JavaScript Private Fields

Summary: in this tutorial, you'll learn about JavaScript private fields and how to use them effectively.

Introduction to the JavaScript private fields

ES2022 allows you to define private fields for a class. To define a private field, you prefix the field name with the `#` sign.

For example, the following defines the `Circle` class with a private field `radius`:

```
class Circle {
  #radius;
  constructor(value) {
    this.#radius = value;
  }
  get area() {
    return Math.PI * Math.pow(this.#radius, 2);
  }
}
```

In this example:

- First, define the private field `#radius` in the class body.
- Second, initialize the `#radius` field in the constructor with an argument.
- Third, calculate the area of the circle by accessing the `#radius` private field in the getter method.

The following creates a new instance of the `Circle` class and calculates its area:

```
let circle = new Circle(10);
console.log(circle.area); // 314.1592653589793
```

Because the `#radius` is a private field, you can only access it inside the `Circle` class. In other words, the `#radius` field is invisible outside of the `Circle` class.

Using getter and setter to access private fields

The following redefines the `Circle` class by adding the `radius` getter and setter to provide access to the `#radius` private field:

```
class Circle {
  #radius = 0;
  constructor(radius) {
    this.radius = radius; // calling setter
  }
  get area() {
    return Math.PI * Math.pow(this.#radius, 2);
  }
  set radius(value) {
    if (typeof value === 'number' && value > 0) {
      this.#radius = value;
    } else {
      throw 'The radius must be a positive number';
    }
  }
  get radius() {
    return this.#radius;
  }
}
```

How it works.

- The `radius` setter validates the argument before assigning it to the `#radius` private field. If the argument is not a positive number, the `radius` setter throws an error.
- The `radius` getter returns the value of the `#radius` private field.

- The constructor calls the `radius` setter to assign the argument to the `#radius` private field.

Private fields and subclasses

Private fields are only accessible inside the class where they're defined. Also, they're not accessible from the subclasses. For example, the following defines the `Cylinder` class that extends the `Circle` class:

```
class Cylinder extends Circle {
  #height;
  constructor(radius, height) {
    super(radius);
    this.#height = height;

    // cannot access the #radius of the Circle class here
  }
}
```

If you attempt to access the `#radius` private field in the `Cylinder` class, you'll get a `SyntaxError`.

The in operator: check private fields exist

To check if an object has a private field inside a class, you use the `in` operator:

```
fieldName in objectName
```

For example, the following adds the `hasRadius()` static method to the `Circle` class that uses the `in` operator to check if the `circle` object has the `#radius` private field:

```
class Circle {
  #radius = 0;
  constructor(radius) {
    this.radius = radius;
  }
```

```
  get area() {
    return Math.PI * Math.pow(this.radius, 2);
  }
  set radius(value) {
    if (typeof value === 'number' && value > 0) {
      this.#radius = value;
    } else {
      throw 'The radius must be a positive number';
    }
  }
  get radius() {
    return this.#radius;
  }
  static hasRadius(circle) {
    return #radius in circle;
  }
}

let circle = new Circle(10);

console.log(Circle.hasRadius(circle));
```

Output:

```
true
```

Static private fields

The following example shows how to use a static private field:

```
class Circle {
  #radius = 0;
  static #count = 0;
  constructor(radius) {
    this.radius = radius; // calling setter
    Circle.#count++;
  }
```

```
  get area() {
    return Math.PI * Math.pow(this.radius, 2);
  }
  set radius(value) {
    if (typeof value === 'number' && value > 0) {
      this.#radius = value;
    } else {
      throw 'The radius must be a positive number';
    }
  }
  get radius() {
    return this.#radius;
  }
  static hasRadius(circle) {
    return #radius in circle;
  }
  static getCount() {
    return Circle.#count;
  }
}

let circles = [new Circle(10), new Circle(20), new Circle(30)];

console.log(Circle.getCount());
```

How it works.

First, add a private static field `#count` to the `Circle` class and initialize its value to zero:

```
static #count = 0;
```

Second, increase the `#count` by one in the constructor:

```
Circle.#count++;
```

Third, define a static method that returns the value of the `#count` private static field:

```
static getCount() {
    return Circle.#count;
}
```

Finally, create three instances of the `Circle` class and output the `count` value to the console:

```
let circles = [new Circle(10), new Circle(20), new Circle(30)];
console.log(Circle.getCount());
```

Summary

- Prefix the field name with `#` sign to make it private.
- Private fields are accessible only inside the class, not from outside of the class or subclasses.
- Use the `in` operator to check if an object has a private field.