IS 312: Web Design and Programming

PE07: Programming Exercise

> Revised: Fall 2022
> By: Tj Scharlau

Objective: Creating Interfaces and Classes with generic types

For this exercise you will be using the materials from this week's course materials, in addition to online resources (self-located), to implement generic types via interfaces and classes.

Task #1: Declare an interface using Generic Type

When creating an interface declaration, you can use a generic variable instead of type notations in TypeScript.

1. Open your IDE and start a new file.
2. Declare a simple interface called `Identity` with two properties:
    - `value` and `message`, which use two *generic type* variables (`T`, and `U`) for the *property types*.
3. Paste your source code into the Lab Report document, under `Task 1-2`.
4. Declare two variables, using the `Identity interface` as an object type. They should use the following parameters in the declaration:
    - **number** & **string** for `object one`
    - **string** & **number** for `object two`
    - The parameters should be listed in opposite order for each object.
5. Paste your source code into the lab report, under `Task 1-4`.

Task #2: Generic Interface as a function type in Declarations

For this task you will continue to build on the previous source code file, by completing the following steps:

1. Declare a new interface called `ProcessIdentity` that includes the generic signature of a method, `(value: T, message: U): T`. Notice that this method doesn't have a name.
    - Why do you think you would write it this way?
    - What benefits does it have?
2. Paste your source code into the report under `Task 2-1`.
3. Declare a function called `processIdentity` that has the same type signature as the `ProcessIdentity` interface. It should output the message to the console, and return the contents of value.
4. Paste your source code into your lab report under `Task 2-3`.
5. Declare a function type variable called `processor` with the `ProcessIdentity` *interface* as the variable type, passing in `number` for the `T` *type* and **string** for the `U` *type*.
6. Assign the `ProcessIdentity` function to it.
7. What does this allow you to do?
    - How does TypeScript handle it?
    - Place your answers in your in your lab report under `Task 2-7`
8. Set two variables: `returnNumber1` and `returnNumber2` that call the `processor` function with the following parameters:
    - `returnNumber1` should pass 100 and 'Hello' to the `processor` function in that order.

    - `returnNumber2` should pass 'Hello' and 100 to the `processor` function in that order.
9. What results do these two items return when executed?
    - Why?
    - Place your response under `Task 2-9` in your lab report.

Task #3: Declare a generic interface as a class type

We will now declare a **generic interface** as a class type. First start with an empty source code file.

1. Declare an *interface* called `ProcessIdentity` that has *two properties*: `value` and `message`, and *two generic type variables* `T` and `U`, for the **property types**.
2. Add a generic signature of a method called `process` that returns a value of type `T`.
3. Paste your source code into your lab report under `Task 3-2`.
4. Define a **generic class** called `processIdentity` that implements the *interface* from `Task 3-1`.
5. Name the variable types in the `processIdentity class` `X` and `Y`.
6. Why can you use `X` and `Y` here when the interface uses `T` and `U`?
    - Place your answer in Task 3 – 6 in your lab report.
7. Declare a new variable named `process` and assign a new `processIdentity` object to it, passing in `number` and `string` for the `X` and `Y` variable types, and a `number` and `string` as the argument values.
8. Paste your source code into your lab report under `Task 3-7`.
9. Execute the following commands:
    - `processor.process();`
    - `processor.value = '100';`
10. What did the above two commands produce when you executed them?
    - Why do you believe that you got those results?
    - Place your response in `Task3-10` in your lab report.

Task 4: Defining a Generic Class

For this task you will be defining a generic class. You will start from a blank file and do the following:

1. Declare `processIdentity` as a generic class, without implementing the `ProcessIdentity` interface. This will include the appropriate private values, a `constructor`, and the method `getIdentity()`, that all use generics in an appropriate manner. The variables in the object should be `value` and `message`. The method `getIdentity()` should *output* the `message` variable to the **console**, and return the `value` variables contents.

2. Declare a new variable named `processor` and assign a **new** `processIdentity` *object* to it, passing in `number` and `string` for the `T` and `U` variable types, and a `number` and `string` as the argument values.

3. Paste your source code into your lab report under `Task 4 – 1`.

4. Run the following code:

```
processor.getIdentity();
```

5. What output does that return? Place your response under `Task 4 - 5`

Submission

To submit this assignment, you should submit your completed lab report to Brightspace.

Task 1-2

```
/*
** CLASS:   IS 312: Web Design and Programming
** ASGN:    PE07: Programming Exercise
** QUARTER: FALL 2022/23
** STUDENT: Thaddeus Thomas
** DATE:    11 DEC 2022
*/

/*Task #1
Paste your source code into the lab report under `1-4`
*/
interface Identity<T,U> {
    process(value: T, message: U): void;
};

let obj1: Identity<number, string>;
let obj2: Identity<string, number>;

// Task #2
interface ProcessIdentity<T, U> {
    (value: T, message: U): T
};

/* Task 2-1
Why do you think you would write it this way?
Probably because its used as a template. Either that or its used a function in
comparisons

What benefits does it have?
You could parse strings with it relatively easily for finding duplicates.
*/

// Paste your source code into your lab report under Task 2-3
function processIdentity <T, U> (arg: T){
     (value: T, message: U): T
     return T
    };
    console.log(processIdentity<U>

/*
let: ProcessIdentity<number, string> = myProcessIdentity;
let myprocessor("hello", 1)
console.log('return value is ')
*/

/*
 Task 2-7
 How does Typescript handle it?
 - No Clue
```

```
*/
let processor = ProcessIdentity<number, string>();

/*
return number1 and return number2

*/
```

Task #3

```
interface ProcessIdentity<T, U> {
    process(value: T, message: U): void;
};

class processIdentity<X, Y> implements ProcessIdentity<X, Y> {
    process(value: X, message: Y): void {
        console.log(`Key = ${value}, val= ${message}`);
    }
}

let process: processIdentity<number, string> = new processIdentity();
process.process(100, 'Stuff');
```

Task #4

```
class ProcessIdentity<T,U>
{
    private value: T;
    private message: U;

    getIdentity(value: T, message: U): void {
        this.value = value;
        this.message = message;
    }

    display(): constructor {
        console.log(`value = ${this.value}, message = ${this.message}`);
    }
}

let processor = new processIdentity<number, string>();
processor.setKeyValue(100, "Stuff");
processor.getIdentity();
```