CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 4 Borrowing, Struct, Traits
Sep 6, 2022

Copyright ©, All rights reserved. 2022 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/openpub/) license defines the copyright on this document.

More Borrowing

```
fn main() {
    let v = vec![100, 32, 57];
    for i in v {
        println!("{}", i);
    }
    v[0];
}
```

```
fn main() {
    let mut v = vec![100, 32, 57];
    for i in &v {
        println!("{}", i);
    };
    v[0] = 9;
}
```

```
fn main() {
    let v = vec![100, 32, 57];
    for i in &v {
        println!("{}", i);
    }
    v[0];
}
```

```
fn main() {
    let mut v = vec![100, 32, 57];
    for i in &v {
        println!("{}", i);
        v[0] = 9;
    };
}
```

More Borrowing

```
fn main() {
    let mut v = vec![100, 32, 57];
    for i in &mut v {
        *i += 50;
    }
    assert_eq!(150, v[0]);
}
```

Slices are References

```
fn main() {
    let s = String::from("hello world");

    let hello = &s[0..5];
    let world = &s[6..11];
}
```

```
#![allow(unused)]
fn main() {
let a = [1, 2, 3, 4, 5];

let slice = &a[1..3];

assert_eq!(slice, &[2, 3]);
}
```

```
#![allow(unused)]
fn main() {
let s = String::from("hello");

let slice = &s[0..2];
let slice = &s[..2];

let len = s.len();
let slice = &s[3..len];
let slice = &s[3..];

let slice = &s[0..len];
let slice = &s[..];
}
```

Lifetime of a Reference

Scope for which a reference is valid

A reference can not out live is borrowed value

```
fn main() {
    {
        let r;

        {
            let x = 5;
            r = &x;
        }

        println!("r: {}", r);
    }
}
```

Lifetimes

```
fn main() {
    let reference_to_nothing = dangle();
}

fn dangle() -> &String {

    let s = String::from("hello"); // s is a new String

    &s // Compile error
} /
```

```
fn main() {
    let string = no_dangle();
}

fn no_dangle() -> String {
    let s = String::from("hello");

    s
}
```

A reference can not out live the value

Structs

```
struct User {
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64,
}
```

```
fn build_user(email: String, username: String) -> User {
    User {
        email,
        username,
        active: true,
        sign_in_count: 1,
    }
}
```

```
fn main() {
    let mut user1 = User {
        email: String::from("someone@example.com"),
        username: String::from("someusername123"),
        active: true,
        sign_in_count: 1,
    };

    user1.email = String::from("anotheremail@example.com");
}
```

```
    let user2 = User {
        email: String::from("another@foo.com"),
        ..user1
    };
}
```

Tuple & Unit Structs

```
struct Color(i32, i32, i32);
struct Point(i32, i32, i32);

fn main() {
    let black = Color(0, 0, 0);
    let origin = Point(0, 0, 0);
}
```

```
struct AlwaysEqual;

fn main() {
    let subject = AlwaysEqual;
}
```

Adding Methods to struct

```
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}

impl Rectangle {
    fn size(&self) -> u32 {
        12
    }
}
```

```
fn main() {
    let rect1 = Rectangle {
        width: 30,
        height: 50,
    };

    println!(
        "The area of the rectangle is {} square pixels.",
        rect1.area()
    );
}

impl Rectangle {
    fn width(&self) -> bool {
        self.width > 0
    }
}
```

Static Methods

```
#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn square(size: u32) -> Rectangle {
        Rectangle {
            width: size,
            height: size,
        }
    }
}

fn main() {
    let sq = Rectangle::square(3);
}
```

Access Level of methods & Fields

Rust
- Public in module,
- Private outside unless marked pub

Python
- Everything is public

Smalltalk
- Fields private
- Methods public

Java
- Public,
- Protected - subclass & Package
- Private

Swift
- Open
  - Code outside of module can subclass & override
- Public
  - Access anywhere
- Internal
  - Access from any file in the module
- File-private
  - Accessible only in the source file it is defined
- Private
  - Only accessible in enclosing declaration

Default is Internal - recommended

Why &self as First Argument?

```
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}
```

```
fn main() {
    let rect1 = Rectangle {
        width: 30,
        height: 50,
    };

    rect1.area();
    Rectangle::area(&rect1);
    assert_eq!(rect1.area(),  Rectangle::area(&rect1));
}
```

Records, Structs, Classes

Records (struct)

- Group of heterogeneous fields
- Introduced in Cobol

C, C++ struct

- Group of heterogeneous fields

Class

- Group of heterogeneous fields + methods on fields
- C++, Java, Swift, Kotlin, Python, ...

Swift - Struct & Class have Methods

```
class Counter {
    var count = 0

    func increment() {
        count += 1;
    }

    func set(to count:Int)->() {
        self.count = count
    }

    func increment(by amount:Int, repeated:Int) {
        count = count + amount*repeated
    }
}
```

```
struct Point {
    var x:Int = 0
    var y:Int = 0

    func sample()->String {
        return "(\(x),\(y))"
    }

    mutating func moveBy(deltaX:Int, deltaY:Int) {
        self.x += deltaX
        self.y += deltaY
    }

    mutating func setTo(newX:Int, newY:Int) {
        self = Point(x:newX, y:newY)
    }
}
```

Swift - Value versus Reference Semantics

- Structure
  - Value type
  - Copied when
    - Assigned to variable
    - Passed as a parameter

Swift value type = Rust Copy

- Class
  - Reference type
  - Shared when
    - Assigned to variable
    - Passed as a parameter

Rust & Inheritance

Rust does not support inheritance of structs

Why not?

Performance

a.foo();

With inheritance foo() is bound at runtime

Without inheritance foo() is bound at compile time

Trait

Functionality that can be shared with other types

Like Java interface, but also

- Implement methods
- Define static methods
- Traits can be subclassed
- Add trait to existing types

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

Trait Example

```
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}
```

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

impl Summary for Rectangle {
    fn summarize(&self) -> String {
        format!("{}x{}", self.width, self.height)
    }
}
```

```
fn main() {
    let rect1 = Rectangle {
        width: 30,
        height: 50,
    };

    println!("The area of the rectangle is {} square pixels.", rect1.summarize());
}
```

Trait as Parameter

```
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}
```

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

impl Summary for Rectangle {
    fn summarize(&self) -> String {
        format!("{}x{}", self.width, self.height)
    }
}
```

```
pub fn notify(item: &impl Summary) {
    println!("Breaking news! {}", item.summarize());
}
```

Adding method to i32

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

impl Summary for i32 {
    fn summarize(&self) -> String {
        format!("i32: {}", self)
    }
}

fn main() {
    12.summarize();
    println!("{}", 42.summarize());
}
```