Typestate Pattern in Rust

With a Strict Builder Example

David Wallace Croft, M.Sc.

Presented to the

Dallas Rust User Meetup

2024 Aug 13 Tue

Typestate Pattern

- Problem
  - Permit state transitions only when valid
  - Enforce using static compile-time checks
- Solution
  - Represent the states using enum variants
  - State transition methods are variant-specific
  - State transition methods consume old self

Example Code

- Open source example code on GitHub
- https://github.com/david-wallace-croft/pattern-typestate/tree/main/src/typestate_2

Design Patterns

- Borrowed from a book on building architecture
- Adopted by software architects
  - Gamma, et al., "Design Patterns", 1994
  - Known as the "Gang of Four Book"
- A reusable pattern that fits a type of situation
  - Problem and solution
  - Customized as needed
- Quickly communicate design ideas
  - Using just the name of the pattern

Antipatterns

- Sub-optimal design patterns
  - Used frequently enough to be named
- Should generally be avoided
  - Disadvantages outweigh the advantages

Builder Pattern

- Problem
  - Make a recipe to assemble a complex object
  - Enable swapping out implementations
- Solution
  - Client instantiates a ConcreteBuilder
  - Client passes ConcreteBuilder to Director
  - Director operates on an AbstractBuilder
  - Client retrieves Product from ConcreteBuilder

Named Arguments

- A potential run-time error
  - my_function(height, width)
  - my_function(width, height)
- Named arguments (a.k.a. named parameters)
  - my_function(width => width, height => height)
- Languages with named arguments
  - Ada, C#, Fortran, Kotlin, Python, Ruby, [...]
- Rust
  - let rectangle = Rectangle { width, height };

Pseudo-Builder Antipattern

- Work-around for a lack of named arguments
  - let p = Product::builder().a(a).b(b).build();
- Frequently misidentified as the Builder Pattern
  - I call it the Pseudo-Builder
- Easily misused resulting in a run-time error
  - Building before all required arguments given
  - Reusing after build when not designed for it
  - Permits invalid argument combinations
  - Breaks when arguments added to constructor

Strict Builder Pattern

- A Pseudo-Builder that cannot be misused
  - Based on the TypeState Pattern
- Compile-time errors instead of run-time errors
  - Cannot build until all required values provided
  - Prevents invalid argument combinations
  - New arguments require code updates
  - Initial arguments determine next ones allowed
  - Prevents reuse after build

Example Code

- Open source example code on GitHub
- https://github.com/david-wallace-croft/pattern-type state/tree/main/src/strict_builder_0

Links

- Cliff L. Biffle, "The Typestate Pattern in Rust", 2019-06-05, https://cliffle.com/blog/rust-typestate/
- Eric Smith, "Game Development with Rust and WebAssembly", 2022 Apr, https://www.packtpub.com/en-us/product/game-development-with-rust-and-webassembly-9781801070973/
- Gamma, et al., "Design Patterns: Elements of Reusable Object-Oriented Software" (1E), Addison-Wesley Professional, 1994. https://en.wikipedia.org/wiki/Design_Patterns

Licenses

- Slides and code are © 2024 CroftSoft Inc
- This slide presentation is available under the terms of the Creative Commons Attribution 4.0 International License https://creativecommons.org/licenses/by/4.0/
- The code is available under the terms of the open source MIT License https://opensource.org/license/mit/