

A trait object has some specific lifetime '1, but it was used in a way that requires it to have a 'static lifetime.

Example of erroneous code:

```
trait BooleanLike {}
trait Person {}

impl BooleanLike for bool {}

impl dyn Person {
    fn is_cool(&self) -> bool {
        // hey you, you're pretty cool
        true
    }
}

fn get_is_cool<'p>(person: &'p dyn Person) -> impl BooleanLike {
    // error: `person` has an anonymous lifetime ``p` but calling
    //      `print_cool_fn` introduces an implicit ``static` lifetime
    //      requirement
    person.is_cool()
}
```

The trait object `person` in the function `get_is_cool`, while already being behind a reference with lifetime 'p, also has it's own implicit lifetime, '2.

Lifetime '2 represents the data the trait object might hold inside, for example:

```
trait MyTrait {}

struct MyStruct<'a>(&'a i32);

impl<'a> MyTrait for MyStruct<'a> {}
```

With this scenario, if a trait object of `dyn MyTrait + '2` was made from `MyStruct<'a>`, 'a must live as long, if not longer than '2. This allows the trait object's internal data to be accessed safely from any trait methods. This rule also goes for any lifetime any struct made into a trait object may have.

In the implementation for `dyn Person`, the '2 lifetime representing the internal data was omitted, meaning that the compiler inferred the lifetime 'static. As a result, the implementation's `is_cool` is inferred by the compiler to look like this:

```
# trait Person {}
#
# impl dyn Person {
fn is_cool<'a>(self: &'a (dyn Person + 'static)) -> bool {unimplemented!()}
# }
```

While the `get_is_cool` function is inferred to look like this:

```
# trait Person {}
# trait BooleanLike {}
#
fn get_is_cool<'p, R: BooleanLike>(person: &'p (dyn Person + 'p)) -> R {
    unimplemented!()
}
```

Which brings us to the core of the problem; the assignment of type `&'_ (dyn Person + '_)` to type `&'_ (dyn Person + 'static)` is impossible.

Fixing it is as simple as being generic over lifetime '2, as to prevent the compiler from inferring it as 'static:

```
# trait Person {}
#
impl<'d> dyn Person + 'd { /* ... */ }

// This works too, and is more elegant:
//impl dyn Person + '_ { /* ... */ }
```

See the [\[Rust Reference on Trait Object Lifetime Bounds\]](#)[\[trait-objects\]](#) for more information on trait object lifetimes.