity ends. In Java 1.5 an enumeration is actually a class, and each of the enumerators is a derived class. These classes can have variables. For example:" And I modified the code on the screen with my finger.

```
public class Taco {
  enum Color {red, green, burntumber}

  enum Salsa {
    mild(10, Color.red),
    medium(1000, Color.green),
    hot(100000, Color.burntumber);

    public final int scovilles;
    public final Color color;

    Salsa(int scovilles, Color color) {
      this.scovilles = scovilles;
      this.color = color;
    }
  }
}
```

"Hot damn!" said Jasmine. "That's pretty cool. Look, Jerry, the `enum` has a constructor and instance variables!"

Jerry looked at the wall for a second and said, "Yeah, that's pretty useful. This means that a constant can have more than one value. So consider this representation for a Red/Green/Blue color:" Jerry started writing on the wall too.

```
public enum RGB {
  red(0xff, 0x00, 0x00),
  green(0x00, 0xff, 0x00),
  blue(0x00, 0x00, 0xff),
  white(0xff,0xff,0xff);

  public final int r;
  public final int g;
  public final int b;

  RGB(int r, int g, int b) {
    this.r = r;
    this.g = g;
    this.b = b;
  }
}
```

I muscled my way back up to the wall and said: “Right, and this allows you to do interesting things like this.”

```
public class Taco {
  enum Color {red, green, burntumber}

  enum Salsa {
    mild(10, Color.red),
    medium(1000, Color.green),
    hot(100000, Color.burntumber);

    public final int scovilles;
    public final Color color;

    Salsa(int scovilles, Color color) {
      this.scovilles = scovilles;
      this.color = color;
    }
  }

  public static boolean isTooHotForMe(Salsa s) {
    return s.scovilles > 30000;
  }
}
```

“Oh wow!” Said Jasmine. “That means we can write code that makes decisions about enums without knowing any of the enumerations!”

“Huh?” said Jerry. “What do you mean?”

“Well, just look at what Alphonse wrote!” She urged. “The `isTooHotForMe` method will work for any enumerator within the Salsa enumeration. Even if we add new enumerators later, the `isTooHotForMe` method will continue to work unchanged.”

“The Open Closed Principle!” I blurted.

“Yeah!” Said Avery. “We can add, change, or remove enumerations, without affecting the `isTooHotForMe` method!”

Jerry looked intently at the wall for a few seconds, and then said: “Hmmm, that implies a whole new way of thinking about constants and enums. Sometimes we want to NOT know what enumerators there are, and rather depend on the internal structure. For example, we could write an `isBlue` function as follows:

```
class SomeClass {
  public static boolean isBlue(RGB color){
    return color.b > 2*(color.r + color.g);
  }
}
```

Jasmine rolled her eyes and jabbed Jerry in the ribs with her elbow. “You’ve got color on the brain, don’t you buddy-boy.” Then they smiled at each other just a little too long.

I shook my head and said: “That method might be better off in the enumeration itself. Look at this.”

```
public enum RGB {
  red(0xff, 0x00, 0x00),
  green(0x00, 0xff, 0x00),
  blue(0x00, 0x00, 0xff),
  white(0xff,0xff,0xff);

  public final int r;
  public final int g;
```

```
  public final int b;

  RGB(int r, int g, int b) {
    this.r = r;
    this.g = g;
    this.b = b;
  }

    public boolean isBlue(){
    return b > 2*(r + g);
  }
}
```

"Does that work?" Avery demanded.

"Sure, I said, and wrote a test to prove it."

```
public class RGBTest extends TestCase {
  public void testIsBlue() throws Exception {
    assertTrue(RGB.blue.isBlue());
    assertFalse(RGB.red.isBlue());
    assertFalse(RGB.green.isBlue());
  }
}
```

"Wow." Said Jerry.

"Wow." Said Jasmine.

"Jinx!" They both said, giggling.

"So we could do the same thing to the `Taco.Salsa enum` too, right?" , asked Avery. "Let me try." And he started changing things on the wall.

```
  enum Color {
    red, green, burntumber}

  enum Salsa {
    mild(10, Color.red),
    medium(1000, Color.green),
    hot(100000, Color.burntumber);

    public final int scovilles;
    public final Color color;

    Salsa(int scovilles, Color color) {
      this.scovilles = scovilles;
      this.color = color;
    }

    public boolean isTooHotForMe() {
      return scovilles > 30000;
    }
  }
```

"And here are the tests." He continued.

```
public class TacoTest extends TestCase {
  public void testIsTooHot() throws Exception {
    assertTrue(Taco.Salsa.hot.isTooHotForMe());
    assertFalse(Taco.Salsa.medium.isTooHotForMe());
    assertFalse(Taco.Salsa.mild.isTooHotForMe());
  }
```

```
}
```

Avery stood with a silly grin as he stared at the green test indicator bar on the wall. "That's really cool." He finally said.

"Yes." I agreed. "But it's better than that. Those methods can be polymorphic."

"What?" said Jerry.

"What?" said Jasmine.

"Jinx!" (giggle).

Avery and I rolled our eyes at each other.

"Yeah." Look at this.

```
public class Taco {
  enum Color {
    red, green, burntumber}

  enum Salsa {
    mild(10, Color.red) {
      public int sips() {
        return 0;
      }
    },
    medium(1000, Color.green){
      public int sips() {
        return 1;
      }
    },
    hot(100000, Color.burntumber){
      public int sips() {
        return 5;
      }
    };

    public final int scovilles;
    public final Color color;
    public abstract int sips();

    Salsa(int scovilles, Color color) {
      this.scovilles = scovilles;
      this.color = color;
    }

    public boolean isTooHotForMe() {
      return scovilles > 30000;
    }
  }
}
```

"Does that work?!" Avery demanded once again.

Everyone looked at him for a second, and then started to laugh. After a few seconds, even Avery laughed.

"Sure, here are the tests." I said.

```
  public void testSips() throws Exception {
    assertEquals(0,Taco.Salsa.mild.sips());
    assertEquals(1, Taco.Salsa.medium.sips());
    assertEquals(5, Taco.Salsa.hot.sips());
  }
```

What happened next I can only describe in code:

```
wow() * 3;
jinx() * 3;
laugh() * 4;
```

"OK." Jerry said with some finality. "This has gotten out of control. I think it's time we got back to work."

There were nods of agreement, and everyone headed for the door.

"But wait!" I said. "I still haven't asked my question."

"OK, Hotshot." Jasmine responded. "Hurry it up. What's your question."

I stood my ground and looked at the three of them. Then I said: "What would you use this for?"