Traits

This lesson introduces you to traits.

We'll cover the following

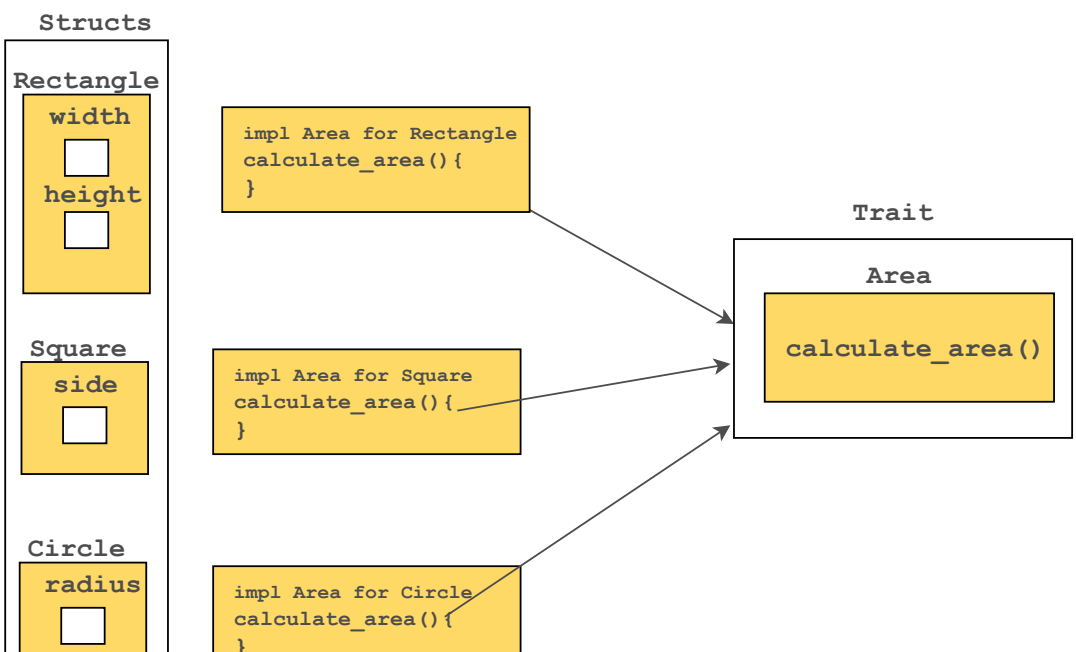
When there are **multiple different types behind a single interface**, the interface can tell which concrete type to access. This is where the traits come in handy.

What Are Traits?

Traits are used to define a standard set of behavior for multiple structs.

They are like **interfaces** in Java.

Suppose you want to calculate area for different shapes. We know that the area is calculated differently for every shape. The best solution is to make a `trait` and define an abstract



method in it and implement that method within every `struct` `impl` construct.

Types of Methods in Traits

There can be two types of methods in traits

- **Concrete Method**

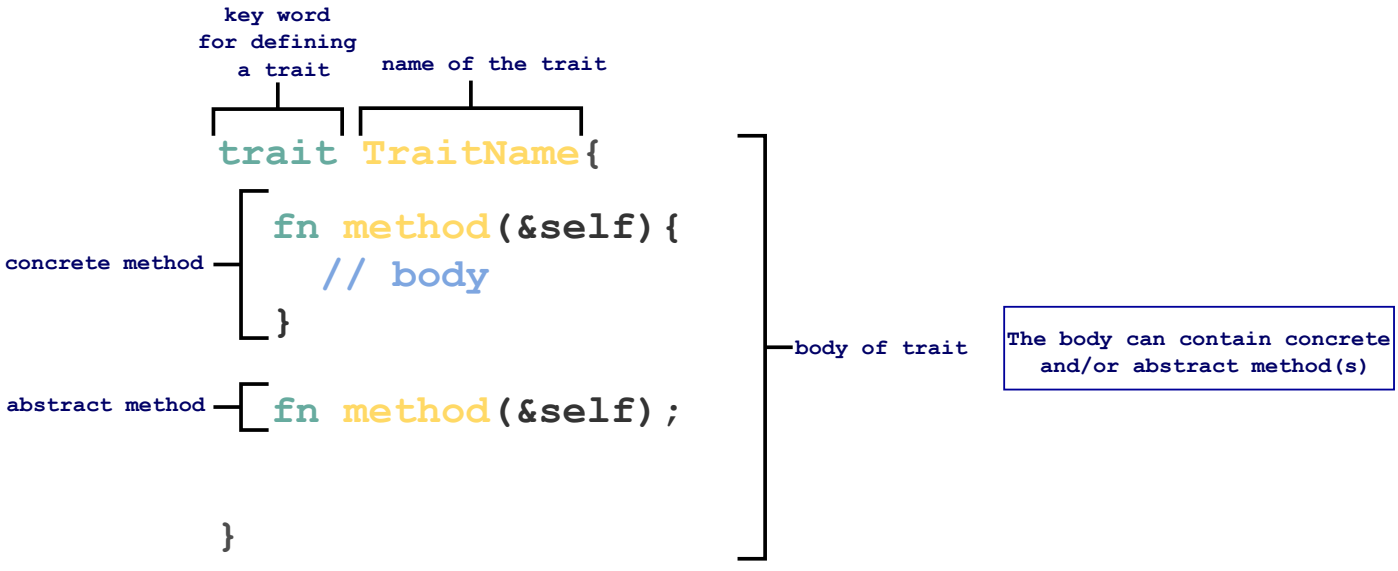
  The method that has a body meaning that implementation of the method is done within the method.

