✓ 100 XP

Exercise - Organize code by using single file namespaces

5 minutes

You can implement namespaces within a single TypeScript file or across multiple TypeScript files.

Note

You must use an IDE, such as Visual Studio Code, to implement multi-file namespaces. It is not possible to do this in the TypeScript Playground.

Complete the steps to define a single file namespace:

1. Open a new Visual Studio Code workspace.
2. Create a new file called **module08_exercise.ts**.
3. Define a new namespace by using the `namespace` keyword followed by the namespace name. You can define as many namespaces as needed within a single TypeScript file. At the top of the file, define two namespaces named `Greetings` and `GreetingsWithLength`.

TypeScript

```
namespace Greetings {
}
namespace GreetingsWithLength {
}
```

4. You can now define functions and classes inside of the namespace definition. All components defined within the namespace are scoped to the namespace and removed from the global scope. Add a new function called `returnGreeting` to the `Greetings` namespace. This function returns the value of a parameter to the console.

TypeScript

```typescript
namespace Greetings {
    function returnGreeting (greeting: string) {
        console.log(`The message from namespace Greetings is ${greeting}.`);
    }
}
```

5. Add two new functions, `returnGreeting` and `getLength`, to the `GreetingsWithLength` namespace. The `returnGreeting` function uses the helper function `getLength` to determine the length of the greeting before returning the message to the console.

TypeScript

```typescript
namespace GreetingsWithLength {
    function returnGreeting (greeting: string) {
        let greetingLength = getLength(greeting);
        console.log(`The message from namespace GreetingsWithLength is
${greeting}. It is ${greetingLength} characters long.`);
    }
    function getLength(message: string): number {
        return message.length
    }
}
```

6. If you want to make a function or class available to code outside of the namespace, add the `export` keyword before its name. If you omit the `export` keyword, the component is only available inside the namespace. You can do this if defining components that should only be directly accessible to other components within the namespace. Add the `export` keyword to the `returnGreeting` function in both namespaces. The `getLength` function should not be accessible outside of the `GreetingsWithLength` namespace so omit the `export` keyword.

TypeScript

```typescript
namespace Greetings {
    export function returnGreeting (greeting: string) {
        console.log(`The message from namespace Greetings is ${greeting}.`);
    }
}
namespace GreetingsWithLength {
    export function returnGreeting (greeting: string) {
        let greetingLength = getLength(greeting);
        console.log(`The message from namespace GreetingsWithLength is
${greeting}. It is ${greetingLength} characters long.`);
    }
    function getLength(message: string): number {
        return message.length
    }
}
```

7. To use a class or function within a namespace, prefix the component name with the namespace name. Try calling the `returnGreeting` function without specifying the namespace. This returns an error because both `returnGreeting` functions are out of scope in the global namespace. Now, try prefixing `Greetings` or `GreetingsWithLength` to the `returnGreeting` function. This provides access to the function within each respective namespace.

TypeScript

```
returnGreeting('Hello');                     // Returns error
Greetings.returnGreeting('Bonjour');         // OK
GreetingsWithLength.returnGreeting('Hola');  // OK
```

Organize code using nested namespaces

You can also nest namespaces within namespaces, providing even more options for organizing your code.

Continue working in the code editor.

1. Create a new namespace called `AllGreetings` and then move the `Greetings` and `GreetingsWithLength` namespaces that you created previously inside it. Add the `export` keyword before both namespace names. This allows the namespace to be accessible outside of the `AllGreetings` namespace.

TypeScript

```
namespace AllGreetings {
    export namespace Greetings {
        export function returnGreeting (greeting: string) {
            console.log(`The message from namespace Greetings is ${greet-
ing}.`);
        }
    }
    export namespace GreetingsWithLength {
        export function returnGreeting (greeting: string) {
            let greetingLength = getLength(greeting);
            console.log(`The message from namespace GreetingsWithLength is
${greeting}. It is ${greetingLength} characters long.`);
        }
        function getLength(message: string): number {
            return message.length
        }
    }
}
```

2. To call the functions, start by typing the outermost namespace name, `AllGreetings`, a period, and then the next level in the namespace hierarchy, `Greetings` or `GreetingsWithLength`. Continue down the hierarchy until you reach the function name.

TypeScript

```
AllGreetings.Greetings.returnGreeting('Bonjour');        // OK
AllGreetings.GreetingsWithLength.returnGreeting('Hola');  // OK
```

Defining a namespace alias

TypeScript creates an easy-to-navigate hierarchy of nested namespaces. But, as your nested namespaces become more complex, you may want to create an alias to shorten and simplify your code. To do this, use the `import` keyword.

Continue working in the code editor.

1. Type `import greet = AllGreetings.Greetings`. This defines a new alias called `greet` that represents `AllGreetings.Greetings`. You can now use `greet` in place of `AllGreetings.Greetings` in your code.

TypeScript

```
import greet = AllGreetings.Greetings;
greet.returnGreeting('Bonjour');
```

Compiling single file namespaces

You compile a single file namespace the same way that you compile any other TypeScript file. Because namespaces are a TypeScript-only construct, they are removed from the resulting JavaScript code and converted into variables that nest as necessary to form namespace-like objects.

Next unit: Organize code by using multi-file namespaces

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