```
interface Document {
    createElement(tagName: "div"): HTMLDivElement;
    createElement(tagName: "span"): HTMLSpanElement;
    createElement(tagName: "canvas"): HTMLCanvasElement;
    createElement(tagName: string): HTMLElement;
}
```

states that calls to 'createElement' with the string literals "div", "span", and "canvas" return values of type 'HTMLDivElement', 'HTMLSpanElement', and 'HTMLCanvasElement' respectively, and that calls with all other string expressions return values of type 'HTMLElement'.

When writing overloaded declarations such as the one above it is important to list the non-specialized signature last. This is because overload resolution (section 4.12.1) processes the candidates in declaration order and picks the first one that matches.

Every specialized call or construct signature in an object type must be a subtype of at least one non-specialized call or construct signature in the same object type (where a call signature *A* is considered a subtype of another call signature *B* if an object type containing only *A* would be a subtype of an object type containing only *B*). For example, the 'createElement' property in the example above is of a type that contains three specialized signatures, all of which are subtypes of the non-specialized signature in the type.

3.7.3 Construct Signatures

A construct signature defines the parameter list and return type associated with applying the `new` operator (section 4.11) to an instance of the containing type. A type may overload `new` operations by defining multiple construct signatures with different parameter lists.

ConstructSignature:
`new` *TypeParameters*$_{opt}$ `(` *ParameterList*$_{opt}$ `)` *TypeAnnotation*$_{opt}$

The type parameters, parameter list, and return type of a construct signature are subject to the same rules as a call signature.

A type containing construct signatures is said to be a ***constructor type***.

It is an error for a type to declare multiple construct signatures that are considered identical (section 3.8.2) or differ only by their return types.

3.7.4 Index Signatures

An index signature defines a type constraint for properties in the containing type.

IndexSignature:
`[` *Identifier* `:` `string` `]` *TypeAnnotation*
`[` *Identifier* `:` `number` `]` *TypeAnnotation*

There are two kinds of index signatures: