```
class CheckingAccount extends BankAccount {
    constructor(balance: number) {
        super(balance);
    }
    writeCheck(debit: number) {
        this.balance -= debit;
    }
}
```

In this example, the class 'CheckingAccount' *derives* from class 'BankAccount'. The constructor for 'CheckingAccount' calls the constructor for class 'BankAccount' using the 'super' keyword. In the emitted JavaScript code, the prototype of 'CheckingAccount' will chain to the prototype of 'BankingAccount'.

TypeScript classes may also specify static members. Static class members become properties of the class constructor.

Section 8 provides additional information about classes.

1.7 Enum Types

TypeScript enables programmers to summarize a set of numeric constants as an *enum type*. The example below creates an enum type to represent operators in a calculator application.

```
enum Operator {
    ADD,
    DIV,
    MUL,
    SUB
}

function compute(op: Operator, a: number, b: number) {
    console.log("the operator is" + Operator[op]);
    // ...
}
```

In this example, the compute function logs the operator 'op' using a feature of enum types: reverse mapping from the enum value ('op') to the string corresponding to that value. For example, the declaration of 'Operator' automatically assigns integers, starting from zero, to the listed enum members. Section 9 describes how programmers can also explicitly assign integers to enum members, and can use any string to name an enum member.

If all enum members have explicitly assigned literal integers, or if an enum has all members automatically assigned, the TypeScript compiler will emit for an enum member a JavaScript constant corresponding to that member's assigned value (annotated with a comment). This improves performance on many JavaScript engines.

For example, the 'compute' function could contain a switch statement like the following.