Note: this error code is no longer emitted by the compiler.

`where` clauses must use generic type parameters: it does not make sense to use them otherwise. An example causing this error:

```
trait Foo {
    fn bar(&self);
}

#[derive(Copy,Clone)]
struct Wrapper<T> {
    Wrapped: T
}

impl Foo for Wrapper<u32> where Wrapper<u32>: Clone {
    fn bar(&self) { }
}
```

This use of a `where` clause is strange - a more common usage would look something like the following:

```
trait Foo {
    fn bar(&self);
}

#[derive(Copy,Clone)]
struct Wrapper<T> {
    Wrapped: T
}
impl <T> Foo for Wrapper<T> where Wrapper<T>: Clone {
    fn bar(&self) { }
}
```

Here, we're saying that the implementation exists on Wrapper only when the wrapped type `T` implements `Clone`. The `where` clause is important because some types will not implement `Clone`, and thus will not get this method.

In our erroneous example, however, we're referencing a single concrete type. Since we know for certain that `Wrapper<u32>` implements `Clone`, there's no reason to also specify it in a `where` clause.