Lab - Organize code with namespaces

20 minutes

In this lab, you'll try out different ways to organize your code using TypeScript.

Exercise 1

The TypeScript code in the project contains two interfaces: `Loan` and `ConventionalLoan`. It also contains three functions:

- `calculateInterestOnlyLoanPayment`, which calculates the payment for an interest only loan.
- `calculateConventionalLoanPayment`, which calculates the payment for a conventional loan.
- `calculateInterestRate`, a worker function that calculates the monthly interest rate of the loan.

The `calculateInterestOnlyLoanPayment` and `calculateConventionalLoanPayment` functions accept `principle` 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
`principle`	The principle 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'll organize the code using namespaces in a single TypeScript file.

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

```Bash
git clone https://github.com/MicrosoftDocs/mslearn-typescript
cd mslearn-typescript/code/module-08/m08-start
code .
```

2. In the starting workspace, open the file **module08_main.ts** the TypeScript editor.

3. Locate `TODO Create a the Loans namespace.`

4. Create a new namespace called `Loans`.

5. Move the `Loan` and `ConventionalLoan` interfaces into the `Loans` namespace.

6. Update the `Loan` and `ConventionalLoan` interfaces so they are visible outside of the `Loans` namespace.

```TypeScript
namespace Loans {
   export interface Loan {
       principle: number,
       interestRate: number        //* Interest rate percentage (eg. 14 is
14%)
   }
   export interface ConventionalLoan extends Loan {
       numberOfMonths: number      //* Total number of months
   }
}
```

7. Locate `TODO Create LoanPrograms namespace.`

8. Create a new namespace called `LoanPrograms`.

9. Move the three functions into the `LoanPrograms` namespace.

```TypeScript
namespace LoanPrograms {
   /*  TODO Update the calculateInterestOnlyLoanPayment function. */
   function calculateInterestOnlyLoanPayment(loanTerms: Loan): string {
       let payment: number;
```

```
        payment = loanTerms.principle *
calculateInterestRate(loanTerms.interestRate);
        return 'The interest only loan payment is ' + payment.toFixed(2);
    }
    /*  TODO Update the calculateConventionalLoanPayment function. */
    function calculateConventionalLoanPayment(loanTerms: ConventionalLoan):
string {
        let interest: number = calculateInterestRate(loanTerms.interestRate);
        let payment: number;
        payment = loanTerms.principle * interest / (1 - (Math.pow(1/(1 + in-
terest), loanTerms.months)));
        return 'The conventional loan payment is ' + payment.toFixed(2);
    }
    function calculateInterestRate (interestRate: number): number {
        let interest: number = interestRate / 1200;
        return interest
    }
}
```

10. Locate `TODO Update the calculateInterestOnlyLoanPayment function.`

11. In the `calculateInterestOnlyLoanPayment` function:
    a. Update the reference to the `Loan` interface so it includes the `Loans` namespace.
    b. Make the function visible outside of the `LoanPrograms` namespace.

TypeScript

```
export function calculateInterestOnlyLoanPayment(loanTerms: Loans.Loan):
string {
    let payment: number;
    payment = loanTerms.principle *
calculateInterestRate(loanTerms.interestRate);
    return 'The interest only loan payment is ' + payment.toFixed(2);
}
```

12. Locate `TODO Update the calculateConventionalLoanPayment function.`

13. In the `calculateConventionalLoanPayment` function:
    a. Update the reference to the `ConventionalLoan` interface so it includes the `Loans` namespace.
    b. Make the function visible outside of the `LoanPrograms` namespace.

TypeScript

```
 // Calculates the monthly payment of a conventional loan
 export function calculateConventionalLoanPayment(loanTerms:
Loans.ConventionalLoan): string {
     let interest: number = calculateInterestRate(loanTerms.interestRate);
     let payment: number;
     payment = loanTerms.principle * interest / (1 - (Math.pow(1/(1 + inter-
est), loanTerms.months)));
     return 'The conventional loan payment is ' + payment.toFixed(2);
 }
```

14. Locate `TODO Update the function calls`. ADD THIS TO THE START AND END CODE.

15. Add the namespace `LoanPrograms` to the name of the functions.

TypeScript

```
let interestOnlyPayment = LoanPrograms.calculateInterestOnlyLoanPayment({princi-
ple: 30000, interestRate: 5});
let conventionalLoanPayment = LoanPrograms.calculateConventionalLoanPayment({prin-
ciple: 30000, interestRate: 5, months: 180});
```

1. Save, compile, and test your work.

Exercise 2

In this exercise, you'll reorganize the namespaces into multiple TypeScript files.

1. Continue your project from Exercise 1.

2. Create two new TypeScript files in your workspace, **module08_loans.ts** and **module08_loan-programs.ts**.

3. Move the `Loans` namespace from **module08_main.ts** to **module08_loans.ts**.

4. Move the `LoanPrograms` namespace from **module08_main.ts** to **module08_loan-programs.ts**.

5. At the top of **module08_loan-programs.ts**, add a `reference` statement that describes the relationship between the interfaces in **module08_loans.ts** and **module08_loan-programs.ts**.

```
TypeScript
/// <reference path="module08_loans.ts" />
```

6. In **module08_main.ts**, locate `TODO Add reference paths.`

7. Add the `reference` statements that describe the relationship between **module08_loans.ts**, **module08_loan-programs.ts**, and **module08_main.ts**.

```
TypeScript
/// <reference path="module08_loans.ts" />
/// <reference path="module08_loan-programs.ts" />
```

8. At the command prompt, run the following command to compile all the dependent **.ts** files and create a single JavaScript file named **main.js**.

```
Bash
tsc --outFile main.js module08_main.ts
```

9. Test your work by running the **main.js** file.

Lab solution

Clone the ending repository by entering the following at the command prompt.

```
Bash
git clone https://github.com/MicrosoftDocs/mslearn-typescript
cd mslearn-typescript/code/module-08/m08-end
code .
```

Open each **.ts** file 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? ☆ ☆ ☆ ☆ ☆