600086 Lab Book

Week 4 – 600086-Lab-D

Date: 1st Mar 2022

Q1. Ownership limitations

Question:

Try to expand the code to include a data member in Engine that links to the Aircraft.

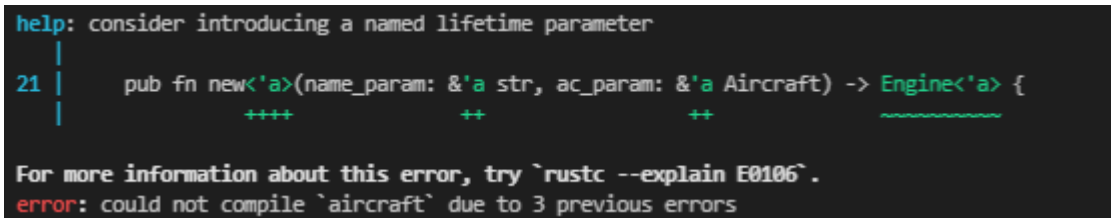
Solution:

```
struct Aircraft<'a> {
    name: String,
    engines: Vec<&'a Engine>,
}
impl Aircraft<'_> {
    pub fn new(name_param: &str) -> Aircraft {
        Aircraft {
            name: name_param.to_string(),
            engines: Vec::new()
        }
    }
}
struct Engine<'a> {
    aircraft : Aircraft<'a>,
    name: String,
}
impl Engine {
    pub fn new(name_param: &str, ac_param: &Aircraft) -> Engine {
        Engine {
            name: name_param.to_string(),
            aircraft: ac_param
        }
    }
}

fn main() {
    let engine1 = Engine::new( "General Electric F404" );
    let engine2 = Engine::new( "General Electric F404" );
    let mut f18 = Aircraft::new( "F-18" );

    f18.engines.push (&engine1);
    f18.engines.push (&engine2);

    println! ("Aircraft: {} has a {} and {} ", f18.name, f18.engines[0].name,
f18.engines[1].name );
}
```

Test data:

n/a

Sample output:



Reflection:

This cannot be done as the engines are owned by the aircraft class therefore the engines cannot own a reference to the aircraft in the way specified as this would create a circular reference.

Metadata:

F18 , Aircraft,ownership

Further information:

Q2. Reference counters 1

Question:

Use your knowledge of the Rust ownership model to explain what is happening with the reference counters and why we do not need to pass them as references.

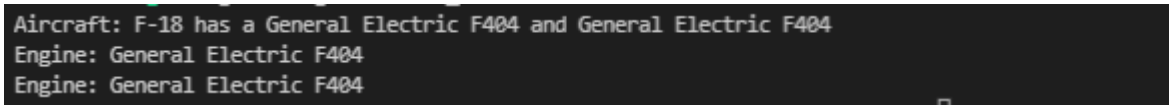
Solution:

```
use std::rc::Rc;

struct Aircraft {
    name: String,
    engines: Vec<Rc<Engine>>,
}

impl Aircraft {
    pub fn new(name_param: &str) -> Aircraft {
        Aircraft {
            name: name_param.to_string(),
            engines: Vec::new()
        }
    }
}

struct Engine {
    name: String,
}

impl Engine {
    pub fn new(name_param: &str) -> Engine {
        Engine {
            name: name_param.to_string(),
        }
    }
}

fn main() {
    let engine1 = Rc::new(Engine::new( "General Electric F404" ));
    let engine2 = Rc::new(Engine::new( "General Electric F404" ));

    let mut f18 = Aircraft::new( "F-18" );

    f18.engines.push (engine1.clone());
    f18.engines.push (engine2.clone());

    println! ("Aircraft: {} has a {} and {}", f18.name, f18.engines[0].name ,
f18.engines[1].name );
    println! ("Engine: {} ", engine1.name );
    println! ("Engine: {} ", engine2.name );
}
```

Test data:

n/a

Sample output:



Reflection:

The RCs do not require passing in as references as the engines are owned by the RC the engine1 and engine2 variables are actually RC references to the memory locations where the instance of the Engine is stored so when creating the aircraft rather than moving ownership it creates a new instance that is a clone of the one owned by the RC.

