@name Import Cycle Detected @category compiler @shortDescription Import cycles would need to be created to compile this component

@description

A component, directive or pipe that is referenced by this component would require the compiler to add an import that would lead to a cycle of imports. For example, consider a scenario where a `ParentComponent` references a `ChildComponent` in its template:

There is already an import from `child.component.ts` to `parent.component.ts` since the `ChildComponent` references the `ParentComponent` in its constructor.

But note that the parent component's template contains `<child></child>`. The generated code for this template must therefore contain a reference to the `ChildComponent` class. In order to make this reference the compiler would have to add an import from `parent.component.ts` to `child.component.ts`, which would cause an import cycle:

```
parent.component.ts -> child.component.ts -> parent.component.ts
```

### Remote Scoping

To avoid adding imports that create cycles, additional code is added to the `NgModule` class where the component is declared that wires up the dependencies. This is known as "remote scoping".

### Libraries

Unfortunately, "remote scoping" code is side-effectful, which prevents tree shaking, and cannot be used in libraries. So when building libraries using the `"compilationMode": "partial"` setting, any component that would require a cyclic import will cause this `NG3003` compiler error to be raised.

@debugging

The cycle that would be generated is shown as part of the error message. For example:

The component ChildComponent is used in the template but importing it would create a cycle: /parent.component.ts -> /child.component.ts -> /parent.component.ts
Use this to identify how the referenced component, pipe or directive has a dependency back to the component being compiled. Here are some ideas for fixing the problem:

- Try to re-arrange your dependencies to avoid the cycle. For example using an intermediate interface that is stored in an independent file that can be imported to both dependent files without causing an import cycle.
- Move the classes that reference each other into the same file, to avoid any imports between them.
- Convert import statements to type-only imports (using `import type` syntax) if the imported declarations are only used as types, as type-only imports do not contribute to cycles.