Difference between Abstract & Interface classes

1. A regular class can only inherit from one Abstract class, but it can inherit from multiple interfaces.
2. An interface can only declare methods and properties; But an Abstract class in addition to them can have methods and properties with full code.
3. The elements in the Abstract class can have an access level like a normal class; But interfaces do not have this feature.
4. When you add a method to the Abstract class, it is automatically applied to all subclasses; But in the Interface, if you add a method, you must apply it to all subclasses.
5. Abstract classes, like regular classes, can have fields and other elements (such as constants); While an interface does not have this feature. The abstract class can also contain a constructor, but the interface cannot.
6. Abstract is one of the types of classes; But Interface is not a class.
7. The interface can only inherit from the interface, but the abstract class can inherit from the interface, the Abstract class, or other classes.

Inheritance in function constructor

Here, we will discuss inheriting a constructor function in JavaScript. Constructor functions define the prototype of the properties an object will contain. Using the constructor function, we can create a new object after passing the required parameters.

Inheriting a previously defined constructor function means using the parameters of the previously defined function along with adding some new parameters to the newly defined constructor function. For this, we need to use the call() function which allows us to call a function defined somewhere else in the current context.

Parameter values:

- **function:** This is a constructor function from which we want to inherit the parameters in a new constructor function.
- **this**: The values of parameters that will use this keyword while calling myFunction.
- **property1, property2, …, propertyN:** The parameters that are to be inherited in the new constructor function.

Return type: A constructor function or the function which has inherited its properties does not have any return type. It specifies the **prototype** of the properties an object will contain which are created from that constructor function.

Example: Here, we create an ‘Employee’ constructor function. A new ‘Developer’ constructor function is created that will inherit the basic properties of ‘Employee’ as well as will contain some new properties and describes the object created with a new keyword to create an instance of an object that has a constructor function.

HTML

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <title>
        JavaScript inheritance of constructor functions
     </title>
</head>

<body>
    <script>
        function Employee(name, age, gender, id) {
            this.name = name;
            this.age = age;
            this.gender = gender;
            this.id = id;
        };

        function Developer(name, age, gender, id, specialization) {

            // Calling Employee constructor function
            Employee.call(this, name, age, gender, id);

            // Adding a new parameter
            this.specialization = specialization;
        }

        // Creating objects
        let Employee1 = new Employee("Suraj", 28, "Male", 564);
        let Developer1 = new Developer("Karishma", 31, "Female", 345,
            "Frontend Developer");
        console.log(Employee1);
        console.log(Developer1);
    </script>
</body>

</html>
```

Output:

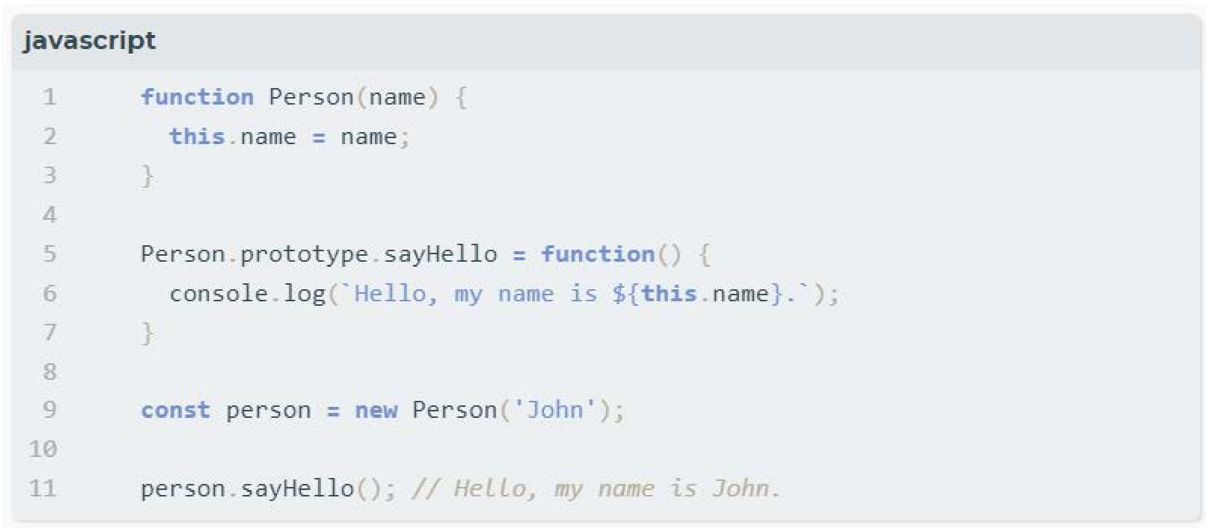
```
Employee {name: 'Suraj', age: 28, gender: 'Male', id: 564}
age: 28
gender: "Male"
id: 564
name: "Suraj"
[[Prototype]]: Object

Developer {name: 'Karishma', age: 31, gender: 'Female', id: 345,
    specialization: 'Frontend Developer'}
age: 31
gender: "Female"
id: 345
name: "Karishma"
specialization: "Frontend Developer"
[[Prototype]]: Object
```

We can observe that the constructor function of Employee is inherited to create a new constructor function Developer which can be used to create objects with new properties along with the inherited properties of the parent constructor.

Constructors Vs Prototype

The difference between constructor and prototype is that the constructor is a function that is used to create an object, while the prototype is an object that contains properties and methods that are inherited by objects created from a constructor.



```javascript
function Person(name) {
  this.name = name;
}

Person.prototype.sayHello = function() {
  console.log(`Hello, my name is ${this.name}.`);
}

const person = new Person('John');

person.sayHello(); // Hello, my name is John.
```

In the example above, we have a constructor function that takes a name parameter and assigns it to the name property of the object. We also have a prototype method called sayHello (), which prints a message to the console. When we create a new Person object using the constructor function, we can call the sayHello() method on it, and it will print the message with the name that we passed into the constructor. The person object can use the sayHello() method because it's inherited from the prototype of the Person constructor function.