Metadata:

F18 , Aircraft,ownership

Further information:

Q2. Reference counters 2

Question:

Remove the clone() method from this line
f18.engines.push (engine1.clone());
Can you explain why this program now fails to build?

Solution:

```
use std::rc::Rc;

struct Aircraft {
    name: String,
    engines: Vec<Rc<Engine>>,
}

impl Aircraft {
    pub fn new(name_param: &str) -> Aircraft {
        Aircraft {
            name: name_param.to_string(),
            engines: Vec::new()
        }
    }
}

struct Engine {
    name: String,
}

impl Engine {
    pub fn new(name_param: &str) -> Engine {
        Engine {
            name: name_param.to_string(),
        }
    }
}

fn main() {
    let engine1 = Rc::new(Engine::new( "General Electric F404" ));
    let engine2 = Rc::new(Engine::new( "General Electric F404" ));

    let mut f18 = Aircraft::new( "F-18" );

    f18.engines.push (engine1);
    f18.engines.push (engine2.clone());

    println! ("Aircraft: {} has a {} and {}", f18.name, f18.engines[0].name ,
f18.engines[1].name );
    println! ("Engine: {} ", engine1.name );
    println! ("Engine: {} ", engine2.name );
}
```

Test data:

n/a

Sample output:

```
error: could not compile `Reference_Counters` due to previous error
```

Reflection:

The code will no longer compile because unlike before where the instanced engine object ref is cloned into a new ref of engine when passed in, the code is now attempting to pass an object that is of type RC<Engine> which is not an engine but is actually the memory location where the RC is storing the engine object and therefore lacks the required properties that are called for in the programme e.g name.

Metadata:

F18 , Aircraft,ownership

Further information:

Q2. Reference counters 3

Question:

You should be able to service engine3.
You will get a build error if you try and service engine2, which is accessed through an rc.
Again, using your knowledge of the Rust ownership module can you explain why the error is occurring?

Solution:

```
use std::rc::Rc;
struct Aircraft {
    name: String,
    engines: Vec<Rc<Engine>>,
}
impl Aircraft {
    pub fn new(name_param: &str) -> Aircraft {
        Aircraft {
            name: name_param.to_string(),
            engines: Vec::new()
        }
    }
}
struct Engine {
    name: String,
    requires_service: bool,
}
impl Engine {
    pub fn new(name_param: &str) -> Engine {
        Engine {
            name: name_param.to_string(),
            requires_service: false,
        }
    }
    pub fn service(&mut self) {
        self.requires_service = false;
    }
}
fn main() {
    let engine1 = Rc::new(Engine::new( "General Electric F404" ));
    let engine2 = Rc::new(Engine::new( "General Electric F404" ));
    let mut engine3 = Engine::new( "General Electric F404" );
    engine2.service();
    let mut f18 = Aircraft::new( "F-18" );
    f18.engines.push (engine1.clone());
    f18.engines.push (engine2.clone());

    println! ("Aircraft: {} has a {} and {}", f18.name, f18.engines[0].name , f18.engines[1].name );
    println! ("Engine: {} ", engine1.name );
    println! ("Engine: {} ", engine2.name );
}
```

Test data:

n/a

Sample output:

```
error: could not compile `Reference_Counters` due to previous error; 2 warnings emitted
```

Reflection:

The Program will not build when attempting to call service() on engine2 as the reference owned by the RC is immutable and so cannot be borrowed as a mutable reference, this is an issue as the service() requires a mutable reference as it writes to the properties of the engine.

Metadata:

F18 , Aircraft,ownership

Further information: