Since function and constructor types are just object types containing call and construct signatures, interfaces can be used to declare named function and constructor types. For example:

```
interface StringComparer { (a: string, b: string): number; }
```

This declares type 'StringComparer' to be a function type taking two strings and returning a number.

7.2 Declaration Merging

Interfaces are "open-ended" and interface declarations with the same qualified name relative to a common root (as defined in section 2.3) contribute to a single interface.

When a generic interface has multiple declarations, all declarations must have identical type parameter lists, i.e. identical type parameter names with identical constraints in identical order.

In an interface with multiple declarations, the `extends` clauses are merged into a single set of base types and the bodies of the interface declarations are merged into a single object type.

7.3 Interfaces Extending Classes

When an interface type extends a class type it inherits the members of the class but not their implementations. It is as if the interface had declared all of the members of the class without providing an implementation. Interfaces inherit even the private members of a base class. When a class containing private members is the base type of an interface type, that interface type can only be implemented by that class or a descendant class. For example:

```
class Control {
    private state: any;
}

interface SelectableControl extends Control {
    select(): void;
}

class Button extends Control {
    select() { }
}

class TextBox extends Control {
    select() { }
}

class Image extends Control {
}

class Location {
    select() { }
}
```