The map method, invoked on an array 'a' with element type 'T', will apply function 'func' to each element of 'a', returning a value of type 'U'.

The TypeScript compiler can often infer generic method parameters, making it unnecessary for the programmer to explicitly provide them. In the following example, the compiler infers that parameter 'U' of the map method has type 'string', because the function passed to map returns a string.

```
function numberToString(a: number[]) {
    var stringArray = a.map(v => v.toString());
    return stringArray;
}
```

The compiler infers in this example that the 'numberToString' function returns an array of strings.

In TypeScript, classes can also have type parameters. The following code declares a class that implements a linked list of items of type 'T'. This code illustrates how programmers can *constrain* type parameters to extend a specific type. In this case, the items on the list must extend the type 'NamedItem'. This enables the programmer to implement the 'log' function, which logs the name of the item.

```
interface NamedItem {
    name: string;
}

class List<T extends NamedItem> {
    next: List<T> = null;

    constructor(public item: T) {
    }

    insertAfter(item: T) {
        var temp = this.next;
        this.next = new List(item);
        this.next.next = temp;
    }

    log() {
        console.log(this.item.name);
    }

    // ...
}
```

Section 3.5 provides further information about generic types.

## 1.10 Modules

Classes and interfaces support large-scale JavaScript development by providing a mechanism for describing how to use a software component that can be separated from that component's implementation. TypeScript enforces *encapsulation* of implementation in classes at design time (by