Name: Radwa Hassan Sadek
Track: Open Source

Interface	Abstract Class
All members are abstract.	Some members are abstract, and some are fully implemented.
Interfaces support multiple inheritances.	Abstract class does not support multiple inheritances.
If a class uses an interface, it must implement all the methods and properties declared in the interface.	If a class uses an abstract class, it can decide which methods and properties to use as long as they are not declared as abstract.
It cannot be instantiated	They cannot be instantiated, but they can be used as subclasses.
It cannot be used to define data. An interface is a class that is used to define methods and properties.	abstract classes are allowed to define data.

1) Abstract class in JavaScript:

```
class Person {
    constructor() {
        if (this.constructor == Person) {
            throw new Error("Your Error Message...");
        }
    }
    info() {
        throw new Error(" Added abstract Method has no implementation");
    }
}
class Teacher extends Person {
    info() {
        console.log("I am a Teacher");
    }
}
var teacher1 = new Teacher();
teacher1.info();
```

2) Interface in JavaScript:

JavaScript doesn’t support interfaces. there's no notion of "this class must have these functions" (that is, no interfaces per se), because:

1. JavaScript inheritance is based on objects, not classes.
2. JavaScript is an extremely dynamically typed language -- you can create an object with the proper methods, which would make it conform to the interface, and then undefine all the stuff that made it conform.

Instead, JavaScript uses what's called duck typing. (If it walks like a duck, and quacks like a duck, as far as JS cares, it's a duck.) If your object has quack(), walk(), and fly() methods, code can use it wherever it expects an object that can walk, quack, and fly, without requiring the implementation of some "Duckable" interface. The interface is exactly the set of functions that the code uses (and the return values from those functions), and with duck typing, you get that for free.

```
module.exports = class MyInterface {

    /**
     * @return {string}
     */
    firstMethod() {}
}
const MyInterface = require('./MyInterface')

module.exports = class MyImplementation extends MyInterface {

    myVar

    constructor(theVar) {
        this.myVar = theVar
    }

    /**
     * @inheritDoc
     */
    firstMethod() {
        return 'secondMethod'
    }
}

/**
 * {MyInterface} myObject
 */
const myFunction = () => {
    return new MyImplementation()
}
```