```
module B {
    var A = 1;
    import Y = A;
}
```

'Y' is a local alias for the non-instantiated module 'A'. If the declaration of 'A' is changed such that 'A' becomes an instantiated module, for example by including a variable declaration in 'A', the import statement in 'B' above would be an error because the expression 'A' doesn't reference the module instance of module 'A'.

When an import statement includes an export modifier, all meanings of the local alias are exported.

10.4 Export Declarations

An export declaration declares an externally accessible module member. An export declaration is simply a regular declaration prefixed with the keyword `export`.

Exported class, interface, and enum types can be accessed as a *TypeName* (section 3.6.2) of the form *M.T*, where *M* is a reference to the containing module and *T* is the exported type name. Likewise, as part of a *TypeName*, exported modules can be accessed as a *ModuleName* of the form *M.N*, where *M* is a reference to the containing module and *N* is the exported module.

Exported variable, function, class, enum, module, and import alias declarations become properties on the module instance and together establish the module's ***instance type***. This unnamed type has the following members:

- A property for each exported variable declaration.
- A property of a function type for each exported function declaration.
- A property of a constructor type for each exported class declaration.
- A property of an object type for each exported enum declaration.
- A property of an object type for each exported instantiated module declaration.
- A property for each exported import alias that references a variable, function, class, enum, or instantiated module.

An exported member depends on a (possibly empty) set of named types (section 3.5). Those named types must be at least as accessible as the exported member, or otherwise an error occurs.

The named types upon which a member depends are the named types occurring in the transitive closure of the ***directly depends on*** relationship defined as follows:

- A variable directly depends on the *Type* specified in its type annotation.
- A function directly depends on each *Type* specified in a parameter or return type annotation.
- A class directly depends on each *Type* specified as a type parameter constraint, each *TypeReference* specified as a base class or implemented interface, and each *Type* specified in a constructor parameter type annotation, member variable type annotation, member function