The Craftsman: 49
Brown Bag VI
Abstract Factory

Robert C. Martin
18 October 2006

...Continued from last month.

October, 1944.

"Dr Ulam's idea is simplicity itself!" Dr. Von Braun said to Dr. Robert Goddard as they ate lunch at the Nimbus facility. "A nuclear powered spaceship can be propelled simply by dropping atomic bombs out the back and blowing them up!"

"I see," said Goddard, "That seems a bit drastic. I presume you'd use a shield, a kind of big pusher-plate to absorb the momentum of the blast and transmit it to the whole ship."

"Precisely, Robert! We use one-kiloton bombs at a rate of one per second or so. A few thousand such bombs could propel the entire Nimbus facility, and all it's inhabitants, to Saturn and back!"

"Yes, I suppose that when you use atomic bombs as a propellant lifting power is not a problem. Are you really thinking about going to Saturn?"

"Oh, no that's just something we could *do if we wanted to. However we* are *going to try to land on Mars by this time next year."*

Goddard was taken aback. He stared at Von Braun in disbelief. After a few seconds he said: "That's quite, er..., aggressive, isn't it?"

Von Braun just looked soberly at Goddard and said nothing. Goddard finally conceded: "Yes, I suppose we are in a bit of a rush." Von Braun just nodded.

"Why Mars?"

"It's a reasonable test of the ship's capabilities, and it might just be our new home."

Goddard nodded. "Yes, canals, vegetation, perhaps even an ancient civilization. I hope they don't resent us just showing up like this."

"There's nobody there, Robert, they all died centuries ago, millennia ago. And if someone is there, we'll just have to deal with that at the time. You can bet we'll be prepared."

"Hmmph." Goddard acknowledged. Then he looked thoughtful for a moment and said: "Have you considered, Werner, that if you can propel a ship with atomic bombs, that you might be able to shift the orbit of Clyde in the same manner?"

Von Braun began to shake his head but then he stopped and stared at Goddard, and his eyes got very large.

