Difference between Interface and Abstraction in JavaScript

Abstract classes

JavaScript lacks a native implementation of abstract classes, a feature present in some other programming languages such as Java or Python. Nevertheless, a similar outcome can be achieved by combining constructor functions, prototypes, and the throw statement.

Example:

```
class Shape {
  constructor(name) {
    if(this.constructor == Shape) {
      throw new Error("Class is of abstract type and can't be instantiated");
    };
    if(this.getArea == undefined) {
       throw new Error("getArea method must be implemented");
    };
    this.name = name;
  }
}


class Rectangle extends Shape {
  constructor(name, length, width){
    super(name);
    this.length = length;
    this.width = width;
  }
}
const myShape = new Shape('My shape'); // This will throw an Error
const smallRectangle = new Rectangle("Small Rectangle", 3, 5) // This will throw an error
```

Interface

When it comes to interfaces, JavaScript lacks built-in support, unlike some other programming languages. However, interface-like behavior can be simulated using object literals or by defining methods that must be implemented by objects. The following example demonstrates using an object literal:

Example:

```
const myInterface = {
  method1: function () {
    throw new Error("Method1 not implemented");
  },
  method2: function () {
    throw new Error("Method2 not implemented");
  }
 };


// Implement the interface in a concrete object
class MyImplementation {
  method1() {
    console.log("Method1 implemented");
  }
  method2() {
    console.log("Method2 implemented");
  }
}
// Create an instance of the implemented object
const myObject = new MyImplementation();


myObject.method1(); // Outputs: Method1 implemented
myObject.method2(); // Outputs: Method2 implemented
```