Traits in Rust Programming

Hitesh Kumar Sharma
Instructor, Pluralsight

Agenda

- Traits
- Implementation of Traits

Rust – Traits

A trait tells the Rust compiler about functionality a particular type has and can share with other types. An abstract definition of shared behavior among several types is known as a trait. Therefore, traits can be compared to Java interfaces or C++ abstract classes. Other methods within a trait can be accessed by a trait method.

Traits are defined by providing the name of the Trait followed by the keyword “trait”. While defining any Trait we have to provide method declaration inside the Trait.

```
trait Shape {
    fn area(&self) -> f64;
}
```

Implementing a Trait

Implementing a trait is comparable to creating an interface in other programming languages like Java.

Here, we use the term "impl" followed by the name of the trait that needs to be put into action. Next, we use the keyword "for" the implementing type. Inside the impl block, define method bodies for each method declared in the trait's definition.

```
struct Rectangle {
    width: f64,
    height: f64,
}
impl Shape for Rectangle {
    fn area(&self) -> f64 {
        self.width * self.height
    }
}
```

Implementing a Trait (Example)

Let’s implement a trait called Shape Rectangle and Circle struct:

```
trait Shape {
    fn area(&self) -> f64;
}
struct Rectangle {
    width: f64,
    height: f64,
}
impl Shape for Rectangle {
    fn area(&self) -> f64 {
        self.width * self.height
    }
}
struct Circle {
    radius: f64,
}
impl Shape for Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * self.radius * self.radius
    }
}
```

Implementing a Trait (Example)

```
fn main() {
    let rectangle = Rectangle {
        width: 5.0,
        height: 10.0,
    };

    let circle = Circle {
        radius: 7.0,
    };

    println!("Area of the rectangle: {}",
rectangle.area());
    println!("Area of the circle: {}",
circle.area());
}
```

End of Module