TypeScript Getters and Setters

Summary: in this tutorial, you learn how to use the TypeScript getters and setters.

Introduction to TypeScript Getters and Setters

The following shows a simple `Person` class with three properties: `age` , `firstName` , and `lastName` :

```
class Person {
    public age: number;
    public firstName: string;
    public lastName: string;

    constructor(age: number, firstName: string, lastName: string) {
        this.age = age;
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

To access any property of the `Person` class, you can do this:

```
let person = new Person(22, 'John','Doe');
person.age = 23;
```

Suppose that you assign a value that comes from user input to the `age` property:

```
person.age = inputAge;
```

The `inputAge` can be any number. To ensure the validity of the age, you can carry a check before the assignment as follows:

```
if( inputAge > 0 && inputAge < 200 ) {
    person.age = inputAge;
}
```

Using this check all over places is redundant and tedious.

To avoid repeating the check, you can use setters and getters. The getters and setters allow you to control access to the properties of a class.

For each property:

- A getter method returns the value of the property's value. A getter is also called an accessor.
- A setter method updates the property's value. A setter is also known as a mutator.

A getter method starts with the keyword `get` and a setter method begins with the keyword `set`.

```
class Person {
  private _age: number;
  private _firstName: string;
  private _lastName: string;

  constructor(age: number, firstName: string, lastName: string) {
    this._age = age;
    this._firstName = firstName;
    this._lastName = lastName;
  }

  public get age() {
    return this._age;
  }

  public set age(theAge: number) {
    if (theAge <= 0 || theAge >= 200) {
      throw new Error('The age is invalid');
    }
    this._age = theAge;
  }

  public getFullName(): string {
    return `${this._firstName} ${this._lastName}`;
  }
}
```

How it works.

- First, change the access modifiers of the `age`, `firstName`, and `lastName` properties from `public` to `private`.
- Second, change the property `age` to `_age`.
- Third, create getter and setter methods for the `_age` property. In the setter method, check the validity of the input age before assigning it to the `_age` property.

Now, you can access the `age` setter method as follows:

```
let person = new Person(22, 'John', 'Doe');
person.age = 23;
```

Notice that the call to the setter doesn't have parentheses like a regular method. When you call `person.age`, the `age` setter method is invoked. If you assign an invalid `age` value, the setter will throw an error:

```
person.age = 0;
```

Error:

```
Error: The age is invalid
```

When you access the `person.age`, the `age` getter is invoked.

```
console.log(person.age);
```

The following adds the getters and setters to the `firstName` and `lastName` properties.

```
class Person {
  private _age: number;
  private _firstName: string;
  private _lastName: string;

  constructor(age: number, firstName: string, lastName: string) {
    this._age = age;
    this._firstName = firstName;
    this._lastName = lastName;
  }

  public get age() {
    return this._age;
  }

  public set age(theAge: number) {
    if (theAge <= 0 || theAge >= 200) {
      throw new Error('The age is invalid');
    }
    this._age = theAge;
  }

  public get firstName() {
    return this._firstName;
  }

  public set firstName(theFirstName: string) {
    if (!theFirstName) {
      throw new Error('Invalid first name.');
    }
    this._firstName = theFirstName;
  }

  public get lastName() {
    return this._lastName;
  }

  public set lastName(theLastName: string) {
    if (!theLastName) {
      throw new Error('Invalid last name.');
    }
    this._lastName = theLastName;
  }

  public getFullName(): string {
    return `${this._firstName} ${this._lastName}`;
  }
}
```

More TypeScript Getters/Setters Examples

As you can see from the code, the setters are useful when you want to validate the data before assigning it to the properties. In addition, you can perform complex logic.

The following shows how to create the `fullname` getter and setter.

```
class Person {
  private _age: number;
  private _firstName: string;
  private _lastName: string;

  constructor(age: number, firstName: string, lastName: string) {
    this._age = age;
    this._firstName = firstName;
    this._lastName = lastName;
  }

  public get age() {
    return this._age;
  }

  public set age(theAge: number) {
    if (theAge <= 0 || theAge >= 200) {
      throw new Error('The age is invalid');
    }
    this._age = theAge;
  }

  public get firstName() {
    return this._firstName;
  }
```

```
    public set firstName(theFirstName: string) {
      if (!theFirstName) {
        throw new Error('Invalid first name.');
      }
      this._firstName = theFirstName;
    }

    public get lastName() {
      return this._lastName;
    }

    public set lastName(theLastName: string) {
      if (!theLastName) {
        throw new Error('Invalid last name.');
      }
      this._lastName = theLastName;
    }

    public get fullName() {
      return `${this.firstName} ${this.lastName}`;
    }

    public set fullName(name: string) {
      let parts = name.split(' ');
      if (parts.length != 2) {
        throw new Error('Invalid name format: first last');
      }
      this.firstName = parts[0];
      this.lastName = parts[1];
    }
  }
```

How it works.

- The getter method returns the concatenation of the first name and last name.
- The setter method accepts a string as the full name with the format: `first last` and assign the first part to the first name property and the second part to the last name property.

Now, you can access the `fullname` setter and getter like a regular class property:

```
let person = new Person(22, 'Jane', 'Doe');


person.fullName = 'Jane Smith';
console.log(person.fullName); //    "Jane Smith"
```

Summary

- Use TypeScript getters/setters to control the access properties of a class.
- The getter/setters are also known as accessors/mutators.