✓ 100 XP

Lab - Use interfaces in TypeScript

20 minutes

In this lab, you'll convert some JavaScript code to strongly typed code using interfaces.

The JavaScript code contains two functions: `calculateInterestOnlyLoanPayment`, which calculates the payment for an interest only loan, and `calculateConventionalLoanPayment`, which calculates the payment for a conventional loan. As with most loan calculations, both functions accept `principal` and `interestRate` parameters. The difference between them is that the `calculateConventionalLoanPayment` function accepts a third property, `months` that the `calculateInterestOnlyLoanPayment` function does not.

Property	Description
`principal`	The principal amount of the loan.
`interestRate`	The annual interest rate of the loan. For example, 5% is specified as 5.
`months`	The term of the loan specified in months. An interest only loan does not require this property because the number of months is irrelevant (the loan will never be repaid when an interest only payment is made each month.)

In this exercise, you will:

1. Declare an interface called `Loan` that defines two properties, `principal` and `interestRate`.
2. Declare an interface called `ConventionalLoan` that extends `Loan`, and defines the additional property required for a conventional loan, `months`.
3. Update the two functions to implement the new interfaces and strongly type the parameters.

Exercise 1 - Declare the interfaces

1. Clone the starting repository by entering the following at the command prompt.

   Bash

   ```bash
   git clone https://github.com/MicrosoftDocs/mslearn-typescript
   cd mslearn-typescript/code/module-03/m03-start
   code .
   ```

2. Open the file **module03.ts**.

3. Locate `TODO: Declare the Loan interface`. Declare an interface called `Loan` that defines two properties, `principal` and `interestRate`, each as a `number`.

   TypeScript

   ```typescript
   interface Loan {
       principal: number,
       interestRate: number    //* Interest rate percentage (eg. 14 is 14%)
   }
   ```

4. Locate `TODO: Declare the ConventionalLoan interface`. Declare an interface called `ConventionalLoan` that extends `Loan`, and defines the additional property required for a conventional loan, `months`, as a `number`.

   TypeScript

   ```typescript
   interface ConventionalLoan extends Loan {
       months: number      //* Total number of months
   }
   ```

Exercise 2 - Implement the interfaces

1. Locate `TODO: Update the calculateInterestOnlyLoanPayment function`. Replace the two parameters in the `calculateInterestOnlyLoanPayment` function with an object of type `Loan` (for example, `loanTerms: Loan`), and enter the return value of the function as a `string`.

   TypeScript

   ```typescript
   function calculateInterestOnlyLoanPayment(loanTerms: Loan): string {
   ```

```
    // ...
}
```

2. You'll notice a couple of errors because TypeScript does not recognize the parameters `interestRate` and `principal`. Replace the parameter names in the function with properties of the `Loan` object. (For example, `loanTerms.interestRate`).

TypeScript

```typescript
function calculateInterestOnlyLoanPayment(loanTerms: Loan): string {
    // Calculates the monthly payment of an interest only loan
    let interest = loanTerms.interestRate / 1200;   // Calculates the Monthly
Interest Rate of the loan
    let payment;
    payment = loanTerms.principal * interest;
    return 'The interest only loan payment is ' + payment.toFixed(2);
}
```

3. Enter the `interest` and `payment` variables in the `calculateInterestOnlyLoanPayment` function as `numbers`.

4. Test the `calculateInterestOnlyLoanPayment` function to verify that it is working correctly. Remember that you must now pass the parameters to the function in the form of a `Loan` object.

TypeScript

```typescript
let interestOnlyPayment = calculateInterestOnlyLoanPayment({principal: 30000,
interestRate: 5});
console.log(interestOnlyPayment);     //* Returns "The interest only loan
payment is 125.00"
```

5. Locate `TODO: Update the calculateConventionalLoanPayment function`. Update the `calculateConventionalLoanPayment` function, this time replacing the three parameters with an object of type `ConventionalLoan`, and enter the return value of the function as a `string`. Make any remaining updates to the implementation of the `calculateConventionalLoanPayment` function.

TypeScript

```typescript
function calculateConventionalLoanPayment(loanTerms: ConventionalLoan):
```

```
string {
   // Calculates the monthly payment of a conventional loan
   let interest: number = loanTerms.interestRate / 1200;   // Calculates the
Monthly Interest Rate of the loan
   let payment: number;
   payment = loanTerms.principal * interest / (1 - (Math.pow(1/(1 + inter-
est), loanTerms.months)));
   return 'The conventional loan payment is ' + payment.toFixed(2);
}
```

6. Test the `calculateConventionalLoanPayment` function to verify that it is working correctly. Remember that you must now pass the parameters to the function in the form of a `ConventionalLoan` object.

TypeScript

```typescript
let conventionalPayment = calculateConventionalLoanPayment({principal: 30000,
interestRate: 5, months: 180});
console.log(conventionalPayment);     //* Returns "The conventional loan pay-
ment is 237.24"
```

Lab solution

View the final version of the code by entering the following at the command prompt.

Bash

```bash
cd ../m03-end
code .
```

Open the file **module03.ts** to see the solution to this lab. See the **Lab setup** section above for more information about setting up your development environment to run the solution.

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