FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Konzepte moderner Programmiersprachen (KmPS)

(Wahlfach, Modulnummer 55685)

Wintersemester 2025/2026

Prof. Dr. Andreas Claßen
(zusammen mit Prof. Dr. Heinrich Faßbender)
Fachbereich 5 Elektrotechnik und Informationstechnik
FH Aachen

"Objekt-Orientierung" in Rust

Strukturierte Datentypen

… mittels **`struct`**, ähnlich wie in C / C++ und Go.

Beispiel:

```
struct Dummy {
  s: String
}
```

Methoden für `struct` (und `enum`)

… mittels **`impl`**, *ähnlich wie in Go (dort Methoden für* Receiver Type*).*
Parameter `&self` oder `&mut self`.

Methode wird also "nachträglich" zum strukturierten Typ hinzugefügt.
Ähnlich wie in Go …

Beispiel:

```
struct Person { … }

impl Person {
    fn full_name(&self) -> String { format!("{} {}", self.first_name, self.last_name) }
}

let p = Person::new(…);
println!("fullname {}", p.full_name());
```

Traits

… spezifizieren einen zusammengehörigen Satz von Methoden.
Analog z.B. zu den Interfaces in Go.

Idee: Traits definieren bestimmtes (Teil-) Verhalten von Daten.
Datentypen können dann solche Traits implementieren.
Z.B. `Copy` Trait: Die Daten von Datentypen, die diesen Trait implementieren, können kopiert werden.

Traits definieren meist nur die Signatur (den Prototyp) der Methoden.

Beispiel:

```
trait MyPrint { fn my_print(&self) -> String; }
```

`impl` für Datentyp bezüglich Traits

… gibt an, wie ein Datentyp einen Trait implementiert.
Analog zu `implements` in anderen Programmiersprachen.
Unterschied zu Go, wo dies nicht programmiert werden muss, sondern automatisch ermittelt wird …

Beispiel:

```
trait MyPrint { fn my_print(&self) -> String; }

impl MyPrint for MyStructType { fn my_print(&self) -> String { … } }

impl MyPrint for i32 { fn my_print(&self) -> String { format!("i32: {}", self) } }
```

Keine Vererbung in Rust (außer bei Traits …)

Statische Typprüfung auf Methoden mittels Traits und mittels generischer Programmierung.

Traits: "Additive Flexibilität" für den Programmierer durch "inkrementelles" Hinzufügen von Methoden.

Aber: Bei den Traits gibt es mit Supertraits/Subtraits dann doch Vererbung …
Zwischen den `struct` *aber keine Vererbung …*

Generische Programmierung als Alternative zu Trait Objects

Rust "kann sich nicht entscheiden" zwischen beiden Ansätzen, unterstützt beide.
Zeigt, dass die Konzepte letztendlich nicht ganz schlüssig / durchgängig sind …