and enums), and a declaration space for local namespaces (containers of named types). Every declaration (whether local or exported) in a module contributes to one or more of these declaration spaces.

10.3 Import Declarations

Import declarations are used to create local aliases for entities in other modules.

ImportDeclaration:
`import` *Identifier* `=` *EntityName* `;`

EntityName:
ModuleName
ModuleName `.` *Identifier*

An *EntityName* consisting of a single identifier is resolved as a *ModuleName* and is thus required to reference an internal module. The resulting local alias references the given internal module and is itself classified as an internal module.

An *EntityName* consisting of more than one identifier is resolved as a *ModuleName* followed by an identifier that names one or more exported entities in the given module. The resulting local alias has all the meanings and classifications of the referenced entity or entities. (As many as three distinct meanings are possible for an entity name—namespace, type, and member.) In effect, it is as if the imported entity or entities were declared locally with the local alias name.

In the example

```
module A {
    export interface X { s: string }
    export var X: X;
}

module B {
    interface A { n: number }
    import Y = A;    // Alias only for module A
    import Z = A.X;  // Alias for both type and member A.X
    var v: Z = Z;
}
```

within 'B', 'Y' is an alias only for module 'A' and not the local interface 'A', whereas 'Z' is an alias for all exported meanings of 'A.X', thus denoting both an interface type and a variable.

If the *ModuleName* portion of an *EntityName* references an instantiated module, the *ModuleName* is required to reference the module instance when evaluated as an expression. In the example

```
module A {
    export interface X { s: string }
}
```