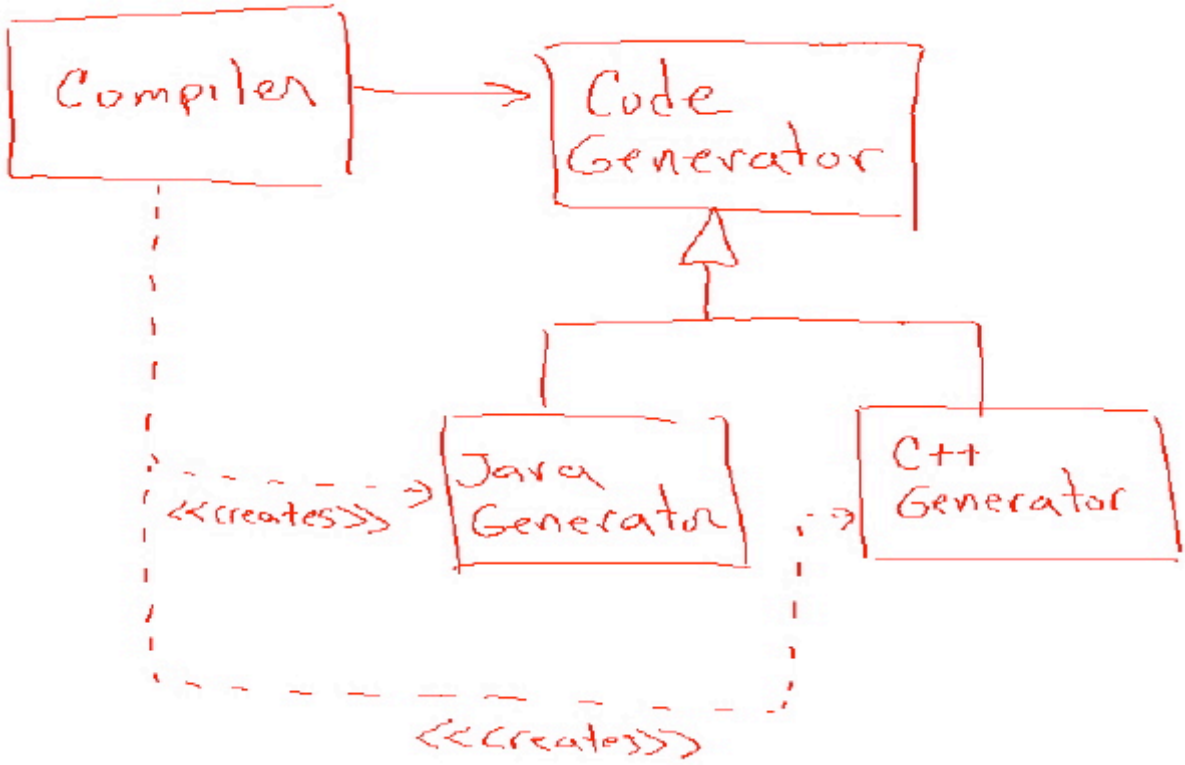
Monday, 4 Mar 2002, 1100

I was determined not to repeat last Friday's late entrance; so I walked into conference room along with everyone else, and noted that Adelaide was getting ready to present. Our little group had grown quite a bit.

There were people here I had never met before. I counted 25 people in the room. Even Jean was there.

Adelaide rapped for attention, and the crowd settled down. She was clearly nervous, but wasn't in a mood to back down. She steeled herself and began.

"OK everybody, Jasmine asked me to talk about the ABSTRACT FACTORY pattern. She and I just used this pattern in the SMCRemote project, and so I guess I can say a few words about it. Here is what I know:" Adelaide began to draw on the wall with her finger.



"Here is the old design of The SMC compiler, except that we have added a feature that allows it to generate both Java and C++ code. As I'll show you in a second, we are proposing a plug-in architecture so that new languages can be added to the compiler simply by creating new derivatives of the `CodeGenerator` class. Unfortunately, as you can see in the current design, in order for the `Compiler` class to create the appropriate instance, it has to have a direct dependency on the derivatives.

"So we propose to use the ABSTRACT FACTORY pattern to break this dependency and insulate the `Compiler` class from any new derivatives of the `CodeGenerator`."

It was clear that Adelaide had rehearsed this pitch a few times. She was a bit too smooth, and going a bit too fast. Jerry must have noticed the same thing because as Adelaide turned around to erase the diagram he stopped her with an obvious question."

"Adelaide, why is it a problem that `Compiler` knows about `JavaGenerator` and `C++Generator`? It seems to me that those dependencies are perfectly appropriate. After all, the `Compiler` class just calls `new` on those classes and then uses them through the `CodeGenerator` interface, doesn't it?"

Adelaide glanced uneasily at Jasmine, but then took a breath and answered Jerry directly.

"We don't want any kind of dependency from `Compiler` to the derivatives of `CodeGenerator` because we don't want to have to recompile `Compiler` any time one of the derivatives changes. The `CodeGenerator` class is there to insulate `Compiler` from such changes. It would be a shame to waste that insulation simply because of the `new` keyword."

I wasn't sure I bought that argument, so I asked:

"Why would changes to the derivatives force a recompile of the `Compiler` class?"

Adelaide looked a little flustered and glanced meaningfully at Jasmine. So Jasmine stood up to rescue her and gave me a stern look.

"OK, Hotshot, we talked about this during the VISITOR discussion last week. So I know you know the answer. But for the benefit of everyone else in the room, our build system recompiles modules based upon the modification dates of files. Since `Compiler` depends on `JavaGenerator`, if `JavaGenerator.java` has a later date than `Compiler.class`, the build system will automatically rebuild `Compiler`. OK Alphonse?"

"Sure, er, thanks."

One of my old classmates, Alex, stood up and said: "OK, sure, but you could fix that by just using the name of the class and `class.forName` in the `Compiler.`"

