An interface has the members specified in the *ObjectType* of its declaration and furthermore inherits all base type members that aren't hidden by declarations in the interface:

- A property declaration hides a public base type property with the same name.
- A call signature declaration hides a base type call signature that is identical when return types are ignored.
- A construct signature declaration hides a base type construct signature that is identical when return types are ignored.
- A string index signature declaration hides a base type string index signature.
- A numeric index signature declaration hides a base type numeric index signature.

The following constraints must be satisfied by an interface declaration or otherwise a compile-time error occurs:

- An interface declaration may not, directly or indirectly, specify a base type that originates in the same declaration. In other words an interface cannot, directly or indirectly, be a base type of itself, regardless of type arguments.
- An interface cannot declare a property with the same name as an inherited private property.
- Inherited properties with the same name must be identical (section 3.8.2).
- All properties of the interface must satisfy the constraints implied by the index signatures of the interface as specified in section 3.7.4.
- The instance type (section 3.5.1) of the declared interface must be a subtype (section 3.8.3) of each of the base type references.

An interface is permitted to inherit identical members from multiple base types and will in that case only contain one occurrence of each particular member.

Below is an example of two interfaces that contain properties with the same name but different types:

```
interface Mover {
    move(): void;
    getStatus(): { speed: number; };
}

interface Shaker {
    shake(): void;
    getStatus(): { frequency: number; };
}
```

An interface that extends 'Mover' and 'Shaker' must declare a new 'getStatus' property as it would otherwise inherit two 'getStatus' properties with different types. The new 'getStatus' property must be declared such that the resulting 'MoverShaker' is a subtype of both 'Mover' and 'Shaker':

```
interface MoverShaker extends Mover, Shaker {
    getStatus(): { speed: number; frequency: number; };
}
```