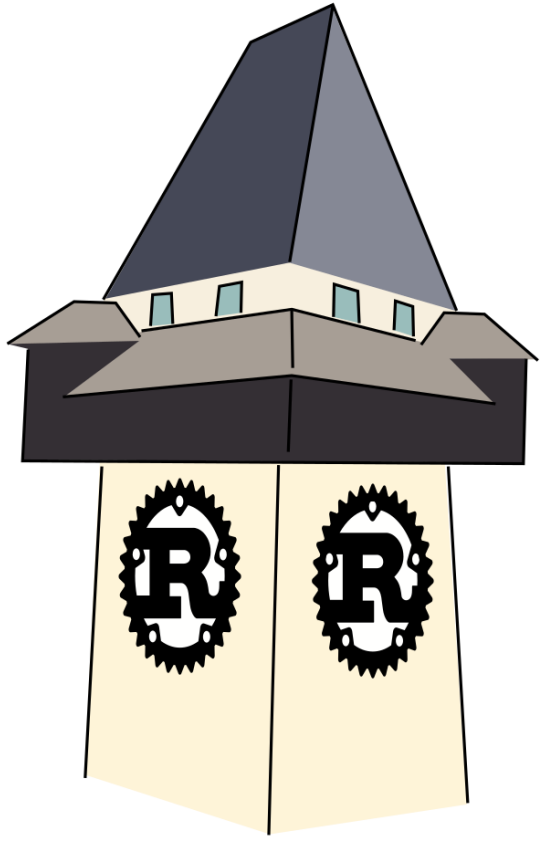
RUST GRAZ – 08 TRAITS

Lukas Prokop

29th of January, 2020



unsafe & traits

29th of January 2020

Rust Graz, lab10

OBJECT-ORIENTED PROGRAMMING

ONE DEFINITION OF OOP

1. Inheritance
2. Polymorphism
3. Data encapsulation

POPULARIZED BY SIMULA (1960S)

via Wikipedia

```
Glyph Class Char (c);
  Character c;
Begin
  Procedure print;
    OutChar(c);
End;
```

DEFINITION BY GANG OF FOUR

Object-oriented programs are made up of objects. An object packages both data and the procedures that operate on that data. The procedures are typically called methods or operations.

IS RUST AN OOP LANGUAGE?

Yes and no. Depends on definition.

(BTW, I would like to stress that OOP is not always class-based OOP. JavaScript and Lua, for example, use prototype-based OOP)

TRAITS

DEFINITION

In computer programming, a trait is a concept used in object-oriented programming, which represents a set of methods that can be used to extend the functionality of a class

Self ⇒ {PHP, rust, Scala, …}

RUST TRAITS

- Inheritance ⇒ no, but default implementations for code reuse
- Polymorphism ⇒ generics and trait bounds
- Data encapsulation ⇒ `pub` keyword

TRAIT EXAMPLE

via rust book

```
pub struct AveragedCollection {
    list: Vec<i32>,
    average: f64,
}
```

TRAIT EXAMPLE

```
impl AveragedCollection {
    pub fn add(&mut self, value: i32) {
        self.list.push(value);
        self.update_average();
    }

    pub fn remove(&mut self) -> Option<i32> {
        let result = self.list.pop();
        match result {
            Some(value) => {
                self.update_average();
                Some(value)
            },
            None => None,
        }
```

TRAIT EXAMPLE

- `AveragedCollection` implements 4 methods
- 3 methods constitute the public API
- We can define an interface, which must be implemented to satisfy a `trait`

TRAIT SYNTAX

```
pub trait Average {
    fn average(&self) -> f64;
}
```

⇒ `AveragedCollection` must implement trait explicitly

TRAIT EXAMPLE

```
impl Average for AveragedCollection {
    fn average(&self) -> f64 {
        self.average
    }
}
```

⇒ `AveragedCollection` implements `Average` explicitly

EPILOGUE

mitsuhiko on Twitter:

> *I'm sorry but that floats/doubles are not Eq in rust is somewhere between hugely frustrating and bullshit. Not sure where. It makes the simplest things really complex for no real added benefit.*

simnalamburt on Twitter:

> *They can't be "Eq" since 'NaN == NaN' should be evaluated as false due to IEEE 754*
>
> *Trust me you'll be much more angry and seriously more frustrated if they are "Eq" and disobey IEEE 754.*

NEXT SESSION

Wed, 2019/02/26 19:00

Topic: Generics

THANKS!