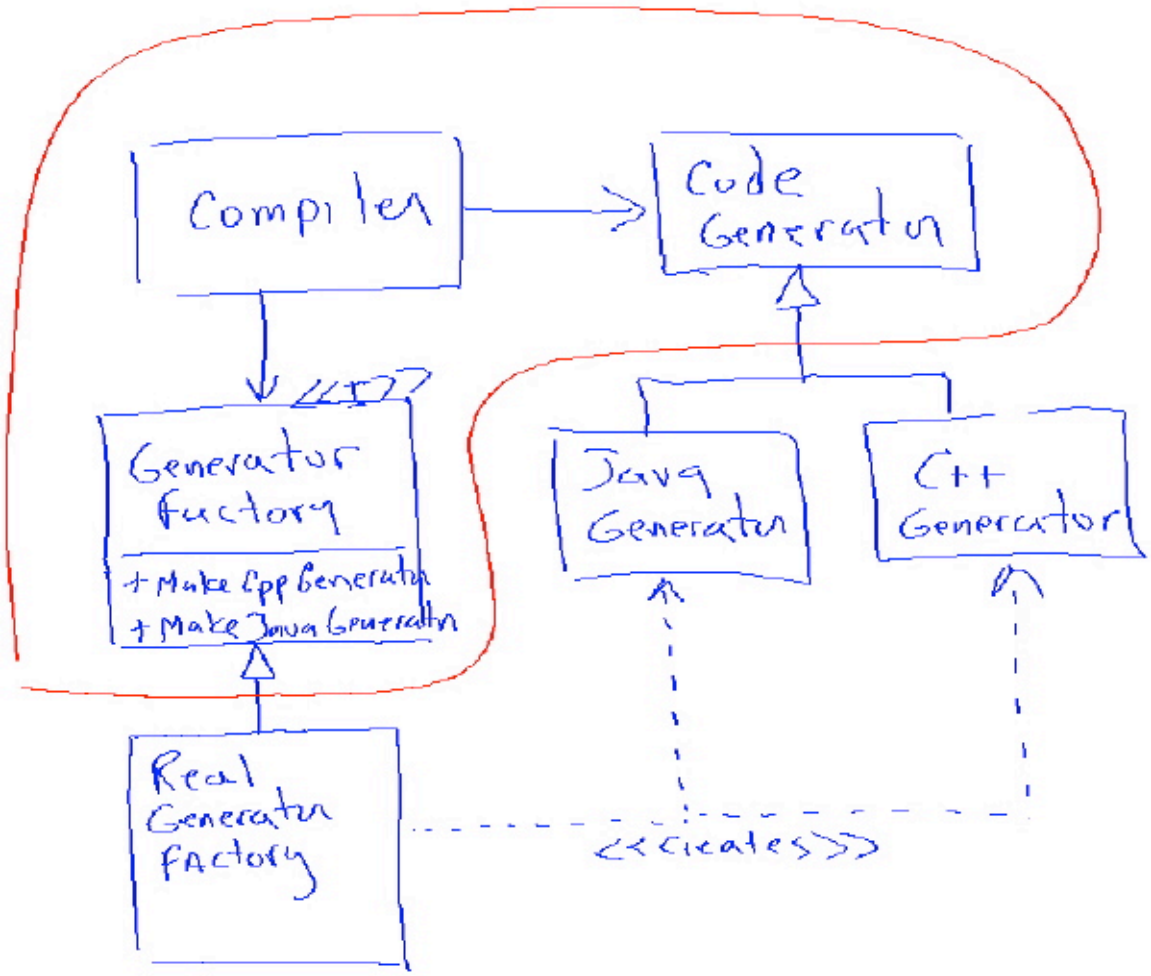
"Yeah, we could." Replied Jasmine. "But we don't want that kind of detailed mechanism inside the `Compiler` class. We are trying to *separate concerns*." Jasmine stretched those last two words out with a sarcasm that forced Alex to sit down again.

Joseph stood up and said: "Jasmine, Alex asked a reasonable question, you don't need to shut him down like that."

Jasmine sighed and her shoulders drooped a bit. Then she said: "OK, you're right Joseph. Alex, I'm sorry I was abrupt. Our goal here is to keep `Compiler` from knowing how the derivatives are created, or, in fact, anything detailed about the derivatives. We want this because we think the mechanism for creating the derivatives may change over time, and we don't want those changes to impact on the `Compiler` class. We want the `Compiler` to just be a compiler, and nothing else."

Jasmine sat down and I caught a glimpse of her rolling here eyes at Jerry.

Adelaide stood back up, waved at the wall to erase the last diagram, and then drew the following:



Then she started her practiced banter again. "This is our proposed solution. It is classic example of the ABSTRACT FACTORY pattern. Notice the `GeneratorFactory` interface. The `Compiler` class calls one of the two `makeXXXGenerator` methods of this interface. In response the `RealGeneratorFactory` creates the appropriate instance and returns it to the `Compiler`. Now notice the red line…"

"I'm sorry dear." Jean interrupted. "But I'm afraid that all this UML is giving me a bit of a headache. I guess I'm just old-fashioned enough to want to see some code. You're a terribly sweet girl, would you mind showing me the code?"