- **Abstract Method**

  The method that does not have a body meaning that implementation of the method is done by each struct in its own `impl` construct.

Declare a Trait

Traits are written with a `trait` keyword.

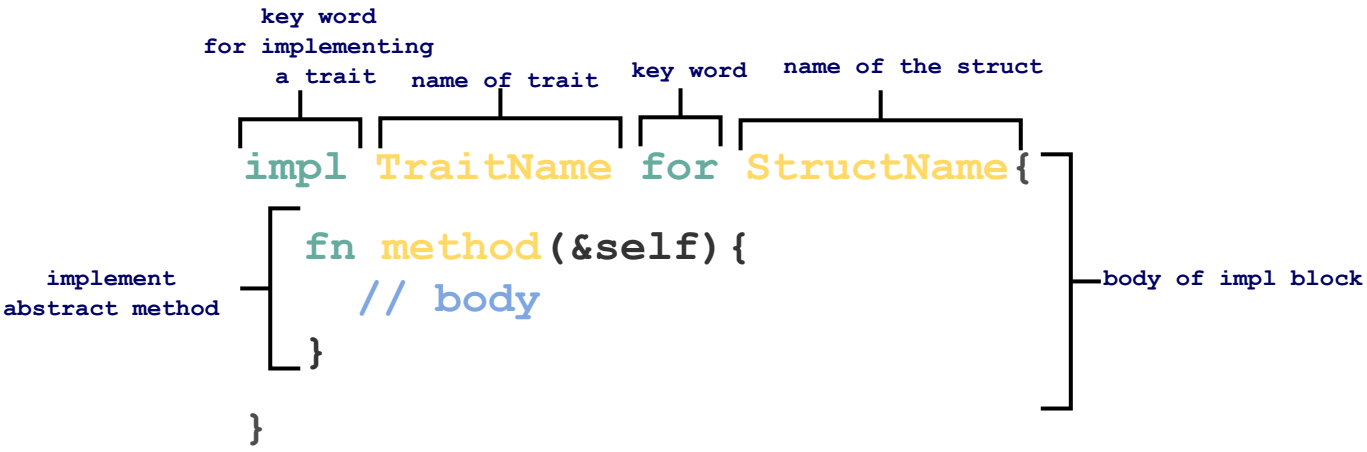


Defining a trait

> **Naming Convention**
>
> Name of the trait is written in **CamelCase**

Implement a trait

Traits can be implemented for any structure.



Implementing a trait

Example

The following example explains the concept of `trait`:

```
fn main(){
   //create an instance of the structure
   let c = Circle  {
      radius : 2.0,
   };
   let r = Rectangle  {
      width : 2.0,
      height : 2.0,
   };
   println!("Area of Circle: {}", c.shape_area());
   println!("Area of Rectangle:{}", r.shape_area());
}
//declare a structure
struct Circle {
   radius : f32,
}
struct Rectangle{
    width : f32,
    height : f32,
}
//declare a trait
trait Area {
   fn shape_area(&self)->f32;
}
//implement the trait
impl Area for Circle {
   fn shape_area(&self)->f32{
      3.13* self.radius *self.radius
   }
}
impl Area for Rectangle {
   fn shape_area(&self)->f32{
```

```
        self.width * self.height
    }
}
```

Explanation

- **`main` function**

  The `main` function is defined from **line 1 to line 12**.

  - On **line 3**, instance `c` of `struct` `Circle` is initialized.
  - On **line 6**, instance `r` of `struct` `Rectangle` is initialized.
  - On **line 10 and 11**, the method of `Circle` and `Rectangle`, `c.shape_area` and `r.shape_area` which is an abstract method declared in `trait` `Area`, is invoked.

- `struct` `Circle`

  On **line 14**, a `struct` `Circle` is defined with item `radius`

- `struct` `Rectangle`

  On **line 17**, a `struct` `Rectangle` is defined with items `width` and `height` respectively.

- `trait` `Area`

  - On **line 22**, a `trait` is defined.
  - An abstract function `shape_area()` is defined inside the `trait`

- `impl` `Area` `for` `Circle`

  This implements the method `shape_area` of traits and returns the area of the circle i.e., $PI * self.radius * self.radius$.

  Here, `self` represents that it's referring to the struct items of Circle.

- `impl` `Area` `for` `Rectangle`

  This implements the method `shape_area` of `traits` and returns the area of the rectangle, i.e., $self.width * self.height$.

Here, `self` represents that it's referring to the struct items of `Rectangle`.

Quiz

Test your understanding of `traits` in Rust.

Quick Quiz on Traits!

1 Which of the following `trait` method allows you to write body of the method?

2 Traits are like interfaces in other object oriented languages.

Retake Quiz

Now that you have learned Traits, let's learn about a generic type "Generics" in the next lesson.