

The compiler doesn't know what method to call because more than one method has the same prototype.

Erroneous code example:

```
struct Test;

trait Trait1 {
    fn foo();
}

trait Trait2 {
    fn foo();
}

impl Trait1 for Test { fn foo() {} }
impl Trait2 for Test { fn foo() {} }

fn main() {
    Test::foo() // error, which foo() to call?
}
```

To avoid this error, you have to keep only one of them and remove the others. So let's take our example and fix it:

```
struct Test;

trait Trait1 {
    fn foo();
}

impl Trait1 for Test { fn foo() {} }

fn main() {
    Test::foo() // and now that's good!
}
```

However, a better solution would be using fully explicit naming of type and trait:

```
struct Test;

trait Trait1 {
    fn foo();
}

trait Trait2 {
    fn foo();
}

impl Trait1 for Test { fn foo() {} }
impl Trait2 for Test { fn foo() {} }

fn main() {
```

```
<Test as Trait1>::foo()  
}
```

One last example:

```
trait F {  
    fn m(&self);  
}  
  
trait G {  
    fn m(&self);  
}  
  
struct X;  
  
impl F for X { fn m(&self) { println!("I am F"); } }  
impl G for X { fn m(&self) { println!("I am G"); } }  
  
fn main() {  
    let f = X;  
  
    F::m(&f); // it displays "I am F"  
    G::m(&f); // it displays "I am G"  
}
```