Systemnahe Programmierung in Rust

- “The Book” / Objektorientierung / Kap. 17 -

Hubert Högl

Hochschule Augsburg / Informatik

2022-12-13 12:25:43

Charakteristiken objektorientierter Sprachen (17.1)

OO-Features

- Objekte
- Kapselung
- Vererbung

Objektorientierte Programme setzen sich aus Objekten zusammen. Ein Objekt verpackt sowohl Daten als auch Prozeduren, die auf diesen Daten operieren. Die Prozeduren werden normalerweise Methoden oder Operationen genannt.

Danach ist Rust auch objektorientiert:

- Strukturen und Aufzählungen haben Daten
- `impl` Blöcke enthalten Methoden
- Kapselung von Implementierungsdetails über `pub` / nicht `pub`

Beispiel: `pub struct AveragedCollection()`

Code in `material/Beispiele/avgcoll/`

https://rust-lang-de.github.io/rustbook-de/ch17-01-what-is-oo.html

Vererbung

Rust hat keinen allgemeinen Vererbungs-Mechanismus und keine Klassen.

Zwei Möglichkeiten der Vererbung bei OO-Sprachen:

a. Beziehung von Typen (“ist ein”)

   Polymorphie - mit Daten unterschiedlichen Typs arbeiten, als wären sie vom gleichen Typ. In OO-Sprachen wird das durch Vererbung ausgedrückt.

   In Rust ist das Programmierung mit generischen Typen und Merkmalsabgrenzungen (*trait bounds*), um verschiedene Typen miteinander in Beziehung zu bringen (Methodenauswahl zur Kompilierzeit).

   Ausserdem gibt es **Trait Objects** (Methodenauswahl zur Laufzeit).

   Fachbegriff: *bounded parametric polymorphism*

b. Code-Wiederverwendung

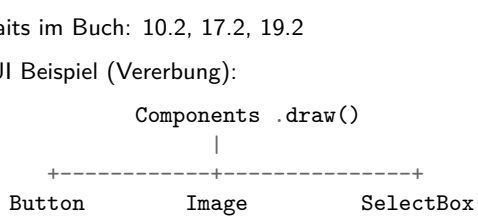
   In Rust geht das durch Default Implementation von Trait Methoden. Diese Standard-Methoden können auch in `impl` Blöcken überschrieben werden.

Die Vererbung in OO-Sprachen wird mittlerweile auch kritisch gesehen.

Trait Objects (17.2)

Traits im Buch: 10.2, 17.2, 19.2

GUI Beispiel (Vererbung):



In Rust: `trait Draw` mit Methode `draw()`

Trait Objekt (`dyn Draw`)

- Muss immer hinter Zeiger liegen, da "unsized": `&dyn Draw` oder `Box<dyn Draw>`.
- Kombiniert Daten und Verhalten
- Verweist auf
  - Instanz des Typs, der Merkmal implementiert
  - Tabelle mit Zeigern, in der die Trait Methoden dieses Typs stehen

Vektor: `Vec<Box<dyn Draw>>`

Trait Objects

Siehe Beispiel in `material/Beispiele/gui/`.

Überprüfung ob Trait implementiert wird zur Laufzeit.

Dynamische Methodenauswahl (*dynamic dispatch*):

- Zeiger auf Tabelle, die Zeiger auf Trait Methoden enhält
- Laufzeitkosten
- Optimierungen wie z.B. Methode inline zu verwenden klappen nicht

Entwurfsmuster *state pattern* (17.3)

17.3: https://rust-lang-de.github.io/rustbook-de/ch17-03-oo-design-patterns.html

- Wird mit Hilfe von Trait Objects implementiert
- Siehe
  - `material/Beispiele/blog/` (Code)
  - `material/Bilder/state-pattern.jpg` (grafische Veranschaulichung)
- Für uns erst mal nicht so interessant ...

Zusätzliche OO Literatur

- Steve Donovan, A Gentle Introduction To Rust, Kap. 8 (Object-Orientation in Rust)

  https://stevedonovan.github.io/rust-gentle-intro/object-orientation.html

  Code in `material/Beispiele/gentle_oo.rs`

- Weiterer Lesestoff zum Thema "OO" aus dem empfehlenswerten Buch von Carlo Milanesi, Beginning Rust, 2. Auflage 2022 (frei von Springer-Link <https://link-springer-com.ezproxy.hs-augsburg.de herunterladbar>)

  Kapitel 20: Object-oriented Programming

  https://hhoegl.informatik.hs-augsburg.de/sysprog/Rust/Milanesi-2e-Kap20-OO.pdf