By the look on Adelaide's face, I think it was becoming clear to her that she was never going to get through her canned presentation. To her credit, she didn't panic, or even glance over at Jasmine for more help. Instead, she paused for an instant, and then opened up a TextMate window on the wall. Using the wall's virtual keyboard she typed:

```
public interface GeneratorFactory {
    public CodeGenerator makeCppGenerator();
    public CodeGenerator makeJavaGenerator();
}

public class RealGeneratorFactory implements GeneratorFactory {
    public CodeGenerator makeCppGenerator() {return new CppGenerator();}
    public CodeGenerator makeJavaGenerator() {return new JavaGenerator();}
}
```

"Does that help?" she asked politely.

Jean didn't even take a second to look at the code. She just smiled indulgently and said, "Yes, dear,

that's just fine."

Avery whispered in my ear: "Jean just wanted to be sure that Adelaide knew how to write the code."

I tended to agree, but did not say anything in response.

"OK, now look at the red line." Adelaide continued. "The red line delimits our plug-in architecture. Everything outside the red line is a plug-in. Everything inside the red bubble is our core engine. Notice how all the dependencies that cross the red line points inwards! This means that changes to elements outside the bubble don't affect the elements inside the bubble. For example, if one of the code generator derivatives changes, the bubble does not need to be recompiled."

"A fat lot of good that does!" Avery had stood up and was speaking loudly and forcefully. "This structure has the same problem that the VISITOR has. There's a dependency cycle between the `GeneratorFactory` and the `CodeGenerator` derivatives. Every time you add a new generator derivative you have to add a new `make` method to the `GeneratorFactory` interface, which means your red-lined bubble *does, in fact,* need to be recompiled. So this whole pattern is just worthless!"

Avery stood there glaring while Adelaide looked to be on the verge of tears. Jasmine shot to her feet. "Avery, you little…"

But before Jasmine could make a move, Jean stood up stiffly and said: "Avery dear, would you please come with me." Jasmine's eyes widened and her eyebrows did a quick raise, and then she sat down again. Avery's face went white. Jean beckoned him towards the door, and the two left in uneasy silence.

The uneasy tension remained for a few moments, and then Jasper said: "Gee, wasn't it just last week when these sessions were fun!" The tension broke and everyone laughed and relaxed…a little.

