A type that is not a trait was used in a trait position, such as a bound or `impl`.

Erroneous code example:

```
struct Foo;
struct Bar;

impl Foo for Bar {} // error: `Foo` is not a trait
fn baz<T: Foo>(t: T) {} // error: `Foo` is not a trait
```

Another erroneous code example:

```
type Foo = Iterator<Item=String>;

fn bar<T: Foo>(t: T) {} // error: `Foo` is a type alias
```

Please verify that the trait's name was not misspelled or that the right identifier was used. Example:

```
trait Foo {
    // some functions
}
struct Bar;

impl Foo for Bar { // ok!
    // functions implementation
}

fn baz<T: Foo>(t: T) {} // ok!
```

Alternatively, you could introduce a new trait with your desired restrictions as a super trait:

```
# trait Foo {}
# struct Bar;
# impl Foo for Bar {}
trait Qux: Foo {} // Anything that implements Qux also needs to implement Foo
fn baz<T: Qux>(t: T) {} // also ok!
```

Finally, if you are on nightly and want to use a trait alias instead of a type alias, you should use `#![feature(trait_alias)]`:

```
#![feature(trait_alias)]
trait Foo = Iterator<Item=String>;

fn bar<T: Foo>(t: T) {} // ok!
```