Is Rust Object-oriented Programming?

Mustakimur R. Khandaker

What is OOP?

Object-oriented programming (OOP) is a way of modeling programs.

- Many competing definitions describe what OOP is; some definitions would classify Rust as object oriented, but other definitions would not.

An **object** is an encapsulation of data together with procedures that manipulate the data and functions that return information about the data.

Characteristics of Object-Oriented Languages:

- **Encapsulation:**
  Encapsulation refers to mechanisms that allow each object to have its own data and methods.
- **Polymorphism:**
  Polymorphism refers to the capability of having methods with the same names and parameter types exhibit different behavior depending on the receiver.
- **Inheritance:**
  Inheritance refers to the capability of defining a new class of objects that inherits from a parent class.

The book **Design Patterns: Elements of Reusable Object-Oriented Software** by *Erich Gamma*, *Richard Helm*, *Ralph Johnson*, *and John Vlissides* (Addison-Wesley Professional, 1994) colloquially referred to as **The Gang of Four book**, is a catalog of object-oriented design patterns.

Rust Object?

In a sense, Rust is object oriented.

- structs and enums have data, and impl blocks provide methods on structs and enums.
- Even though structs and enums with methods aren't called objects, they provide the same functionality of objects.

```
let mut avg_obj = AveragedCollection::new();
avg_obj.add(10);
```

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

impl AveragedCollection {
    fn new() -> AveragedCollection {
        AveragedCollection {
        list: Vec::new(),
        average: 0.0,
        }
    }
    fn add(&mut self, value: i32) {
        self.list.push(value);
        self.update_average();
    }
}
```

Encapsulation? 

Another aspect of OOP is the idea of encapsulation.

- It means that the implementation details of an object aren't accessible to code using that object.
- Therefore, the only way to interact with an object is through its public API.

Remember, we can use the pub keyword to decide which modules, types, functions, and methods in our code should be public, and by default everything else is private.

- If encapsulation is a required aspect for a language to be considered object oriented, then Rust meets that requirement.

```
let mut avg_obj =
list_maintainer::AveragedCollection::new();

avg_obj.add(10);
```

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

    impl AveragedCollection {
        pub fn new() -> AveragedCollection {
            AveragedCollection {
                list: Vec::new(),
                average: 0.0,
            }
        }
        pub fn add(&mut self, value: i32) {
            self.list.push(value);
            self.update_average();
        }
    }
}
```

Polymorphism?

Polymorphism enables a child type to be used in the same places as the parent type. It means that we can substitute multiple objects for each other at runtime if they share certain characteristics.

- Rust uses generics to abstract over different possible types and trait bounds to impose constraints on what those types must provide. This is sometimes called ***bounded parametric polymorphism***.

```
fn code_020() {
    let both_integer = Point { x: 5, y: 10 };
    let both_float = Point { x: 1.0, y: 4.0 };

    println!("p.x = {}", both_integer.x());
    println!("p.x = {}", both_float.x());

    //println!("p.x = {}",
both_integer.distance_from_origin());
    println!("p.x = {}", both_float.distance_from_origin());
}
```

```
struct Point<T, U> {
    x: T,
    y: U,
}
```

```
impl<T, U> Point<T, U> {
    fn x(&self) -> &T {
        &self.x
    }
}

impl Point<f32, f32> {
    fn distance_from_origin(&self) -> f32 {
        (self.x.powi(2) + self.y.powi(2)).sqrt()
    }
}
```

Inheritance?

Inheritance is a mechanism whereby an object can inherit from another object's definition, thus gaining the parent object's data and behavior without you having to define them again.

- In Rust, there is no way to define a struct that inherits the parent struct's fields and method implementations.
- However, we have code reuse (limited) feature in Rust:

```
struct Tweet {
    username: String,
    content: String,
    reply: bool,
    retweet: bool,
}
```

```
trait Summary {
    fn summarize(&self) -> String;
    fn details(&self) -> String {
        String::from("(Read more...)")
    }
}
```

```
impl Summary for Tweet {
    fn summarize(&self) -> String {
        format!("{}: {}", self.username, self.content)
    }
    fn details(&self) -> String {
        format!("{}: {}", self.reply, self.retweet)
    }
}
```

Is Rust OOP?

Feature	Rust	Support?
Object	Creating a value of struct/enum.	Yes.
Encapsulation	Must use pub keyword to expose internal details of module.	Yes.
Polymorphism	Bounded parametric polymorphism with generics and traits.	Yes.
Inheritance	Limited code reuse capacity.	Yes & No.

Rust is not an ideal OOP language but supports most of its features.

< Is Rust OOP />