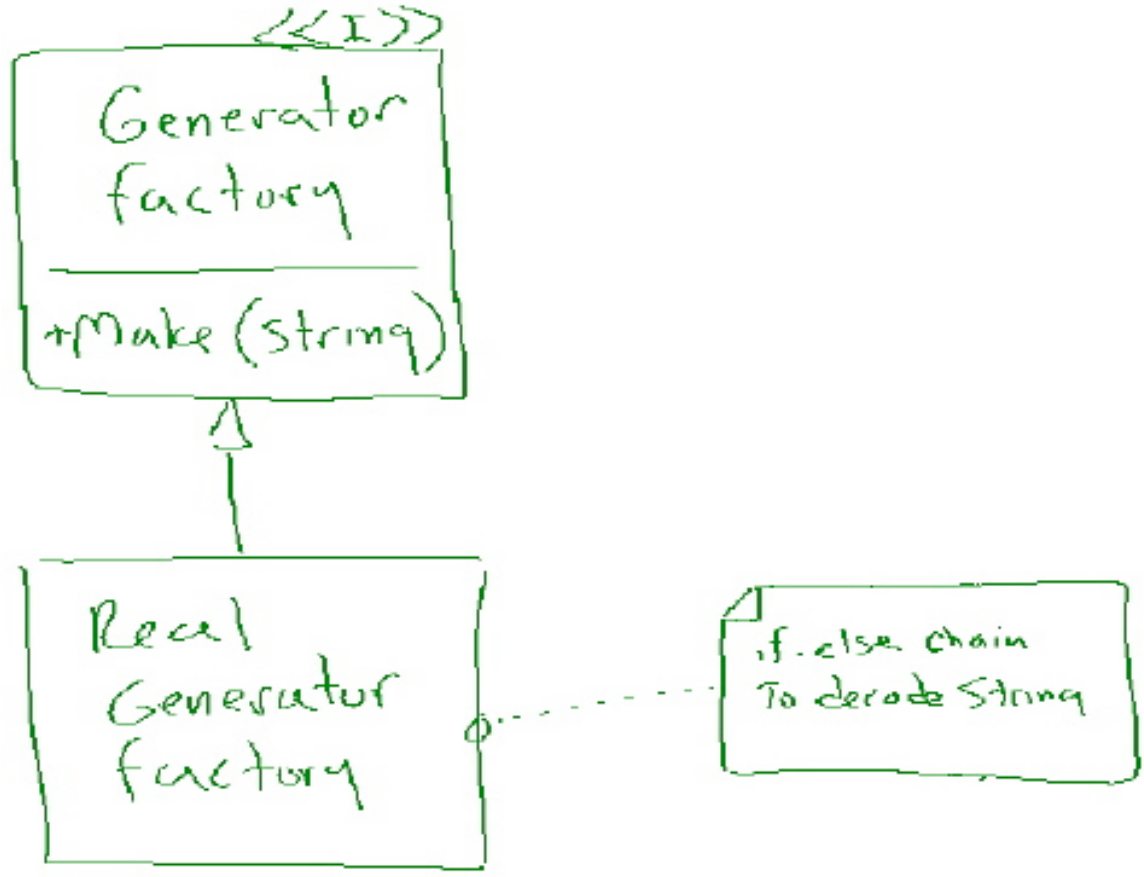
Jerry stood up. "Avery could have said it better, but he had a point, didn't he? You *do* have to change and recompile the core engine every time a new code generator is added."

Adelaide was still staring at the deck, biting back tears, and couldn't seem to answer. Jasmine eventually stood up and said: "Yes, that was an intentional part of the design. We expect that new languages won't be frequent enough to bother protecting ourselves from. What we are trying to insulate the core from are the maintenance changes to the existing code generators."

Jared stood up and said: "OK, that's fair enough, but what if you did want to protect yourself from new languages? What would you do then? How could you keep from recompiling the core engine every time someone wrote a new code generator derivative?"

There was silence in the room for a few seconds, and then I stood up.

"I've been thinking about this for the last few days because we have a similar problem in the Dtrack project. We don't want to recompile our core engine every time a new kind of spacesuit is created." I waved at the wall to erase it, and then drew the following:



"And just in case Jean come back in I'll write the code."

```
public interface GeneratorFactory {
    public CodeGenerator makeGenerator(String type);
}

public class RealGeneratorFactory implements GeneratorFactory {
    public CodeGenerator makeGenerator(String type) {
              if (type.equals("C++")) return new CppGenerator();
              if (type.equals("Java")) return new JavaGenerator();
              return null;
    }
}
```

"Now imagine that the strings `"C++"` and `"Java"` are command line arguments to the `Compiler` class. As you can see, we can add new code generators without ever touching, recompiling, or in any other way impacting the core engine."

Jasmine said: "But, Alphonse, that's not type safe! What if you misspell the string?"

"Wouldn't your unit tests catch that misspelling?"

"Uh…"

Jerry said: "I think Jasmine's point is that we'd like the compiler to check this for us."

"Why?" I replied? "Again, won't you have unit tests that make sure you aren't misspelling?"

"OK, but this is a command line argument!"

"True, but if you misspell a command line argument shouldn't you expect to get an error? In this case

the `Compiler` engine could say: `'Ruby is not a supported language.'`"

"OK, fine, but what if you call the `Compiler` from a script or another program, and there was a spelling error in one of those programs. The compiler wouldn't warn you."

"True, but again, your unit tests would."

"Er…"

This was fun. I seem to have stumbled onto an idea that neither Jerry nor Jasmine had thought about. Indeed, all of the journeymen in the room had intent looks on their faces.

"Why is this such an issue?" I asked.

Jasper spoke up. "Well, Fonze, er… Alphonse. I don't know about the others, but you've made me wonder whether or not type safety is as valuable as we had been thinking. This unit testing stuff we've been doing over the last couple of years may make type safety redundant."

I saw a few heads nod, and several others shake. But as we left the room the mood remained thoughtful.