```
[[questions]]
type = "Tracing"
prompt.program = """
#[derive(Copy, Drop)]
struct Rectangle {
    width: u64,
    height: u64,
}
#[generate_trait]
impl RectangleImpl of RectangleTrait {
    fn area(self: @Rectangle) -> u64 {
        (*self.width) * (*self.height)
    }
    fn new(width: u64, height: u64) -> Rectangle {
        Rectangle { width, height }
    }
    fn compare_areas(self: @Rectangle, r2: @Rectangle) -> bool {
        self.area() == r2.area()
    }
}
fn main() {
    let rect1 = Rectangle {width: 40, height: 50};
    let rect2 = RectangleTrait::new(10, 40);
    println!("{}", rect1.compare_areas(@rect2));
}
"""
answer.doesCompile = true
answer.stdout = "false"
context = """
It compiles, because the type `Rectangle` on which we call the method on reference is
implicitly passed as a `@Rectangle`
"""
id = "98bbc25c-80b2-4226-9219-a8d7b20fb991"
[[questions]]
type = "Tracing"
prompt.program = """
#[derive(Drop)]
struct Rectangle {
    width: u64,
    height: u64,
}
#[derive(Drop)]
struct Circle {
    radius: u64,
}
trait RectangleTrait {
```

```
    fn area(self: @Rectangle) -> u64;
}
impl RectangleImpl of RectangleTrait {
    fn area(self: @Rectangle) -> u64 {
        return (*self.width) * (*self.height);
    }
}
fn main() {
    let my_square = Rectangle { width: 30, height: 50 };
    let my_circle = Circle { radius: 10 };
    let area = my_circle.area();
    println!("{}", area)
}
"""
answer.doesCompile = false
answer.lineNumber = 25
context = """
Methods can only be called on the types they're defined for; here, we try to call `area()`
on `Circle`, when it's defined for `Rectangle`.
"""
id = "1e4a5bff-dc92-4c09-9f04-1d95fcf039bb"
```